LEARNING AND APPLICATIONS OF PARAPHRASTIC REPRESENTATIONS FOR NATURAL LANGUAGE

JOHN WIETING

Language Technologies Institute

Ph.D. Thesis

Language Technologies Institute
Carnegie Mellon University

July 27th 2020

ABSTRACT

Representation learning has had a tremendous impact in machine learning and natural language processing (NLP), especially in recent years. Learned representations provide useful features needed for downstream tasks, allowing models to incorporate knowledge from billions of tokens of text. The result is better performance and generalization on many important problems of interest. Often these representations can also be used in an unsupervised manner to determine the degree of semantic similarity of text or for finding semantically similar items, the latter useful for mining paraphrases or parallel text. Lastly, representations can be probed to better understand what aspects of language have been learned, bringing an additional element of interpretability to our models.

This thesis focuses on the problem of learning *paraphrastic* representations for units of language. These units span from sub-words, to words, to phrases, and to full sentences – the latter being a focal point. Our primary goal is to learn models that can encode arbitrary word sequences into a vector with the property that sequences with similar semantics are near each other in the learned vector space, and that this property transfers across domains.

We first show several effective and simple models, PARAGRAM and CHARAGRAM, to learn word and sentence representations on noisy paraphrases automatically extracted from bilingual corpora. These models outperform contemporary and more complicated models on a variety of semantic evaluations.

We then propose techniques to enable deep networks to learn effective semantic representations, addressing a limitation of our prior work. We found that in order to learn representations for sentences with deeper, more expressive neural networks, we need large amounts of sentential paraphrase data. Since this did not exist yet, we utilized neural machine translation models to create PARANMT-50M, a corpus of 50 million English paraphrases which has found numerous uses by NLP researchers, in addition to providing further gains on our learned paraphrastic sentence representations.

We next propose models for bilingual paraphrastic sentence representations. We first propose a simple and effective approach that outperforms more complicated methods on cross-lingual sentence similarity and mining bitext, and we also show that we can also achieve strong monolingual performance without paraphrase corpora by just using parallel text.

We then propose a generative model capable of concentrating semantic information into our embeddings and separating out extraneous information by viewing parallel text as two different views of a semantic concept. We found that this model has improved performance on both monolingual and cross-lingual tasks. Lastly, we extend this bilingual model to the multilingual setting and show it can be effective on multiple languages simultaneously, significantly surpassing contemporary multilingual models.

Finally, this thesis concludes by showing applications of our learned representations and ParaNMT-50M. The first of these is on generating paraphrases with syntactic control for making classifiers more robust to adversarial attacks. We found that we can generate a controlled paraphrase for a sentence by supplying just the top production of the desired constituent parse – and the generated sentence will follow this structure, filling in the rest of the tree as needed to create the paraphrase. The second application is applying our representations for fine-tuning neural machine translation systems using minimum risk training. The conventional approach is to use BLEU (Papineni et al., 2002), since that is what is commonly used for evaluation. However, we found that using an embedding model to evaluate similarity allows the range of possible scores to be continuous and, as a result, introduces fine-grained distinctions between similar translations. The result is better performance on both human evaluations and BLEU score, along with faster convergence during training.

CONTENTS

INTRODUCTION

Representation learning has had a tremendous impact in machine learning and natural language processing, especially in recent years. Learned representations provide useful features needed for downstream tasks, allowing models to incorporate knowledge from billions of tokens of text. The result is better performance downstream tasks and better generalization. Representations are also useful because they can be probed to better understand what aspects of language have been learned. Vector representations can additionally be useful for finding semantically similar items quickly, an important aspect for text retrieval and mining.

Word representations have had an especially rich history in natural language processing. A common form of these representations are mathematical objects like vectors, points in a high dimensional space (Deerwester et al., 1990; Turney, 2001; Bengio et al., 2003; Turian et al., 2010; Mikolov et al., 2013b). With these representations, the relationship between words can be inferred from this space. For instance, words with similar meaning could be near each other or the different dimensions of the word vectors may also correspond to semantic or grammatical properties. It is important to note that there are other types of lexical representations beyond word vectors. Word clusters (Brown et al., 1992; Kneser and Ney, 1993) are another type of representation that can be represented by a vector indicating membership into a specific cluster. There has also been significant work on creating word ontologies or lexicons such as Wordnet (Miller, 1995), Verbnet (Schuler, 2005), Framenet (Baker et al., 1998), and the Proposition Bank (Palmer et al., 2005). These contain rich linguistic information and also can specify the relationships between words. They are also manually created resources and are therefore costlier to construct than automatic approaches. Moreover, they tend to be more sparse than automatic methods that are able to learn representations from data consisting of billions of words incorporating vocabularies that can number in the millions.

There are many techniques for learning word vectors, most of these relying on the *distributional hypothesis* (Harris, 1954) which states that the meaning of words can be inferred by the contexts in which they occur. This was famously restated by Firth (Firth, 1957) that *You* *shall know a word by the company it keeps.* This insight led to many of the distributional word embedding approaches that have made such an impact on the field. Early approaches to learn word embeddings factorize a matrix of co-occurrence count statistics using singular value decomposition (SVD) (Deerwester et al., 1990). Co-occurrence statistics of other relations have also been successfully used like syntactic dependencies (Lin, 1998). In contrast to word counts, methods using dependencies tend to capture more functional similarity as opposed to topical similarity. Often counts are transformed into other statistics to better differentiate surprising from expected co-occurrences such as pointwise mutual information (PMI) (Church and Hanks, 1990). Besides methods focused on matrix factorization, the other main approach to learning word embeddings involve neural networks. These include methods based on language modelling (Bengio et al., 2003) and methods based on predicting co-occurrences (Mikolov et al., 2013b; Pennington et al., 2014). (Faruqui and Dyer, 2014) found that models based on estimating co-occurrences usually lead to better performance owing to incorporating both sides of the context of a word. Interestingly, (Levy and Goldberg, 2014) found that the skip-gram model of (Mikolov et al., 2013b) can be seen as equivalent to factoring a matrix of PMI statistics shifted by $\log(k)$, where k is the number of negative samples used during training.

Recently, contextualized representations (Dai and Le, 2015; McCann et al., 2017; Peters et al., 2018; Devlin et al., 2018) of words have found a great deal of success, often improving on downstream tasks over static word embeddings. In these models, the representations for words change depending on their current context. Therefore, a single word is no longer limited to a specific vector, but instead its representation is generated specifically for its current context. These models are learned through language modelling, machine translation or related objectives such as masked language modelling.

With the success of word representations in many tasks in natural language processing (Clark, 2003; Turian et al., 2010; Bansal et al., 2014), a natural question to ask is what about learning representations for larger units of text such as word bigrams, phrases, sentences, or even paragraphs and documents? These representations pose additional challenged because unlike words, the possible number of instances increases exponentially as the text sequences become larger. Therefore, developing ontologies or borrowing techniques from learning distributional word vectors is infeasible, and therefore different approaches to learning these representations must be used.

Many approaches for learning sentential representations have been proposed in the literature. These include constituent parsing (Klein and Manning, 2003), dependency parsing (McDonald et al., 2005), semantic parsing (Berant and Liang, 2014), semantic role labelling (Punyakanok et al., 2008), and abstract meaning representations (Banarescu et al., 2013). These representations can be seen as analogs to the ontology and lexicons for words since they are also created with the input of human linguists and provide rich syntactic and/or semantic information about the text. However drawbacks include that models must be trained to predict these structures, which can introduce errors when applied to new text, especially if it is out-of-domain from the training data. Further, to apply these representations to downstream tasks, additional processing is required whether they are mined for features or transformed into continuous representations to be incorporated as features into neural models.

Perhaps the earliest work on vector representations of sentences are feature vectors for classification tasks. In these early models, features would be extracted by manually designed feature functions, which could be as simple as the identity of the words in the sentence. Later, with approaches using neural networks in (Collobert and Weston, 2008; Collobert et al., 2011; Socher et al., 2011), features were learned automatically. The first general purpose sentence embeddings however, are relatively recent in the literature (Le and Mikolov, 2014; Kiros et al., 2015). In contrast to the supervised approaches, these embeddings were not trained for any particular goal task, but to provide useful features for *any* task.

There have been many approaches proposed for learning sentence embeddings. These include predicting the next and previous sentences (Kiros et al., 2015), machine translation (Espana-Bonet et al., 2017; Schwenk and Douze, 2017; Schwenk, 2018; Artetxe and Schwenk, 2018b), training on natural language inference (NLI; (Bowman et al., 2015)) data (Conneau et al., 2017), discourse based objectives (Jernite et al., 2017; Nie et al., 2017), and multi-task objectives which include some of the previously mentioned objectives (Cer et al., 2018) as well as additional tasks like constituency parsing (Subramanian et al., 2018).

In this thesis, we focus on learning *paraphrastic* representations for sentences. We hypothesize that a minimum requirement of quality representations is that the distance of the representations in the learned vector space is related to the semantic distance of the underlying text. This intuition motivates many of the algorithms and strategies in this thesis, and distinguishes our work from much of the literature. The problem of learning these representations is difficult because sentences with similar semantics can have significantly different surface forms, while sentences with contradictory semantics can have very similar surface forms. For instance, paraphrases can have large lexical and syntactic differences such as *Other ways are needed.* and *It is necessary to find other means.*, but have similar semantics. Further, subtle lexical changes can drastically change the meaning of the sentences as in *Flights are on sale from New York to Paris.* and *Flights are on sale from Paris to New York.*

The overarching theme of our methods in this thesis is that to adequately distill semantics into vector representations, two views of the underlying concept are needed. When only a single view of a concept is used, for instance when training word representations like GloVe or Word2Vec or sentence representations like Skip-Thought, representations that are close to each other in the learned embedding space will have similar contexts. However, this does not mean that there are paraphrases of each other, as they could simply be related words that share many of the same contexts. For instance, they could be words like *brother* and *sister* or *rampant* and *epidemic*

These views can take a variety of forms (text and video or images, human constructed paraphrases, etc.), but in this thesis we largely make use of parallel data in some fashion (although we do dabble with mined human-constructed paraphrases). Pure parallel data refers to translations, for instance, data consisting of English and Spanish or French and Chinese translations. While parallel data makes up the backbone of our multiple views, we make use of it in different ways. For instance, in Chapters 3, 4, and 5, we use data from the Paraphrase Database (PPDB) (Ganitkevitch et al., 2013). In Chapter 6, we explore first using human-generated paraphrase data, but then proceed to use back-translated parallel text in Chapter 7 in order to automatically generate paraphrase data. Lastly, in Chapters 9 to 11, we explore using parallel data directly. All of these approaches have their trade-offs, but they are united by making use of two views of the same concept.

This thesis is organized into four content sections and eleven content chapters covering not only our approaches to learning *paraphrastic* sentence embeddings, but applications of these embeddings from tasks including the construction of large paraphrase corpora, adversarial paraphrase generation, and fine-tuning machine translation outputs.

The first section covers Chapters 3 to 5, and we discuss our first models for learning *paraphrastic* representations using paraphrase text snippets from the Paraphrase Database (PPDB). In this section, we propose three main models. The first are PARAGRAM word embeddings. These significantly surpassed the state-of-the-art word embedding approaches for measuring similarity and are even the basis for state-of-the-art models today. In the second model, we learn sentence embeddings where our primary evaluation is on a suite of semantic textual similarity (STS) datasets, but we also follow the setting of (Kiros et al., 2015) and show that we can rival their performance with our simpler model. Lastly, we discuss our CHARAGRAM models for learning word and sentence representations. In these representations, we represent a character sequence by a vector containing counts of character n-grams, inspired by Huang et al. (2013). This vector is embedded into a low-dimensional space using a single nonlinear transformation. We find that these CHARAGRAM embeddings outperforms character recurrent neural networks (RNNs), character convolutional neural networks (CNNs), as well as PARAGRAM embeddings. We find that modelling subwords yields large gains in performance for rare words and can easily handle spelling variation, morphology, and word choice.

The second section covers Chapters 6 to 8, where we discuss strategies to improve our paraphrastic sentence embeddings. We do this by both moving away from PPDB and by incorporating deeper and more expressive architectures. In Chapter 6, we discuss the importance of training on sentence pairs, instead of text snippets for performance, and introduce a new recurrent neural architecture that combines aspects of both our simpler averaging models and RNNs. In Chapter 7, we show how backtranslation of bilingual data can be used to creating a corpus of sentential paraphrases on par with paraphrase corpora constructed via manual rewrites. Finally, in Chapter 8, we create a corpus of 50 million sentence paraphrases through back-translation of a large bilingual corpus (Bojar et al., 2016). The subsequent performance gains with our embeddings using ParaNMT-50M, allowed us to outperform all competing systems in the SemEval STS competitions held from 2012-2016 despite not using the training data for these tasks. Moreover, as we show in Chapter 12,

The third section covers Chapters 9 to 11, where we experiment with learning *paraphrastic* sentence representations from multilingual data. In Chapter 9, we show: 1) Using bilingual text can rival performance of using ParaNMT-50M, simplifying the procedure if our focus is exclusively on sentence embeddings, since back-translation is no longer required. 2) Using sub-word embeddings in this setting is more effective than using character n-grams or words for cross-lingual similarity. In Chapter 10, we propose learning paraphrastic sentence embeddings as a source separation problem, leading to s significant boost in representation quality. We treat parallel data as two views of the same semantic information, but with different surface forms. We then propose a deep latent variable model, the Bilingual Generative Transformer (BGT) that performs source separation, isolating what the parallel sentences have in common in a latent semantic vector, and explaining what is left over with language-specific latent vectors. We find that the model is effective, pushing more semantic information into the semantic representation, relative to strong baselines, leading to improvement in all of our evaluations. Lastly, in Chapter 11, we extend the BGT to the multilingual setting with the Multilingual Generative Transformer (MGT). In this chapter, we show how we can collapse to the BGT model to use a single encoder and decoder while training on many languages simultaneously and retaining strong performance – far surpassing that of a translation baseline which is used in contemporary state-of-the-art models.

Finally, the last section, covering Chapters 12 and 13, focus on application of our paraphrase corpus and our *paraphrastic* sentence embeddings. In Chapter 12, we apply our ParaNMT-50Mcorpus and sentence embedding models towards learning controllable paraphrase generation. Specifically we focus on controlling the syntax of the generated sentences. We find that we can learn a model where by just supplying a *parse template*, i.e. the top production of a constituent parse, we can generate a sentence with that syntax. We show that when these syntactic paraphrases are added to training, models become more robust to adversarial examples. In Chapter 13, we use our *paraphrastic* representations, along with a proposed length penalty, for fine-tuning neural machine translation systems using minimum risk training. The conventional approach is to use BLEU (Papineni et al., 2002), since that is what is commonly used for evaluation. However, we found that using an embedding model to evaluate similarity allows the range of possible scores to be continuous and, as a result, introduces fine-grained distinctions between similar translations. The result is better performance on both human evaluations and BLEU score, along with faster convergence during training.

We conclude this thesis with a short summary of the work presented, and discuss several directions for further work in both representation learning, text generation, and their applications.

2 BACKGROUND

2.1 REPRESENTATION LEARNING IN NATURAL LANGUAGE PROCESSING

There are many types of representations in Natural Language Processing (NLP). Representations have been studied at the subword (Sennrich et al., 2016b; Wieting et al., 2016a), word (Bengio et al., 2003), sentence (Kiros et al., 2015), and document level (Le and Mikolov, 2014). There are representations as well for knowledge graphs (Bordes et al., 2014b) that can be used to represent entities and relations and even word sense embeddings (Li and Jurafsky, 2015). In this thesis we focus on subword, word, and sentence embeddings primarily and we will discuss those in greater detail below.

2.1.1 *Subword Embeddings*

2.1.1.1 *Morpheme Based Models*

The simplest approaches append subword features to word embeddings, letting the model learn how to use the subword information for particular tasks. Some added knowledge-based morphological features to word representations (Alexandrescu and Kirchhoff, 2006; El-Desoky Mousa et al., 2013). Others learned embeddings jointly for subword units and words, defining simple compositional architectures (often based on addition) to create word embeddings from subword embeddings (Lazaridou et al., 2013; Botha and Blunsom, 2014; Qiu et al., 2014; Chen et al., 2015c).

2.1.1.2 *Character Based Models*

Later, richer functional architectures to convert character sequences into word embeddings became increasingly popular. Luong et al. (2013) used recursive models to compose morphs into word embeddings, using unsupervised morphological analysis. Ling et al. (2015a) used a bidirectional long short-term memory (LSTM) RNN on characters to embed arbitrary word types, showing strong performance for language modeling and POS tagging. Ballesteros et al. (2015) used this model to represent words for dependency parsing. Several have used character-level RNN architectures for machine translation, whether for representing source or target words (Ling et al., 2015b; Luong and Manning, 2016), or even for generating entire translations character-by-character (Chung et al., 2016).

Schütze (1993) learned representations of character four-grams through singular value decomposition. Reprsentations for words were then constructed by summing these four-gram representations. Wieting et al. (2016a) used character n-grams in order to construct sentence and word embeddings. They found that it improved performance on semantic similarity tasks. Also by using character n-grams, embeddings for nearly any word could be created and so they found their model had the most impact over a word embeddings baseline approach when handling rare words (including mispellings, slang, etc.). The popular FastText embeddings Bojanowski et al. (2017) are also based on this idea and are similar to the n-gram embeddings in Wieting et al. (2016a) where they sum together n-grams of sizes 3 to 6 to obtain an embedding for a word, however instead of training on parallel text, they follow Mikolov et al. (2013a) and train on

2.1.1.3 *Byte Pair Encoding (BPE) and* `Sentencepiece`

Sennrich et al. (2016b) proposed using Byte Pair Encoding (BPE) to split sentences into tokens instead of the most common approaches from prior that segmented sentence by words or characters. When BPE is applied to text, the most frequent symbol pair are merged together into a new symbol. In the beginning, the most frequently paired characters are merged and then this process is continued n times where n is a hyperparameter that specifies the maximum number of merge operations permitted. `Sentencepiece` is an alternative to BPE and also seeks to segment text into its most frequent subwords, however, it models the segmentation as

Pretrained embeddings using BPE have been released and found to be useful for a number of tasks. Heinzerling and Strube (2018) trained BPE embeddings on Wikipedia in 275 languages using a wide set of allowed merge operations and embedding sizes. Wieting et al. (2019b) found that for learning paraphrastic sentence representations, using `Sentencepiece` can perform similarly or better than using character n-gram embeddings, while being both faster to compute and requiring less memory.

Many recent deep embedding models use either BPE or `Sentencepiece` models to segment text (Artetxe and Schwenk, 2018b; Devlin et al., 2018; Raffel et al., 2019). They have been found to be especially useful when working with multiples languages as a subword embedding is significantly more likely to be shared between languages than a word embedding if the languages are related, which can reduce the number of parameters in the model (both in the embedding table and in the projection to compute the logits). They also likely enhance the amount of semantic content in the embeddings since they receive more frequent updates.

2.1.2 *Word Embeddings*

The motivation for learning word representations is based on the distributional hypothesis, that the meaning of words can be captured by their context (Harris, 1954). There are a number of ways in which one can compute representations for words, and these representations can take a number of forms, from binary vectors to continuous representations to difference vectors for each word sense.

The earliest embeddings were based on counts, Salton et al. (1975) created document embeddings, where dimensions were the whole vocabulary and the entries were frequency counts of the words. Word embeddings eventually followed this paradigm (Deerwester et al., 1990) where the context for each word was whether it appeared in a set of documents, with a 1 in entry i if it appeared in document i.

The intuition of most methods for learning word embeddings is to create these vectors (or embeddings) is that similar words have similar contexts (Firth, 1957). Earlier models made use of latent semantic analysis (LSA) (Deerwester et al., 1990). More sophisticated neural models, work originating with (Bengio et al., 2003), then gained popularity (Mikolov et al., 2013a; Pennington et al., 2014). These embeddings were then used in new ways by being tailored to specific downstream tasks (Bansal et al., 2014). Other popular word embeddings include (Lin and Pantel, 2001; Bengio et al., 2003; Collobert and Weston, 2008; Turian et al., 2010; Turney and Pantel, 2010; Collobert et al., 2011).

PPDB has been used along with other resources to learn word embeddings for several tasks, including semantic similarity, language modeling, predicting human judgments, and classification (Yu and Dredze, 2014; Faruqui et al., 2015; Wieting et al., 2015; Mrkšić et al., 2016). Other work has used neural MT architectures and training settings to obtain better word embeddings (Hill et al., 2014a,b).

Finally, it is important to mention that word embeddings can encode harmful biases. For instance it not uncommon for word embeddings trained on a large enough corpora may associate certain groups with certain attributes. These biases can then be propagated through trained models since word embeddings are often used to initialize neural NLP models. Research into mitigating biased outputs is an important and active area of research currently, especially with the widespread use of increasingly large pretrained models.

2.1.3 *Sentence Embeddings*

Researchers have also developed many ways to embed word sequences for NLP. They mostly focus on the question of compositionality: given vectors for words, how should we create a vector for a word sequence? Early work in this area started with simply composing pairs of words or bigrams. In (Mitchell and Lapata, 2008, 2010), they compared a variety of binary compositional operations on word vectors and found that simple point-wise multiplication of explicit vector representations performed very well. Follow-up work by (Blacoe and Lapata, 2012) found again that simple operations such as vector addition performed strongly. Other works like (Zanzotto et al., 2010) and (Baroni and Zamparelli, 2010) also explored composition using models based on operations of vectors and matrices. Later, the efficient neural embeddings of (Mikolov et al., 2013a) also had strong performance on compositional tasks simply by adding the word vectors (Mikolov et al., 2013b). (Hashimoto et al., 2014) introduced an alternative word embedding and compositional model based on predicate-argument structures that does well on two simple composition tasks, including the one introduced by (Mitchell and Lapata, 2010). Many other simple compositional architectures have also been proposed based on distributional semantics (Baroni et al., 2014; Paperno et al., 2014; Polajnar et al., 2015; Tian et al., 2015).

An alternative approach to composition, is to learn deep neural models that compose word embeddings. One of the first neural models for NLP was (Weston et al., 2010)One of the earliest works to do this for sentence representations was (Socher et al., 2011), where the authors trained a recursive neural network (RNN) whose structure is defined by a binarized parse tree. This work was followed up in (Socher et al., 2012, 2013; Irsoy and Cardie, 2014; Wieting et al., 2015). In particular, they trained their RNN as an unsupervised autoencoder. The RNN captures the latent structure of composition. Later work has shown that this model struggles in tasks involving compositionality (Blacoe and Lapata, 2012; Hashimoto et al., 2014).[1]

[1] We also replicated this approach in Chapter 3 and found training to be time-consuming even using low-dimensional word vectors.

Since these early forays into compositional models, many other neural computational models have been proposed: neural bag-of-words models (Kalchbrenner et al., 2014), deep averaging networks (DANs) (Iyyer et al., 2015), feature-weighted averaging (Yu and Dredze, 2015), recursive neural networks using syntactic parses (Socher et al., 2012, 2013; Irsoy and Cardie, 2014), recursive networks based on non-syntactic hierarchical structure (Zhao et al., 2015; Chen et al., 2015b), convolutional neural networks (Kalchbrenner et al., 2014; Kim, 2014; Hu et al., 2014a; Yin and Schütze, 2015; He et al., 2015), and recurrent neural networks using long short-term memory (Tai et al., 2015; Ling et al., 2015a; Liu et al., 2015). Most of this work learns compositional models in the context of *supervised* learning. That is, a training set is provided with annotations and the composition function is learned discriminatively for the purposes of optimizing an objective function based on those annotations. The models are then evaluated on a test set drawn from the same distribution as the training set.

In contrast to supervised learning, others have proposed methods for creating general purpose, domain independent embeddings for word sequences. Early sentence embedding methods include paragraph vectors (Le and Mikolov, 2014) and skip-thought vectors (Kiros et al., 2015). Both of these approaches learn sequence representations that are predictive of words inside the sequence or in neighboring sequences. Skip-thought vectors capture similarity in terms of discourse context by using a encoder-decoder architecture. Paragraph vectors ... These methods produce generic representations that can be used to provide features for text classification or sentence similarity tasks. One popular early approach was to train an autoencoder in an attempt to learn the latent structure of the sequence, whether it be a sentence with a parse tree (Socher et al., 2011),a longer sequence such as a paragraph or document (Li et al., 2015), or X (Hill et al., 2016). Another popular approach is to use parallel text, and resources built from parallel text like NMT systems and PPDB, can be used for learning embeddings for words and sentences. Several have used PPDB as a knowledge resource for training or improving embeddings (Wieting et al., 2015; Yu and Dredze, 2015). (Wieting and Gimpel, 2017) later showed that PPDB is not as effective as sentential paraphrases, especially for recurrent networks. These results are intuitive because the phrases in PPDB are short and often cut across constituent boundaries. (Wieting and Gimpel, 2017) used a dataset developed for text simplification by (Coster and Kauchak, 2011). It was created by aligning sentences from Simple English and standard English Wikipedia. NMT architectures and training settings have been used to obtain better embeddings for words (Hill et al., 2014a,b) and words-in-context (McCann et al., 2017). (Hill et al., 2016) evaluated the encoders of English-to-X NMT systems as sentence representations. (Mallinson et al., 2017) adapted trained NMT models to produce sentence similarity scores in semantic evaluations. (Sutskever et al., 2014) trained NMT systems and visualized part of the space of the source language encoder for their English→French system. (Hill et al., 2016) evaluated the encoders of English-to-X NMT systems as sentence representations, finding them to perform poorly compared to several other methods based on unlabeled data. Mallinson et al. (2017) adapted trained NMT models to produce sentence similarity scores in semantic evaluations. They used pairs of NMT systems, one to translate an English sentence into multiple foreign translations and the other to then translate back to English.

This has been a popular area with many proposed architectures including LSTMs (Hill et al., 2016; Conneau et al., 2017; Schwenk and Douze, 2017; Subramanian et al., 2018), Transformers (Cer et al., 2018; Reimers and Gurevych, 2019), and averaging models (Wieting et al., 2016b; Arora et al., 2017; Pagliardini et al., 2017) have found success for learning sentence embeddings. The choice of training data and objective are intimately intertwined, and there are a wide variety of options including next-sentence prediction (Kiros et al., 2015), machine translation (Espana-Bonet et al., 2017; Schwenk and Douze, 2017; Schwenk, 2018; Artetxe and Schwenk, 2018b), natural language inference (NLI) (Conneau et al., 2017), and multi-task objectives which include some of the previously mentioned objectives (Cer et al., 2018) as well as additional tasks like constituency parsing (Subramanian et al., 2018).

2.1.4 *Contextualized Embeddings*

Contextualised embeddings (Dai and Le, 2015; Peters et al., 2018; Devlin et al., 2018) have had a dramatic effect on the performance on NLP tasks. In ELMo (Peters et al., 2018) (embeddings from language model), the contextualized embeddings are the hidden states of a deep bidirectional LSTM, trained as a bidirectional language model. This work has been extended to Transformers with BERT (Devlin et al., 2018). Transformers, compared to RNNs, allow for using much deeper contexts since they are able to handle longer contexts and are not limited to being unidirectional and can attend to context both before and after a word. They also do not need to process the input sequentially, and therefore can better make use of the massive parallel computation allowed for by GPUs. The objective used to train BERT is a masked language model, in contrast to the unidirectional language model used to train ELMo. In a masked language model, the objective is to predict the identity of words given bidirectional context. Since BERT many extensions and adaptions have been proposed (Liu et al., 2019; Lample and Conneau, 2019; Yang et al., 2019b; Lan et al., 2019; Sanh et al., 2019).

2.2 EVALUATING REPRESENTATIONS

2.2.1 *Word Embedding Evaluation*

There are two main approaches to evaluating word embeddings. The first, intrinsic, involves comparing the distance between word embedding pairs on a list and then measuring how close the distances are to some human-rankings. Extrinsic evaluations involve using the word embeddings as part of a larger model and comparing performance on some downstream task.

The early and standard approaches to evaluating word embeddings relied on intrinsic evaluation on lists of word pairs. One of the datasets constructed for this purpose was RG-65 (Rubenstein and Goodenough, 1965), C-30 (Miller and Charles, 1991) which consist of 65 and 30 word pairs respectively.

Wordsim-353 (WS353) (Finkelstein et al., 2001), specifically its similarity (WS-S) and relatedness (WS-R) partitions (Agirre et al., 2009). SimLex-999 (SL999) dataset (Hill et al., 2015) most closely evaluates the paraphrase relationship. Even though WS-S is a close approximation to this relationship, it does not include pairs that are merely associated and assigned low scores, which SL999 rectifies. However, SL999 has been its focus on nouns and high frequency concepts. Other popular datasets include (Bruni et al., 2014; Gerz et al., 2016; Pilehvar et al., 2018; Finkelstein et al., 2001; Hill et al., 2014b; Luong et al., 2013). Mikolov et al. (2013a) proposed using analogies for evaluation.

2.2.2 *Sentence Embedding Evaluation*

There have been two main approaches to evaluating the quality of sentence embeddings: classification tasks and semantic similarity tasks. One of the best approaches to analyzing sentence embeddings is through probing tasks (Ettinger et al., 2016; Conneau et al., 2018a) where embeddings are trained on some data to see how well they classify various linguistic phenomena (for example, the tense of a sentence or how many tokens it contains).

2.2.2.1 *Semantic Textual Similarity (STS)*

Semantic similarity tasks were originally meant to test the performance of supervised semantic similarity systems, training data was provided and teams were advised to make use of resources like WordNet (Miller, 1995), Wikipedia, paraphrase tables, dictionaries, as well as NLP tools for tagging, named entity recognition, and semantic role labeling.

We adopted these datasets to measure *unsupervised* semantic similarity. That is, we did not use the training data. We were the first to propose this use of the datasets and it has since become a standard tool for evaluating sentence embeddings. We selected these datasets because they reflect semantic similarity (as opposed to relatedness like *couch* and *table*) and cover many different domains to discourage overfitting on a specific type of data. Other popular paraphrase datasets include (Marelli et al., 2014; Dolan and Brockett, 2005; Zhang et al., 2019b)

ENGLISH STS The first STS task was held in 2012 and these tasks have been held every year since. Given two sentences, the objective of the task is to predict how similar they are on a 0-5 scale, where 0 indicates the sentences are on different topics and 5 indicates that they are completely equivalent. Each STS task consists of 4-6 different datasets and the tasks cover a wide variety of domains which we have categorized below. Most submissions for these tasks use supervised models that are trained and tuned on either provided training data or similar datasets from older tasks. Details on the number of teams and submissions for each task and the performance of the submitted systems for each dataset are included in Table 1 and Table 14 respectively. For more details on these tasks please refer to the relevant publications for the 2012 (Agirre et al., 2012), 2013 (Agirre et al., 2013), 2014 (Agirre et al., 2014), and 2015 (Agirre et al., 2015) tasks.

Dataset	No. of teams	No. of submissions
2012 STS	35	88
2013 STS	34	89
2014 STS	15	38
2015 STS	29	74
2016 STS	43	119
2014 SICK	17	66
2015 Twitter	19	26

Table 1: Details on numbers of teams and submissions in the STS tasks used for evaluation.

Below are the textual domains contained in the STS tasks:

News: Newswire was used in the 2012 task (MSRpar) and the 2013 and 2014 tasks (deft news).

Image and Video Descriptions: Image descriptions generated via crowdsourcing were used in the 2013 and 2014 tasks (images). Video descriptions were used in the 2012 task (MSRvid).

Glosses: Glosses from WordNet, OntoNotes, and FrameNet were used in the 2012, 2013, and 2014 tasks (OnWN and FNWN).

MT evaluation: The output of machine translation systems with their reference translations was used in the 2012 task (SMT-eur and SMT-news) and the 2013 task (SMT).

Headlines: Headlines of news articles were used in the 2013, 2014, 2015, and 2016 tasks (headline).

Web Forum: Forum posts were used in the 2014 task (deft forum).

Twitter: Pairs containing a tweet related to a news headline and a sentence pertaining to the same news headline. This dataset was used in the 2014 task (tweet news).

Belief: Text from the Deft Committed Belief Annotation (LDC2014E55) was used in the 2015 task (belief).

Questions and Answers: Paired answers to the same question from Stack Exchange (answers-forums) and the BEETLE corpus (Dzikovska et al., 2010) (answers-students) were used in 2015. A similar task **Answer-Answer** was used in the 2016 evaluation also based on Stack Exchange. **Question-Question**: Similar to Answer-Answer, paired questions (the title of the post when it ends in a question mark in this case) were used in the 2016 task. The questions were also taken from Stack Exchange. **Plagarism**: The plagiarism dataset is based on the Corpus of Plagiarised Short Answers (Clough and Stevenson, 2011). This corpus contains short answers to computer science questions that exhibit plagiarism from related Wikipedia articles. **Post-Editing**: (Specia, 2011)

NON-ENGLISH AND CROSS-LINGUAL STS Cross-lingual STS started as a pilot task in 2014 and repeated again in 2015 using Spanish-Spanish pairs. The 2016 task included a pilot track on cross-lingual Spanish-English pairs. In 2017, semantic similarity between non-English sentences and cross-lingual semantic similarity became the primary focus. To create the pairs used in evaluation, sentence pairs were sampled from the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015) and translated into the appropriate language. Scores for the sentence pairs were obtained by human evaluation via Amazon Mechanical Turk.[2]. This evaluation contains Arabic-Arabic, Arabic-English, Spanish-Spanish,

2 https://www.mturk.com

Spanish-English, and Turkish-English STS datasets. These datasets were created by translating one or both pairs of an English STS pair into Arabic (`ar`), Spanish (`es`), or Turkish (`tr`).

In the 2017 competition, 31 teams submitted 84 submissions with 17 teams provided 44 systems participating all in tracks (datasets). The best system (Tian et al., 2017) ensembled feature engineered models with deep learning approaches. They also used Google Translate to translate them to English to simplify the problem. Note that they also used the Paragram embeddings we created outlined in Chapter 4.

(Yang et al., 2019a) (Zweigenbaum et al., 2018) (Artetxe and Schwenk, 2018b)

2.2.2.2 *Sentence Classification*

In addition to the semantic similarity evaluations, there is an alternative approach measuring the effectiveness of sentence embeddings. This approach to evaluation was pioneered in (Kiros et al., 2015). In this setting, sentence representations are used as features for a linear classifier. The sentence embeddings are kept fixed, while training data is used to fine-tune the classifier on the target task. Tasks used in these evaluations have differed and have been even extended to the multilingual domain (Conneau et al., 2018b), but `SentEval` (Conneau and Kiela, 2018) has become a standard for English sentence embeddings.[3] The tasks included are sentiment analysis (MR, Pang and Lee, 2005; CR, Hu and Liu, 2004; SST, Socher et al., 2013), subjectivity classification (SUBJ; Pang and Lee, 2004), opinion polarity (MPQA; Wiebe et al., 2005), question classification (TREC; Li and Roth, 2002), paraphrase detection (MRPC; Dolan et al., 2004), semantic relatedness (SICK-R; Marelli et al., 2014), and textual entailment (SICK-E).

While this method of evaluation has been widespread, it does have a number of drawbacks when compared using semantic similarity tasks as an evaluation. The main one is the lack of diversity in tasks, as most are sentiment. This evaluation can also be more even gamed than evaluating on semantic similarity as training on in-domain data[4] significantly improves results. Furthermore it is also possible that training sentence embeddings to be used as features (as done in Conneau et al. (2017) and Cer et al. (2018)) can also give a boost to performance since it more closely aligns with the evaluation task.

3 `github.com/facebookresearch/SentEval`

4 For example, product reviews for sentiment, Quora data for TREC, Stanford Natural Language Inference (SNLI) (Bowman et al., 2015) for paraphrase detection, semantic relatedness, and semantic similarity)

2.3 PARAPHRASE CORPORA

2.3.1 *Paraphrase Database*

The Paraphrase Database (PPDB; Ganitkevitch et al., 2013), is a collection of confidence-rated paraphrases created using the bilingual pivoting technique of (Bannard and Callison-Burch, 2005) over large parallel corpora. Bilingual pivoting is a method from statistical machine translation to find lexical and phrasal paraphrases in parallel text, where PPDB is a massive resource, containing 220 million paraphrase pairs. It captures many short paraphrases that would be difficult to obtain using any other resource. For example, the pair {*we must do our utmost*, *we must make every effort*} has little lexical overlap but is present in PPDB. The PPDB has recently been used for monolingual alignment (Yao et al., 2013), for predicting sentence similarity (Bjerva et al., 2014), and to improve the coverage of FrameNet (Rastogi and Van Durme, 2014).

Since PPDB only relies on the availability of bilingual text, it can be extracted for many languages (Ganitkevitch and Callison-Burch, 2014). Other extensions to the original PPDB have also been developed. PPDB 2.0 was released in Pavlick et al. (2015), where the paraphrases were re-ranked to be more aligned with human judgements. The original PPDB used a heuristic scoring model based on features for the paraphrases, bu in this paper the authors trained a supervised model using human judgments from Mechanical Turk.[5] They also automatically annotated the paraphrases with entailment relations, word embedding similarities, and style annotations. In (Pavlick and Callison-Burch, 2016), they release Simple PPDB, which consists of 4.5 million simplifying paraphrase score. To create this resource, Turkers are asked to annotate PPDB pairs first on two tasks. The first is whether they or not the pairs show a paraphrase. Then for all paraphrases, they ask the Turkers to rate which of the paraphrases is simpler or if there is no complexity difference. From these annotations they are then able to train a supervised model and annotate a filtered subset of PPDB with a simplification confidence score.

Though effective for multiple NLP tasks, we note some drawbacks of PPDB. The first is lack of coverage: to use the PPDB to compare two phrases, both must be in the database. The second is that PPDB is a nonparametric paraphrase model; the number of parameters (phrase pairs) grows with the size of the dataset used to build it. In practice, it can become unwieldy to work with as the size of the database increases. A third concern is that the

5 https://www.mturk.com/

confidence estimates in PPDB are a heuristic combination of features, and their quality is unclear.

2.3.2 *Sentential Paraphrase Corpora*

Many methods have been developed for generating or finding paraphrases, including using multiple translations of the same source material (Barzilay and McKeown, 2001), using distributional similarity to find similar dependency paths (Lin and Pantel, 2001), using comparable articles from multiple news sources (Dolan et al., 2004; Dolan and Brockett, 2005; Quirk et al., 2004), aligning sentences between standard and Simple English Wikipedia (Coster and Kauchak, 2011), crowdsourcing (Xu et al., 2014, 2015b; Jiang et al., 2017), using diverse MT systems to translate a single source sentence (Suzuki et al., 2017), using tweets with matching URLs (Lan et al., 2017a).

There is also a rich history of research in generating or finding naturally-occurring sentential paraphrases (Barzilay and McKeown, 2001; Dolan et al., 2004; Dolan and Brockett, 2005; Quirk et al., 2004; Coster and Kauchak, 2011; Xu et al., 2014, 2015b).

Part I

PARAPHRASTIC EMBEDDINGS

In this section, we discuss our first models for learning *paraphrastic* representations. This section encompasses work that has appeared in three papers: (Wieting et al., 2015, 2016b,a).

In Chapter 3, we find that state-of-the-art phrase embeddings could be learned by using text snippets from the Paraphrase Database (PPDB) (Ganitkevitch et al., 2013) as training data. PPDB is created by bilingual pivoting (Bannard and Callison-Burch, 2005) on parallel bitext, where two text snippets are paraphrases when they align to the same snippet in another language. While the recursive RNN (Socher et al., 2011) architecture in this paper was overly complicated (which is shown in later papers), we laid the groundwork with a focus on paraphrastic similarity, our training strategy, and our proposed objective function. We use the term paraphrastic similarity to distinguish it from other types of similarity which were largely inter-mixed at the time - specifically separating the notion of *relatedness* from paraphrastic where relatedness refers to words that appear in similar contexts (*table* and *chair*) while paraphrastic refers to words that could be paraphrases in some context (*chair* and *seat*). One other important contribution from that paper were PARAGRAM word embeddings. These significantly surpassed the state-of-the-art word embedding approaches for measuring similarity and are even the basis for state-of-the-art models today.

In Chapter 4. we extended (Wieting et al., 2015) to learn sentence embeddings. Our primary evaluation was on a suite of semantic textual similarity (STS) datasets but we also follow the setting of (Kiros et al., 2015) and show that we can rival their performance with our simpler model. This is also the first paper to use these suite of tasks as an evaluation approach for sentence embeddings, which has become a standard. We experimented with a wide variety of encoding architectures, and we fount that the simplest approach, averaging word embeddings, to be the most effective and are able to surpass the vast majority of proposed STS systems despite being unsupervised. This was a surprising result, considering LSTM (Hochreiter and Schmidhuber, 1997) were setting new state-of-the-arts for many problems in natural language processing. In this paper, we analyzed some hypotheses to explain these results, finding that the reason wasn't due to length, over-fitting, or insufficient parameter tuning. LSTMs (and deeper architectures in general) can work well on this problem, but the training strategies need to change. This is further discussed in Chapter 6 and Chapter 10.

Lastly, in Chapter 5, we discuss our CHARAGRAM models for learning word and sentence representations. In these representations, we represent a character sequence by a vector containing counts of character n-grams, inspired by Huang et al. (2013). This vector is embedded into a low-dimensional space using a single nonlinear transformation. This can be interpreted as learning embeddings of character n-grams, which are learned so as to produce effective sequence embeddings when a summation is performed over the character n-grams in the sequence. We evaluate on three tasks: word similarity, sentence similarity, and part-of-speech tagging and show that CHARAGRAM outperforms character RNNs, character CNNs, as well as PARAGRAM embeddings. We find that modelling subwords yields large gains in performance for rare words and can easily handle spelling variation, morphology, and word choice.

3 RECURSIVE NEURAL NETWORKS AND PARAGRAM WORD EMBEDDINGS

3.1 INTRODUCTION

Paraphrase detection[1] is the task of analyzing two segments of text and determining if they have the same meaning despite differences in structure and wording. It is useful for a variety of NLP tasks like question answering (Rinaldi et al., 2003; Fader et al., 2013), semantic parsing (Berant and Liang, 2014), textual entailment (Bosma and Callison-Burch, 2007), and machine translation (Marton et al., 2009).

One component of many such systems is a paraphrase table containing pairs of text snippets, usually automatically generated, that have the same meaning. The most recent work in this area is the Paraphrase Database (PPDB; Ganitkevitch et al., 2013), a collection of confidence-rated paraphrases created using the pivoting technique of Bannard and Callison-Burch (2005) over large parallel corpora. The PPDB is a massive resource, containing 220 million paraphrase pairs. It captures many short paraphrases that would be difficult to obtain using any other resource. For example, the pair {*we must do our utmost*, *we must make every effort*} has little lexical overlap but is present in PPDB. The PPDB has recently been used for monolingual alignment (Yao et al., 2013), for predicting sentence similarity (Bjerva et al., 2014), and to improve the coverage of FrameNet (Rastogi and Van Durme, 2014).

Though already effective for multiple NLP tasks, we note some drawbacks of PPDB. The first is lack of coverage: to use the PPDB to compare two phrases, both must be in the database. The second is that PPDB is a nonparametric paraphrase model; the number of parameters (phrase pairs) grows with the size of the dataset used to build it. In practice, it can become unwieldy to work with as the size of the database increases. A third concern is that the confidence estimates in PPDB are a heuristic combination of features, and their quality is unclear.

1 See Androutsopoulos and Malakasiotis (2010) for a survey on approaches for detecting paraphrases.

We address these issues in this work by introducing ways to use PPDB to construct parametric paraphrase models. First we show that initial skip-gram word vectors (Mikolov et al., 2013a) can be fine-tuned for the paraphrase task by training on word pairs from PPDB. We call them **paragram** word vectors. We find additive composition of PARAGRAM vectors to be a simple but effective way to embed phrases for short-phrase paraphrase tasks. We find improved performance by training a recursive neural network (RNN; Socher et al., 2010) directly on phrase pairs from PPDB.

We show that our resulting word and phrase representations are effective on a wide variety of tasks, including two new datasets that we introduce. The first, Annotated-PPDB, contains pairs from PPDB that were scored by human annotators. It can be used to evaluate paraphrase models for short phrases. We use it to show that the phrase embeddings produced by our methods are significantly more indicative of paraphrasability than the original heuristic scoring used by Ganitkevitch et al. (2013). Thus we use the power of PPDB to improve its contents.

Our second dataset, ML-Paraphrase, is a re-annotation of the bigram similarity corpus from Mitchell and Lapata (2010). The task was originally developed to measure semantic similarity of bigrams, but some annotations are not congruent with the functional similarity central to paraphrase relationships. Our re-annotation can be used to assess paraphrasing capability of bigram compositional models.

In summary, we make the following contributions:

Provide new paragram word vectors, learned using PPDB, that achieve state-of-the-art performance on the SimLex-999 lexical similarity task (Hill et al., 2015) and lead to improved performance in sentiment analysis.

Provide ways to use PPDB to embed phrases. We compare additive and RNN composition of PARAGRAM vectors. Both can improve PPDB by re-ranking the paraphrases in PPDB to improve correlations with human judgments. They can be used as concise parameterizations of PPDB, thereby vastly increasing its coverage. We also perform a qualitative analysis of the differences between additive and RNN composition.

Introduce two new datasets. The first contains PPDB phrase pairs and evaluates how well models can measure the quality of short paraphrases. The second is a new annotation of the bigram similarity task in Mitchell and Lapata (2010) that makes it suitable for evaluating bigram paraphrases.

We release the new datasets, complete with annotation instructions and raw annotations, as well as our code and the trained models.[2]

3.2 RELATED WORK

There is a vast literature on representing words as vectors. The intuition of most methods to create these vectors (or embeddings) is that similar words have similar contexts (Firth, 1957). Earlier models made use of latent semantic analysis (LSA) (Deerwester et al., 1990). Recently, more sophisticated neural models, work originating with (Bengio et al., 2003), have been gaining popularity (Mikolov et al., 2013a; Pennington et al., 2014). These embeddings are now being used in new ways as they are being tailored to specific downstream tasks (Bansal et al., 2014).

Phrase representations can be created from word vectors using compositional models. Simple but effective compositional models were studied by Mitchell and Lapata (2008; 2010) and Blacoe and Lapata (2012). They compared a variety of binary operations on word vectors and found that simple point-wise multiplication of explicit vector representations performed very well. Other works like Zanzotto et al. (2010) and Baroni and Zamparelli (2010) also explored composition using models based on operations of vectors and matrices.

More recent work has shown that the extremely efficient neural embeddings of Mikolov et al. (2013a) also do well on compositional tasks simply by adding the word vectors (Mikolov et al., 2013b). Hashimoto et al. (2014) introduced an alternative word embedding and compositional model based on predicate-argument structures that does well on two simple composition tasks, including the one introduced by Mitchell and Lapata (2010).

An alternative approach to composition, used by Socher et al. (2011), is to train a recursive neural network (RNN) whose structure is defined by a binarized parse tree. In particular, they trained their RNN as an unsupervised autoencoder. The RNN captures the latent structure of composition. Recent work has shown that this model struggles in tasks involving compositionality (Blacoe and Lapata, 2012; Hashimoto et al., 2014).[3] However, we found success using RNNs in a *supervised* setting, similar to Socher et al. (2014), who used RNNs to learn representations for image descriptions. The objective function we used in this work was motivated by their multimodal objective function for learning joint image-sentence representations.

2 available on the authors' websites

3 We also replicated this approach and found training to be time-consuming even using low-dimensional word vectors.

Lastly, the PPDB has been used along with other resources to learn word embeddings for several tasks, including semantic similarity, language modeling, predicting human judgments, and classification (Yu and Dredze, 2014; Faruqui et al., 2015). Concurrently with our work, it has also been used to construct paraphrase models for short phrases (Yu and Dredze, 2015).

3.3 NEW PARAPHRASE DATASETS

We created two novel datasets: (1) Annotated-PPDB, a subset of phrase pairs from PPDB which are annotated according to how strongly they represent a paraphrase relationship, and (2) ML-Paraphrase, a re-annotation of the bigram similarity dataset from Mitchell and Lapata (2010), again annotated for strength of paraphrase relationship.

3.3.1 *Annotated-PPDB*

Our motivation for creating Annotated-PPDB was to establish a way to evaluate compositional paraphrase models on *short phrases*. Most existing paraphrase tasks focus on words, like SimLex-999 (Hill et al., 2015), or entire sentences, such as the Microsoft Research Paraphrase Corpus (Dolan et al., 2004; Quirk et al., 2004). To our knowledge, there are no datasets that focus on the paraphrasability of short phrases. Thus, we created Annotated-PPDB so that researchers can focus on local compositional phenomena and measure the performance of models directly—avoiding the need to do so indirectly in a sentence-level task. Models that have strong performance on Annotated-PPDB can be used to provide more accurate confidence scores for the paraphrases in the PPDB as well as reduce the need for large paraphrase tables altogether.

Annotated-PPDB was created in a multi-step process (outlined below) involving various automatic filtering steps followed by crowdsourced human annotation. One of the aims for our dataset was to collect a variety of paraphrase types—we wanted to include pairs that were non-trivial to recognize as well as those with a range of similarity and length. We focused on phrase pairs with limited lexical overlap to avoid including those with only trivial differences.

We started with candidate phrases extracted from the first 10M pairs in the XXL version of the PPDB and then executed the following steps.[4]

Filter phrases for quality: Only those phrases whose tokens were in our vocabulary were retained.[5] Next, all duplicate paraphrase pairs were removed; in PPDB, these are distinct pairs that contain the same two phrases with the order swapped.

Filter by lexical overlap: Next, we calculated the *word overlap score* in each phrase pair and then retained only those pairs that had a score of less than 0.5. By *word overlap score*, we mean the fraction of tokens in the smaller of the phrases with Levenshtein distance $\leqslant 1$ to a token in the larger of the phrases. This was done to exclude less interesting phrase pairs like ⟨*my dad had*, *my father had*⟩ or ⟨*ballistic missiles*, *of ballistic missiles*⟩ that only differ in a synonym or the addition of a single word.

Select range of paraphrasabilities: To balance our dataset with both clear paraphrases and erroneous pairs in PPDB, we sampled 5,000 examples from ten chunks of the first 10M initial phrase pairs where a chunk is defined as 1M phrase pairs.

Select range of phrase lengths: We then selected 1,500 phrases from each 5000-example sample that encompassed a wide range of phrase lengths. To do this, we first binned the phrase pairs by their *effective size*. Let n_1 be the number of tokens of length greater than one character in the first phrase and n_2 the same for the second phrase. Then the *effective size* is defined as $\max(n_1, n_2)$. The bins contained pairs of *effective size* of 3, 4, and 5 or more, and 500 pairs were selected from each bin. This gave us a total of 15,000 phrase pairs.

Prune to 3,000: 3,000 phrase pairs were then selected randomly from the 15,000 remaining pairs to form an initial dataset, Annotated-PPDB-3K. The phrases were selected so that every phrase in the dataset was unique.

Annotate with Mechanical Turk: The dataset was then rated on a scale from 1-5 using Amazon Mechanical Turk, where a score of 5 denoted phrases that are equivalent in a large number of contexts, 3 meant that the phrases had some overlap in meaning, and 1 indicated that the phrases were dissimilar or contradictory in some way (e.g., *can not adopt* and *is able to accept*).

We only permitted workers whose location was in the United States and who had done at least 1,000 HITS with a 99% acceptance rate. Each example was labeled by 5 annotators and their scores were averaged to produce the final rating. Table 2 shows some statistics of

4 Note that the confidence scores for phrase pairs in PPDB are based on a weighted combination of features with weights determined heuristically. The confidence scores were used to place the phrase pairs into their respective sets (S, M, L, XL, XXL, etc.), where each larger set subsumes all smaller ones.

5 Throughout, our vocabulary is defined as the most common 100K word types in English Wikipedia, following tokenization and lowercasing (see section 3.5).

Score Range	MD	% of Data
$[1,2)$	0.66	8.1
$[2,3)$	1.05	20.0
$[3,4)$	0.93	34.9
$[4,5]$	0.59	36.9

Table 2: An analysis of Annotated-PPDB-3K extracted from PPDB. The statistics shown are for the splits of the data according to the average score by workers. MD denotes mean deviation and % of Data refers to the percentage of our dataset that fell into each range.

the data. Overall, the annotated data had a mean deviation (MD)[6] of 0.80. Table 2 shows that overall, workers found the phrases to be of high quality, as more than two-thirds of the pairs had an average score of at least 3. Also from the Table, we can see that workers had stronger agreement on very low and very high quality pairs and were less certain in the middle of the range.

Prune to 1,260: To create our final dataset, Annotated-PPDB, we selected 1,260 phrase pairs from the 3,000 annotations. We did this by first binning the phrases into 3 categories: those with scores in the interval $[1, 2.5)$, those with scores in the interval $[2.5, 3.5]$, and those with scores in the interval $(3.5, 5]$. We took the 420 phrase pairs with the lowest MD in each bin, as these have the most agreement about their label, to form Annotated-PPDB.

These 1,260 examples were then randomly split into a development set of 260 examples and a test set of 1,000 examples. The development set had an MD of 0.61 and the test set had an MD of 0.60, indicating the final dataset had pairs of higher agreement than the initial 3,000.

3.3.2 *ML-Paraphrase*

Our second newly-annotated dataset, ML-Paraphrase, is based on the bigram similarity task originally introduced by Mitchell and Lapata (2010); we refer to the original annotations as the ML dataset.

The ML dataset consists of human similarity ratings for three types of bigrams: adjective-noun (JN), noun-noun (NN), and verb-noun (VN). Through manual inspection, we found that the annotations were not consistent with the notion of similarity central to paraphrase tasks. For instance, *television set* and *television programme* were the highest rated phrases in the NN section (based on average annotator score). Similarly, one of the highest ranked JN

6 MD is similar to standard deviation, but uses absolute value instead of squared value and thus is both more intuitive and less sensitive to outliers.

pairs was *older man* and *elderly woman*. This indicates that the annotations reflect topical similarity in addition to capturing functional or definitional similarity.

Therefore, we had the data re-annotated by two authors of this paper who are native English speakers.[7] The bigrams were labeled on a scale from 1-5 where 5 denotes phrases that are equivalent in a large number of contexts, 3 indicates the phrases are roughly equivalent in a narrow set of contexts, and 1 means the phrases are not at all equivalent in any context. Following annotation, we collapsed the rating scale by merging 4s and 5s together and 1s and 2s together.

Data	IA ρ	IA κ	ML comp. ρ	ML Human ρ
JN	0.87	0.79	0.56	0.52
NN	0.64	0.58	0.38	0.49
VN	0.73	0.73	0.55	0.55

Table 3: Inter-annotator agreement of ML-Paraphrase and comparison with ML dataset. Columns 2 and 3 show the inter-annotator agreement between the two annotators measured with Spearman ρ and Cohen's κ. Column 4 shows the ρ between ML-Paraphrase and all of the ML dataset. The last column is the average human ρ on the ML dataset.

Statistics for the data are shown in Table 3. We show inter-annotator Spearman ρ and Cohen's κ in columns 2 and 3, indicating substantial agreement on the JN and VN portions but only moderate agreement on NN. In fact, when evaluating our NN annotations against those from the original ML data (column 4), we find ρ to be 0.38, well below the average human correlation of 0.49 (final column) reported by Mitchell and Lapata and also surpassed by pointwise multiplication (Mitchell and Lapata, 2010). This suggests that the original NN portion, more so than the others, favored a notion of similarity more related to association than paraphrase.

3.4 PARAPHRASE MODELS

We now present parametric paraphrase models and discuss training. Our goal is to embed phrases into a low-dimensional space such that cosine similarity in the space corresponds to the strength of the paraphrase relationship between phrases.

We use a recursive neural network (RNN) similar to that used by Socher et al. (2014). We first use a constituent parser to obtain a binarized parse of a phrase. For phrase p, we compute its vector $g(p)$ through recursive computation on the parse. That is, if phrase p is

7 We tried using Mechanical Turk here, but due to such short phrases, with few having the paraphrase relationship, workers did not perform well on the task.

the yield of a parent node in a parse tree, and phrases c_1 and c_2 are the yields of its two child nodes, we define $g(p)$ recursively as follows:

$$g(p) = f(W[g(c_1); g(c_2)] + b)$$

where f is an element-wise activation function (tanh), $[g(c_1); g(c_2)] \in \mathbb{R}^{2n}$ is the concatenation of the child vectors, $W \in \mathbb{R}^{n \times 2n}$ is the composition matrix, $b \in \mathbb{R}^n$ is the offset, and n is the dimensionality of the word embeddings. If node p has no children (i.e., it is a single token), we define $g(p) = W_w^{(p)}$, where W_w is the word embedding matrix in which particular word vectors are indexed using superscripts. The trainable parameters of the model are W, b, and W_w.

3.4.1 *Objective Functions*

We now present objective functions for training on pairs extracted from PPDB. The training data consists of (possibly noisy) pairs taken directly from the original PPDB. In subsequent sections, we discuss how we extract training pairs for particular tasks.

We assume our training data consists of a set X of phrase pairs $\langle x_1, x_2 \rangle$, where x_1 and x_2 are assumed to be paraphrases. To learn the model parameters (W, b, W_w), we minimize our objective function over the data using AdaGrad (Duchi et al., 2011) with mini-batches. The objective function follows:

$$\begin{aligned}
\min_{W,b,W_w} \frac{1}{|X|} \Bigg(\sum_{\langle x_1, x_2 \rangle \in X} \\
& \max(0, \delta - g(x_1) \cdot g(x_2) + g(x_1) \cdot g(t_1)) \\
& + \max(0, \delta - g(x_1) \cdot g(x_2) + g(x_2) \cdot g(t_2)) \Bigg) \\
& \qquad + \lambda_W (\|W\|^2 + \|b\|^2) + \lambda_{W_w} \left\| W_{w_{initial}} - W_w \right\|^2
\end{aligned}$$

where λ_W and λ_{W_w} are regularization parameters, $W_{w_{initial}}$ is the initial word embedding matrix, δ is the **margin** (set to 1 in all of our experiments), and t_1 and t_2 are carefully-selected **negative examples** taken from a mini-batch during optimization.

The intuition for this objective is that we want the two phrases to be more similar to each other $(g(x_1) \cdot g(x_2))$ than either is to their respective negative examples t_1 and t_2, by a margin of at least δ.

SELECTING NEGATIVE EXAMPLES To select t_1 and t_2 in Eq. 3.4.1, we simply chose the most similar phrase in the mini-batch (other than those in the given phrase pair). E.g., for choosing t_1 for a given $\langle x_1, x_2 \rangle$:

$$t_1 = \operatorname*{argmax}_{t:\langle t,\cdot\rangle \in X_b \setminus \{\langle x_1, x_2 \rangle\}} g(x_1) \cdot g(t)$$

where $X_b \subseteq X$ is the current mini-batch. That is, we want to choose a negative example t_i that is similar to x_i according to the current model parameters. The downside of this approach is that we may occasionally choose a phrase t_i that is actually a true paraphrase of x_i. We also tried a strategy in which we selected the least similar phrase that would trigger an update (i.e., $g(t_i) \cdot g(x_i) > g(x_1) \cdot g(x_2) - \delta$), but we found the simpler strategy above to work better and used it for all experiments reported below.

DISCUSSION The objective in Eq. 3.4.1 is similar to one used by Socher et al. (2014), but with several differences. Their objective compared text and projected images. They also did not update the underlying word embeddings; we do so here, and in a way such that they are penalized from deviating from their initialization. Also for a given $\langle x_1, x_2 \rangle$, they do not select a single t_1 and t_2 as we do, but use the entire training set, which can be very expensive with a large training dataset.

We also experimented with a simpler objective that sought to directly minimize the squared L2-norm between $g(x_1)$ and $g(x_2)$ in each pair, along with the same regularization terms as in Eq. 3.4.1. One problem with this objective function is that the global minimum is 0 and is achieved simply by driving the parameters to 0. We obtained much better results using the objective in Eq. 3.4.1.

TRAINING WORD PARAPHRASE MODELS To train just word vectors on word paraphrase pairs (again from PPDB), we used the same objective function as above, but simply dropped the composition terms. This gave us an objective that bears some similarity to the skip-gram objective with negative sampling in `word2vec` (Mikolov et al., 2013a). Both seek to maximize the dot products of certain word pairs while minimizing the dot products of others. This objective function is:

$$\min_{W_w} \frac{1}{|X|} \Bigg(\sum_{\langle x_1, x_2 \rangle \in X} \max(0, \delta - W_w^{(x_1)} \cdot W_w^{(x_2)} + W_w^{(x_1)} \cdot W_w^{(t_1)}) + \max(0, \delta - W_w^{(x_1)} \cdot W_w^{(x_2)} + W_w^{(x_2)} \cdot W_w^{(t_2)}) \Bigg) + \lambda_{W_w} \left\| W_{w_{initial}} - W_w \right\|^2$$

It is like Eq. 3.4.1 except with word vectors replacing the RNN composition function and with the regularization terms on the W and b removed.

We further found we could improve this model by incorporating constraints. From our training pairs, for a given word w, we assembled all other words that were paired with it in PPDB and all of their lemmas. These were then used as constraints during the pairing process: a word t could only be paired with w if it was not in its list of assembled words.

3.5 Experiments – Word Paraphrasing

We first present experiments on learning lexical paraphrasability. We train on word pairs from PPDB and evaluate on the SimLex-999 dataset (Hill et al., 2015), achieving the best results reported to date.

3.5.1 *Training Procedure*

To learn word vectors that reflect paraphrasability, we optimized Eq. 3.4.1. There are many tunable hyperparameters with this objective, so to make training tractable we fixed the initial learning rates for the word embeddings to 0.5 and the margin δ to 1. Then we did a coarse grid search over a parameter space for λ_{W_w} and the mini-batch size. We considered λ_{W_w} values in $\{10^{-2}, 10^{-3}, ..., 10^{-7}, 0\}$ and mini-batch sizes in {100, 250, 500, 1000}. We trained for 20 epochs for each set of hyperparameters using AdaGrad (Duchi et al., 2011).

For all experiments, we initialized our word vectors with skip-gram vectors trained using `word2vec` (Mikolov et al., 2013a). The vectors were trained on English Wikipedia (tokenized and lowercased, yielding 1.8B tokens).[8] We used a window size of 5 and a minimum count cut-off of 60, producing vectors for approximately 270K word types. We retained vectors for

8 We used the December 2, 2013 snapshot.

only the 100K most frequent words, averaging the rest to obtain a single vector for unknown words. We will refer to this set of the 100K most frequent words as our **vocabulary**.

3.5.2 *Extracting Training Data*

For training, we extracted word pairs from the lexical XL section of PPDB. We used the XL data for all experiments, including those for phrases. We used XL instead of XXL because XL has better quality overall while still being large enough so that we could be selective in choosing training pairs. There are a total of 548,085 pairs. We removed 174,766 that either contained numerical digits or words not in our vocabulary. We then removed 260,425 redundant pairs, leaving us with a final training set of 112,894 word pairs.

3.5.3 *Tuning and Evaluation*

Hyperparameters were tuned using the wordsim-353 (WS353) dataset (Finkelstein et al., 2001), specifically its similarity (WS-S) and relatedness (WS-R) partitions (Agirre et al., 2009). In particular, we tuned to maximize 2×WS-S correlation minus the WS-R correlation. The idea was to reward vectors with high similarity and relatively low relatedness, in order to target the paraphrase relationship.

After tuning, we evaluated the best hyperparameters on the SimLex-999 (SL999) dataset (Hill et al., 2015). We chose SL999 as our primary test set as it most closely evaluates the paraphrase relationship. Even though WS-S is a close approximation to this relationship, it does not include pairs that are merely associated and assigned low scores, which SL999 does (see discussion in Hill et al., 2014b).

Note that for all experiments we used cosine similarity as our similarity metric and evaluated the statistical significance of dependent correlations using the one-tailed method of (Steiger, 1980).

3.5.4 *Results*

Table 4 shows results on SL999 when improving the initial word vectors by training on word pairs from PPDB, both with and without constraints. The "PARAGRAM $_{\text{WS}}$" rows show results when tuning to maximize 2×WS-S − WS-R. We also show results for strong skip-

Model	n	SL999 ρ
skip-gram	25	0.21
skip-gram	1000	0.38
PARAGRAM WS	25	0.56*
+ constraints	25	**0.58***
Hill et al. (2015)	200	0.446
Hill et al. (2014b)	-	0.52
inter-annotator agreement	N/A	0.67

Table 4: Results on the SimLex-999 (SL999) word similarity task obtained by performing hyperparameter tuning based on 2×WS-S −WS-R and treating SL999 as a held-out test set. n is word vector dimensionality. A $*$ indicates statistical significance ($p < 0.05$) over the 1000-dimensional skip-gram vectors.

gram baselines and the best results from the literature, including the state-of-the-art results from Hill et al. (2014b) as well as the inter-annotator agreement from Hill et al. (2015).[9]

The table illustrates that, by training on PPDB, we can surpass the previous best correlations on SL999 by 4-6% absolute, achieving the best results reported to date. We also find that we can train low-dimensional word vectors that exceed the performance of much larger vectors. This is very useful as using large vectors can increase both time and memory consumption in NLP applications.

To generate word vectors to use for downstream applications, we chose hyperparameters so as to maximize performance on SL999.[10] These word vectors, which we refer to as **paragram** vectors, had a ρ of 0.57 on SL999. We use them as initial word vectors for the remainder of the paper.

3.5.5 *Sentiment Analysis*

As an extrinsic evaluation of our PARAGRAM word vectors, we used them in a convolutional neural network (CNN) for sentiment analysis. We used the simple CNN from Kim (2014) and the binary sentence-level sentiment analysis task from Socher et al. (2013). We used the standard data splits, removing examples with a neutral rating. We trained on all constituents in the training set while only using full sentences from development and test, giving us train/development/test sizes of 67,349/872/1,821.

The CNN uses m-gram filters, each of which is an $m \times n$ vector. The CNN computes the inner product between an m-gram filter and each m-gram in an example, retaining the maximum match (so-called "max-pooling"). The score of the match is a single dimension in

9 Hill et al. (2014b) did not report the dimensionality of the vectors that led to their state-of-the-art results.
10 We did not use constraints during training.

word vectors	n	accuracy (%)
skip-gram	25	77.0
skip-gram	50	79.6
PARAGRAM	25	**80.9**

Table 5: Test set accuracies when comparing embeddings in a static CNN on the binary sentiment analysis task from Socher et al. (2013).

a feature vector for the example, which is then associated with a weight in a linear classifier used to predict positive or negative sentiment.

While Kim (2014) used m-gram filters of several lengths, we only used unigram filters. We also fixed the word vectors during learning (called "static" by Kim). After learning, the unigram filters correspond to locations in the fixed word vector space. The learned classifier weights represent how strongly each location corresponds to positive or negative sentiment. We expect this static CNN to be more effective if the word vector space separates positive and negative sentiment.

In our experiments, we compared baseline skip-gram embeddings to our PARAGRAM vectors. We used AdaGrad learning rate of 0.1, mini-batches of size 10, and a dropout rate of 0.5. We used 200 unigram filters and rectified linear units as the activation (applied to the filter output + filter bias). We trained for 30 epochs, predicting labels on the development set after each set of 3,000 examples. We recorded the highest development accuracy and used those parameters to predict labels on the test set.

Results are shown in Table 5. We see improvements over the baselines when using PARAGRAM vectors, even exceeding the performance of higher-dimensional skip-gram vectors.

3.5.6 *Scaling up to 300 Dimensions*

Increasing the dimension of word embeddings or training them on more data can have a significant positive impact on many tasks—both at the word level and on downstream tasks. We scaled up our original 25-dimensional PARAGRAM embeddings and modified our training procedure slightly in order to produce two sets of 300-dimensional PARAGRAM vectors.[11] The vectors outperform our original 25-dimensional PARAGRAM vectors on all tasks and achieve human-level performance on SL999 and WS353. Moreover, when simply using vector addition as a compositional model, they are both on par with the RNN models we trained specifically for each task. These results can be seen in Tables 12, 13, and 6.

11 Both $\text{PARAGRAM}_{300,WS353}$ and $\text{PARAGRAM}_{300,SL999}$ vectors can be found on the authors' websites.

Model			
word vectors	n	comp.	Annotated-PPDB
GloVe	300	+	0.27
PARAGRAM$_{300,WS353}$	300	+	**0.43**
PARAGRAM$_{300,SL999}$	300	+	0.41

Table 6: Evaluation of 300 dimensional PARAGRAM vectors on Annotated-PPDB.

The main modification was to use higher-dimensional initial embeddings, in our case the pretrained 300-dimensional GloVe embeddings.[12] Since PPDB only contains lowercased words, we extracted only one GloVe vector per word type (regardless of case) by taking the first occurrence of each word in the vocabulary. This is the vector for the most common casing of the word, and was used as the word's single initial vector in our experiments. This reduced the vocabulary from the original 2.2 million types to 1.7 million.

Smaller changes included replacing dot product with cosine similarity in Equation 3.4.1 and a change to the negative sampling procedure. We experimented with three approaches: *MAX sampling* discussed in Section 3.4.1, *RAND sampling* which is random sampling from the batch, and a 50/50 mixture of *MAX sampling* and *RAND sampling*.

For training data, we selected all word pairs in the lexical portion of PPDB XL that were in our vocabulary, removing redundancies. This resulted in 169,591 pairs for training. We trained our models for 10 epochs and tuned hyperparameters (batch size, λ_{W_w}, δ, and sampling method) in two ways: maximum correlation on WS353 (PARAGRAM$_{300,WS353}$) and maximum correlation on SL999 (PARAGRAM$_{300,SL999}$).[13] We report results for both sets of embeddings in Tables 12, 13, and 6, and make both available to the community in the hope that they may be useful for other downstream tasks.

3.6 EXPERIMENTS – COMPOSITIONAL PARAPHRASING

In this section, we describe experiments on a variety of compositional phrase-based paraphrasing tasks. We start with the simplest case of bigrams, and then proceed to short phrases. For all tasks, we again train on appropriate data from PPDB and test on various evaluation datasets, including our two novel datasets (Annotated-PPDB and ML-Paraphrase).

12 We used the GloVe vectors trained on 840 billion tokens of Common Crawl data, available at `http://nlp.stanford.edu/projects/glove/`

13 Note that if we use the approach in Section 3.5.3 in which we tune to maximize 2×WS-S correlation minus the WS-R correlation, the SL999 ρ is 0.640, still higher than any other reported result to the best of our knowledge.

3.6.1 *Training Procedure*

We trained our models by optimizing Eq. 3.4.1 using AdaGrad (Duchi et al., 2011). We fixed the initial learning rates to 0.5 for the word embeddings and 0.05 for the composition parameters, and the margin to 1. Then we did a coarse grid search over a parameter space for λ_{W_w}, λ_W, and mini-batch size.

For λ_{W_w}, our search space again consisted of $\{10^{-2}, 10^{-3}, ..., 10^{-7}, 0\}$, for λ_W it was $\{10^{-1}, 10^{-2}, 10^{-3}, 0\}$, and we explored batch sizes of {100, 250, 500, 1000, 2000}. When initializing with PARAGRAM vectors, the search space for λ_{W_w} was shifted upwards to be $\{10, 1, 10^{-1}, 10^{-3}, ..., 10^{-6}\}$ to reflect our increased confidence in the initial vectors. We trained only for 5 epochs for each set of parameters. For baselines, we used the same initial skip-gram vectors as in Section 3.5.

3.6.2 *Evaluation and Baselines*

For all experiments, we again used cosine similarity as our similarity metric and evaluated the statistical significance using the method of (Steiger, 1980).

A baseline used in all compositional experiments is vector addition of skip-gram (or PARAGRAM) word vectors. Unlike explicit word vectors, where point-wise multiplication acts as a conjunction of features and performs well on composition tasks (Mitchell and Lapata, 2008), using addition with skip-gram vectors (Mikolov et al., 2013b) gives better performance than multiplication.

3.6.3 *Bigram Paraphrasability*

To evaluate our ability to paraphrase bigrams, we consider the original bigram similarity task from Mitchell and Lapata (2010) as well as our newly-annotated version of it: ML-Paraphrase.

EXTRACTING TRAINING DATA Training data for these tasks was extracted from the XL portion of PPDB. The bigram similarity task from Mitchell and Lapata (2010) contains three types of bigrams: adjective-noun (JN), noun-noun (NN), and verb-noun (VN). We aimed to collect pairs from PPDB that mirrored these three types of bigrams.

Model			Mitchell and Lapata (2010) Bigrams				ML-Paraphrase			
word vectors	n	comp.	JN	NN	VN	Avg	JN	NN	VN	Avg
skip-gram	25	+	0.36	0.44	0.36	0.39	0.32	0.35	0.42	0.36
PARAGRAM	25	+	0.44*	0.34	0.48*	0.42	0.50*	0.29	**0.58***‡	0.46
PARAGRAM	25	RNN	**0.51***†	0.40†	**0.50***‡	**0.47**	**0.57***‡	**0.44**†	0.55*	**0.52**
Hashimoto et al. (2014)			0.49	0.45	0.46	**0.47**	0.38	0.39	0.45	0.41
Mitchell and Lapata (2010)			0.46	**0.49**	0.38	0.44	-	-	-	-
Human			-	-	-	-	0.87	0.64	0.73	0.75

Table 7: Results on the test section of the bigram similarity task of Mitchell and Lapata (2010) and our newly annotated version (ML-Paraphrase). (n) shows the word vector dimensionality and ("comp.") shows the composition function used: "+" is vector addition and "RNN" is the recursive neural network. The * indicates statistically significant ($p < 0.05$) over the skip-gram model, † statistically significant over the {PARAGRAM, +} model, and ‡ statistically significant over Hashimoto et al. (2014).

We found parsing to be unreliable on such short segments of text, so we used a POS tagger (Manning et al., 2014) to tag the tokens in each phrase. We then used the word alignments in PPDB to extract bigrams for training. For JN and NN, we extracted pairs containing aligned, adjacent tokens in the two phrases with the appropriate part-of-speech tag. Thus we extracted pairs like ⟨*easy job*, *simple task*⟩ for the JN section and ⟨*town meeting*, *town council*⟩ for the NN section. We used a different strategy for extracting training data for the VN subset: we took aligned VN tokens and took the closest noun after the verb. This was done to approximate the direct object that would have been ideally extracted with a dependency parse. An example from this section is ⟨*achieve goal*, *achieve aim*⟩.

We removed phrase pairs that (1) contained words not in our vocabulary, (2) were redundant with others, (3) contained brackets, or (4) had Levenshtein distance $\leqslant 1$. The final criterion helps to ensure that we train on phrase pairs with non-trivial differences. The final training data consisted of 133,997 JN pairs, 62,640 VN pairs and 35,601 NN pairs.

BASELINES In addition to RNN models, we report baselines that use vector addition as the composition function, both with our skip-gram embeddings and PARAGRAM embeddings from Section 3.5.

We also compare to several results from prior work. When doing so, we took their *best* correlations for each data subset. That is, the JN and NN results from Mitchell and Lapata (2010) use their multiplicative model and the VN results use their dilation model. From Hashimoto et al. (2014) we used their PAS-CLBLM Add_l and PAS-CLBLM Add_{nl} models. We note that their vector dimensionalities are larger than ours, using $n = 2000$ and 50 respectively.

RESULTS Results are shown in Table 7. We report results on the test portion of the original Mitchell and Lapata (2010) dataset (ML) as well as the entirety of our newly-annotated dataset (ML-Paraphrase). RNN results on ML were tuned on the respective development sections and RNN results on ML-Paraphrase were tuned on the entire ML dataset.

Our RNN model outperforms results from the literature on most sections in both datasets and its average correlations are among the highest.[14] The one subset of the data that posed difficulty was the NN section of the ML dataset. We suspect this is due to the reasons discussed in Section 3.3.2; for our ML-Paraphrase dataset, by contrast, we do see gains on the NN section.

We also outperform the strong baseline of adding 1000-dimensional skip-gram embeddings, a model with 40 times the number of parameters, on our ML-Paraphrase dataset. This baseline had correlations of 0.45, 0.43, and 0.47 on the JN, NN, and VN partitions, with an average of 0.45—below the average ρ of the RNN (0.52) and even the {PARAGRAM, +} model (0.46).

Interestingly, the type of vectors used to initialize the RNN has a significant effect on performance. If we initialize using the 25-dimensional skip-gram vectors, the average ρ on ML-Paraphrase drops to 0.43, below even the {PARAGRAM, +} model.

3.6.4 *Phrase Paraphrasability*

In this section we show that by training a model based on filtered phrase pairs in PPDB, we can actually distinguish between quality paraphrases and poor paraphrases in PPDB better than the original heuristic scoring scheme from Ganitkevitch et al. (2013).

EXTRACTING TRAINING DATA As before, training data was extracted from the XL section of PPDB. Similar to the procedure to create our Annotated-PPDB dataset, phrases were filtered such that only those with a *word overlap score* of less than 0.5 were kept. We also removed redundant phrases and phrases that contained tokens not in our vocabulary. The phrases were then binned according to their *effective size* and 20,000 examples were selected from bins of *effective sizes* of 3, 4, and more than 5, creating a training set of 60,000 examples. Care was taken to ensure that none of our training pairs was also present in our development and test sets.

14 The results obtained here differ from those reported in Hashimoto et al. (2014) as we scored their vectors with a newer Python implementation of Spearman ρ that handles ties (Hashimoto, P.C.).

BASELINES We compare our models with strong lexical baselines. The first, *strict word overlap*, is the percentage of words in the smaller phrase that are also in the larger phrase. We also include a version where the words are lemmatized prior to the calculation.

We also train a support vector regression model (epsilon-SVR) (Chang and Lin, 2011) on the 33 features that are included for each phrase pair in PPDB. We scaled the features such that each lies in the interval $[-1, 1]$ and tuned the parameters using 5-fold cross validation on our dev set.[15] We then trained on the entire dev set after finding the best performing C and ϵ combination and evaluated on the test set of Annotated-PPDB.

Model			
word vectors	n	comp.	Annotated-PPDB
skip-gram	25	+	0.20
PARAGRAM	25	+	0.32^{*}
PARAGRAM	25	RNN	**0.40***†‡
Ganitkevitch et al. (2013)			0.25
word overlap (strict)			0.26
word overlap (lemmatized)			0.20
PPDB+SVR			0.33

Table 8: Spearman correlation on Annotated-PPDB. The * indicates statistically significant ($p < 0.05$) over the skip-gram model, the † indicates statistically significant over the {PARAGRAM, +} model, and the ‡ indicates statistically significant over PPDB+SVR.

RESULTS We evaluated on our Annotated-PPDB dataset described in section 3.3.1. Table 8 shows the Spearman correlations on the 1000-example test set. RNN models were tuned on the development set of 260 examples. All other methods had no hyperparameters and therefore required no tuning.

We note that the confidence estimates from Ganitkevitch et al. (2013) reach a ρ of 0.25 on the test set, similar to the results of strict overlap. While 25-dimensional skip-gram embeddings only reach 0.20, we can improve this to 0.32 by fine-tuning them using PPDB (thereby obtaining our PARAGRAM vectors). By using the PARAGRAM vectors to initialize the RNN, we reach a correlation of 0.40, which is better than the PPDB confidence estimates by 15% absolute.

We again consider addition of 1000-dimensional skip-gram embeddings as a baseline, and they continue to perform strongly ($\rho = 0.37$). The RNN initialized with PARAGRAM vectors does reach a higher ρ (0.40), but the difference is not statistically significant ($p = 0.16$). Thus we can achieve similarly-strong results with far fewer parameters.

15 We tuned both parameters over $\{2^{-10}, 2^{-9}, ..., 2^{10}\}$.

This task also illustrates the importance of initializing our RNN model with appropriate word embeddings. An RNN initialized with skip-gram vectors has a modest ρ of 0.22, well below the ρ of the RNN initialized with PARAGRAM vectors. Clearly, initialization is important when optimizing non-convex objectives like ours, but it is noteworthy that our best results came from first improving the word vectors and then learning the composition model, rather than jointly learning both from scratch.

3.7 QUALITATIVE ANALYSIS

Score Range	+	RNN
$[1,2)$	2.35	2.08
$[2,3)$	1.56	1.38
$[3,4)$	0.87	0.85
$[4,5]$	0.43	0.47

Table 9: Average absolute error of addition and RNN models on different ranges of gold scores.

Index	Phrase 1	Phrase 2	Length Ratio	Overlap Ratio	Gold	RNN	+
1	scheduled to be held in	that will take place in	1.0	0.4	4.6	2.9	**4.4**
2	according to the paper ,	the newspaper reported that	0.8	0.5	4.6	2.8	**4.1**
3	at no cost to	without charge to	0.75	1.0	4.8	3.1	**4.6**
4	's surname	family name of	0.67	1.0	4.4	2.8	**4.1**
5	could have an impact on	may influence	0.4	0.5	4.6	**4.2**	3.2
6	to participate actively	to play an active role	0.6	0.67	5.0	**4.8**	4.0
7	earliest opportunity	early as possible	0.67	0.0	4.4	**4.3**	2.9
8	does not exceed	is no more than	0.75	0.0	5.0	**4.8**	3.5

Table 10: Illustrative phrase pairs from Annotated-PPDB with gold similarity > 4. The last three columns show the gold similarity score, the similarity score of the RNN model, and the similarity score of vector addition. We note that addition performs better when the pairs have high length ratio (rows 1–2) or overlap ratio (rows 3–4) while the RNN does better when those values are low (rows 5–6 and 7–8 respectively). Boldface indicates smaller error compared to gold scores.

We performed a qualitative analysis to uncover sources of error and determine differences between adding PARAGRAM vectors and using an RNN initialized with them. To do so, we took the output of both systems on Annotated-PPDB and mapped their cosine similarities to the interval $[1,5]$. We then computed their absolute error as compared to the gold ratings.

Table 9 shows how the average of these absolute errors changes with the magnitude of the gold ratings. The RNN performs better (has lower average absolute error) for less similar pairs. Vector addition only does better on the most similar pairs. This is presumably because the most positive pairs have high word overlap and so can be represented effectively with a simpler model.

To further investigate the differences between these models, we removed those pairs with gold scores in $[2, 4]$, in order to focus on pairs with extreme scores. We identified two factors that distinguished the performance between the two models: length ratio and the amount of lexical overlap. We did not find evidence that non-compositional phrases, such as idioms, were a source of error as these were not found in ML-Paraphrase and only appear rarely in Annotated-PPDB.

We define length ratio as simply the number of tokens in the smaller phrase divided by the number of tokens in the larger phrase. Overlap ratio is the number of *equivalent tokens* in the phrase pair divided by the number of tokens in the smaller of the two phrases. *Equivalent tokens* are defined as tokens that are either exact matches or are paired up in the lexical portion of PPDB used to train the PARAGRAM vectors.

Table 11 shows how the performance of the models changes under different values of length ratio and overlap ratio.[16] The values in this table are the percentage changes in absolute error when using the RNN over the PARAGRAM vector addition model. So negative values indicate superior performance by the RNN.

Length Ratio	$[0, 0.6]$	$(0.6, 0.8]$	$(0.8, 1]$
Positive Examples	-22.4	10.0	35.5
Negative Examples	-9.9	-11.1	-12.2
Both	-13.0	-6.4	-2.0
Overlap Ratio	$[0,\frac{1}{3}]$	$(\frac{1}{3},\frac{2}{3}]$	$(\frac{2}{3},1]$
Positive Examples	-4.5	7.0	19.4
Negative Examples	-11.3	-7.5	-15.0
Both	-10.6	-5.3	0.0

Table 11: Comparison of the addition and RNN model on phrase pairs of different overlap and length ratios. The values in the table are the percent change in absolute error from the addition model to the RNN model. Negative examples are defined as pairs from Annotated-PPDB whose gold score is less than 2 and positive examples are those with scores greater than 4. "Both" refers to both negative and positive examples.

A few trends emerge from this table. One is that as the length ratio increases (i.e., the phrase pairs are closer in length), addition surpasses the RNN for positive examples. For negative examples, the trend is reversed. The same trend appears for overlap ratio. Examples from Annotated-PPDB illustrating these trends on positive examples are shown in Table 10.

16 The bin delimiters were chosen to be uniform over the range of output values of the length ratio ([0.4,1] with one outlier data point removed) and overlap ratio ([0,1]).

Model	n	SL999	WS353	WS-S	WS-R
GloVe	300	0.376	0.579	0.630	0.571
PARAGRAM$_{300,WS353}$	300	0.667	**0.769**	**0.814**	**0.730**
PARAGRAM$_{300,SL999}$	300	**0.685**	0.720	0.779	0.652
inter-annotator agreement*	N/A	0.67	0.756	N/A	N/A

Table 12: Evaluation of 300 dimensional PARAGRAM vectors on SL999 and WS353. Note that the inter-annotator agreement ρ was calculated differently for WS353 and SL999. For SL999, the agreement was computed as the average pairwise correlation between pairs of annotators, while for WS353, agreement was computed as the average correlation between a single annotator with the average over all other annotators. If one uses the alternative measure of agreement for WS353, the agreement is 0.611, which is easily beaten by automatic methods (Hill et al., 2015).

Model			Mitchell and Lapata (2010) Bigrams				ML-Paraphrase			
word vectors	n	comp.	JN	NN	VN	Avg	JN	NN	VN	Avg
GloVe	300	+	0.40	0.46	0.37	0.41	0.39	0.36	0.45	0.40
PARAGRAM$_{300,WS353}$	300	+	0.52	0.41	0.49	**0.48**	0.55	0.42	0.55	0.51
PARAGRAM$_{300,SL999}$	300	+	0.51	0.36	0.51	0.46	0.57	0.39	0.59	**0.52**

Table 13: Evaluation of 300 dimensional PARAGRAM vectors on the bigram tasks.

When considering both positive and negative examples ("Both"), we see that the RNN excels on the most difficult examples (large differences in phrase length and less lexical overlap). For easier examples, the two fare similarly overall (-2.0 to 0.0% change), but the RNN does much better on negative examples. This aligns with the intuition that addition should perform well when two paraphrastic phrases have high lexical overlap and similar length. But when they are not paraphrases, simple addition is misled and the RNN's learned composition function better captures the relationship. This may suggest new architectures for modeling compositionality differently depending on differences in length and amount of overlap.

3.8 CONCLUSION

We have shown how to leverage PPDB to learn state-of-the-art word embeddings and compositional models for paraphrase tasks. Since PPDB was created automatically from parallel corpora, our models are also built automatically. Only small amounts of annotated data are used to tune hyperparameters.

We also introduced two new datasets to evaluate compositional models of short paraphrases, filling a gap in the NLP community, as currently there are no datasets created for this purpose. Successful models on these datasets can then be used to extend the coverage of, or provide an alternative to, PPDB.

There remains a great deal of work to be done in developing new composition models, whether with new network architectures or distance functions. In this work, we based our composition function on constituent parse trees, but this may not be the best approach—especially for short phrases. Dependency syntax may be a better alternative (Socher et al., 2014). Besides improving composition, another direction to explore is how to use models for short phrases in sentence-level paraphrase recognition and other downstream tasks.

4

PARAGRAM SENTENCE REPRESENTATIONS DEFINED AND EXPLORED

In this chapter, we discuss our first models for learning *paraphrastic* representations. This chapter encompasses work that has appeared in three papers: (Wieting et al., 2015, 2016b,a).

While (Wieting et al., 2015), is not discussed in detail in this chapter, a lot of the work in this paper formed the backbone of this thesis. In this work, we find that state-of-the-art phrase embeddings could be learned by using text snippets from the Paraphrase Database (PPDB) (Ganitkevitch et al., 2013) as training data. PPDB is created by bilingual pivoting (Bannard and Callison-Burch, 2005) on parallel bitext, where two text snippets are paraphrases when they align to the same snippet in another language. While the recursive RNN (Socher et al., 2011) architecture in this paper was overly complicated (which is shown in later papers), we laid the groundwork with a focus on paraphrastic similarity, our training strategy, and our proposed objective function. We use the term paraphrastic similarity to distinguish it from other types of similarity which were largely inter-mixed at the time - specifically separating the notion of *relatedness* from paraphrastic where relatedness refers to words that appear in similar contexts (*table* and *chair*) while paraphrastic refers to words that could be paraphrases in some context (*chair* and *seat*). One other important contribution from that paper were PARAGRAM word embeddings. These significantly surpassed the state-of-the-art word embedding approaches for measuring similarity and are even the basis for state-of-the-art models today.

In (Wieting et al., 2016b). we extended (Wieting et al., 2015) to learn sentence embeddings. Our primary evaluation was on a suite of semantic textual similarity (STS) datasets but we also follow the setting of (Kiros et al., 2015) and show that we can rival their performance with our simpler model. This is also the first paper to use these suite of tasks as an evaluation approach for sentence embeddings, which has become a standard. We experimented with a wide variety of encoding architectures, and we fount that the simplest approach, averaging word embeddings, to be the most effective and are able to surpass the vast majority of proposed STS systems despite being unsupervised. This was a surprising result, considering LSTM (Hochreiter and Schmidhuber, 1997) were setting new state-of-the-arts for many problems in natural language processing. In this paper, we analyzed some hypotheses to explain these results, finding that the reason wasn't due to length, over-fitting, or insufficient parameter tuning. LSTMs (and deeper architectures in general) can work well on this problem, but the training strategies need to change. This is further discussed in Chapter 8 and Chapter 10.

Lastly, we discuss our CHARAGRAM models for learning word and sentence representations. In these representations, we represent a character sequence by a vector containing counts of character n-grams, inspired by Huang et al. (2013). This vector is embedded into a low-dimensional space using a single nonlinear transformation. This can be interpreted as learning embeddings of character n-grams, which are learned so as to produce effective sequence embeddings when a summation is performed over the character n-grams in the sequence. We evaluate on three tasks: word similarity, sentence similarity, and part-of-speech tagging and show that CHARAGRAM outperforms character RNNs, character CNNs, as well as PARAGRAM embeddings. We find that modelling subwords yields large gains in performance for rare words and can easily handle spelling variation, morphology, and word choice.

4.1 INTRODUCTION

Word embeddings have become ubiquitous in natural language processing (NLP). Several researchers have developed and shared word embeddings trained on large datasets (Collobert et al., 2011; Mikolov et al., 2013b; Pennington et al., 2014), and these have been used effectively for many downstream tasks (Turian et al., 2010; Socher et al., 2011; Kim, 2014; Bansal et al., 2014; Tai et al., 2015). There has also been recent work on creating representations for word sequences such as phrases or sentences. Many functional architectures have been proposed to model compositionality in such sequences, ranging from those based on simple operations like addition (Mitchell and Lapata, 2010; Yu and Dredze, 2015; Iyyer et al., 2015) to those based on richly-structured functions like recursive neural networks (Socher et al., 2011), convolutional neural networks (Kalchbrenner et al., 2014), and recurrent neural networks using long short-term memory (LSTM) (Tai et al., 2015). However, there is little work on learning sentence representations that can be used across domains with the same ease and effectiveness as word embeddings. In this chapter, we explore compositional models that can encode arbitrary word sequences into a vector with the property that sequences with similar meaning have high cosine similarity, and that can, importantly, also transfer easily across domains. We consider six compositional architectures based on neural networks and train them on noisy phrase pairs from the Paraphrase Database (PPDB; Ganitkevitch et al., 2013).

We consider models spanning the range of complexity from word averaging to LSTMs. With the simplest word averaging model, there are no additional compositional parameters. The only parameters are the word vectors themselves, which are learned to produce effective sequence embeddings when averaging is performed over the sequence. We add complexity by adding layers, leading to variants of deep averaging networks (Iyyer et al., 2015). We next consider several recurrent network variants, culminating in LSTMs because they have been found to be effective for many types of sequential data (Graves et al., 2008, 2013; Greff et al., 2015), including text (Sutskever et al., 2014; Vinyals et al., 2014; Xu et al., 2015a; Hermann et al., 2015; Ling et al., 2015a; Wen et al., 2015).

To evaluate our models, we consider two tasks drawn from the same distribution as the training data, as well as 22 SemEval textual similarity datasets from a variety of domains (such as news, tweets, web forums, and image and video captions). Interestingly, we find that the LSTM performs well on the in-domain task, but performs much worse on the out-of-domain tasks. We discover surprisingly strong performance for the models based on word averaging, which perform well on both the in-domain and out-of-domain tasks, beating the best LSTM model by 16.5 Pearson's r on average. Moreover, we find that learning word embeddings in the context of vector averaging performs much better than simply averaging pretrained, state-of-the-art word embeddings. Our average Pearson's r over all 22 SemEval datasets is 17.1 points higher than averaging GloVe vectors[1] and 12.8 points higher than averaging PARAGRAM-SL999 vectors.[2]

Our final sentence embeddings[3] place in the top 25% of all submitted systems in every SemEval STS task from 2012 through 2015, being best or tied for best on 4 of the datasets.[4] This is surprising because the submitted systems were designed for those particular tasks, with access to training and tuning data specifically developed for each task.

While the above experiments focus on transfer, we also consider the fully supervised setting (Table 17). We compare the same suite of compositional architectures for three supervised NLP tasks: sentence similarity and textual entailment using the 2014 SemEval SICK

1 We used the publicly available 300-dimensional vectors that were trained on the 840 billion token Common Crawl corpus, available at `http://nlp.stanford.edu/projects/glove/`.

2 These are 300-dimensional vectors from Chapter 3 and are available at `https://www.cs.cmu.edu/~jwieting`. They give human-level performance on two commonly used word similarity datasets, WordSim353 (Finkelstein et al., 2001) and Simlex-999 (Hill et al., 2015).

3 Denoted PARAGRAM-PHRASE-XXL and discussed in Section 4.4.3.

4 As measured by the average Pearson's r over all datasets in each task; see Table 16.

dataset (Marelli et al., 2014), and sentiment classification using the Stanford Sentiment Treebank (Socher et al., 2013). We again find strong performance for the word averaging models for both similarity and entailment, outperforming the LSTM. However, for sentiment classification, we see a different trend. The LSTM now performs best, achieving 89.2% on the coarse-grained sentiment classification task. This result, to our knowledge, is the new state of the art on this task.

We then demonstrate how to combine our PPDB-trained sentence embedding models with supervised NLP tasks. We first use our model as a prior, yielding performance on the similarity and entailment tasks that rivals the state of the art. We also use our sentence embeddings as an effective black box feature extractor for downstream tasks, comparing favorably to recent work (Kiros et al., 2015).

We release our strongest sentence embedding model, which we call PARAGRAM-PHRASE XXL, to the research community.[5] Since it consists merely of a new set of word embeddings, it is extremely efficient and easy to use for downstream applications. Our hope is that this model can provide a new simple and strong baseline in the quest for universal sentence embeddings.

4.2 RELATED WORK

Researchers have developed many ways to embed word sequences for NLP. They mostly focus on the question of compositionality: given vectors for words, how should we create a vector for a word sequence? (Mitchell and Lapata, 2008, 2010) considered bigram compositionality, comparing many functions for composing two word vectors into a single vector to represent their bigram. Follow-up work by (Blacoe and Lapata, 2012) found again that simple operations such as vector addition performed strongly. Many other compositional architectures have been proposed. Some have been based on distributional semantics (Baroni et al., 2014; Paperno et al., 2014; Polajnar et al., 2015; Tian et al., 2015), while the current trend is toward development of neural network architectures. These include neural bag-of-words models (Kalchbrenner et al., 2014), deep averaging networks (DANs) (Iyyer et al., 2015), feature-weighted averaging (Yu and Dredze, 2015), recursive neural networks based on parse structure (Socher et al., 2011, 2012, 2013; Irsoy and Cardie, 2014; Wieting et al., 2015), recursive networks based on non-syntactic hierarchical structure (Zhao et al., 2015; Chen et al., 2015b), convolutional neural networks (Kalchbrenner et al., 2014; Kim, 2014; Hu

5 Available at `https://www.cs.cmu.edu/~jwieting`.

et al., 2014b; Yin and Schütze, 2015; He et al., 2015), and recurrent neural networks using long short-term memory (Tai et al., 2015; Ling et al., 2015a; Liu et al., 2015). In this paper, we compare six architectures: word averaging, word averaging followed by a single linear projection, DANs, and three variants of recurrent neural networks, including LSTMs.[6]

Most of the work mentioned above learns compositional models in the context of *supervised* learning. That is, a training set is provided with annotations and the composition function is learned for the purposes of optimizing an objective function based on those annotations. The models are then evaluated on a test set drawn from the same distribution as the training set.

In this paper, in contrast, we are primarily interested in creating general purpose, domain independent embeddings for word sequences. There have been research efforts also targeting this goal. One approach is to train an autoencoder in an attempt to learn the latent structure of the sequence, whether it be a sentence with a parse tree (Socher et al., 2011), or a longer sequence such as a paragraph or document (Li et al., 2015). Other recently proposed methods, including paragraph vectors (Le and Mikolov, 2014) and skip-thought vectors (Kiros et al., 2015), learn sequence representations that are predictive of words inside the sequence or in neighboring sequences. These methods produce generic representations that can be used to provide features for text classification or sentence similarity tasks. While skip-thought vectors capture similarity in terms of discourse context, in this paper we are interested in capturing paraphrastic similarity, i.e., whether two sentences have the same meaning.

Our learning formulation draws from a large body of related work on learning input representations in order to maximize similarity in the learned space (Weston et al., 2010; Yih et al., 2011; Huang et al., 2013; Hermann and Blunsom, 2014; Socher et al., 2014; Faruqui and Dyer, 2014; Bordes et al., 2014b,a; Lu et al., 2015), including our prior work in Chapter 3. We focus our exploration here on modeling and keep the learning methodology mostly fixed, though we do include certain choices about the learning procedure in our hyperparameter tuning space for each model.

6 In prior work, we experimented with recursive neural networks on binarized parses of the PPDB (Chapter 3), but we found that many of the phrases in PPDB are not sentences or even constituents, causing the parser to have unexpected behavior.

4.3 MODELS AND TRAINING

Our goal is to embed sequences into a low-dimensional space such that cosine similarity in the space corresponds to the strength of the paraphrase relationship between the sequences. We experimented with six models of increasing complexity. The simplest model embeds a word sequence $x = \langle x_1, x_2, ..., x_n \rangle$ by averaging the vectors of its tokens. The only parameters learned by this model are the word embedding matrix W_w:

$$g_{\text{PARAGRAM-PHRASE}}(x) = \frac{1}{n}\sum_{i}^{n} W_w^{x_i}$$

where $W_w^{x_i}$ is the word embedding for word x_i. We call the learned embeddings PARAGRAM-PHRASE embeddings.

In our second model, we learn a projection in addition to the word embeddings:

$$g_{\text{proj}}(x) = W_p \left(\frac{1}{n}\sum_{i}^{n} W_w^{x_i} \right) + b$$

where W_p is the projection matrix and b is a bias vector.

Our third model is the deep averaging network (DAN) of (Iyyer et al., 2015). This is a generalization of the above models that typically uses multiple layers as well as nonlinear activation functions. In our experiments below, we tune over the number of layers and choice of activation function.

Our fourth model is a standard recurrent network (RNN) with randomly initialized weight matrices and nonlinear activations:

$$h_t = f(W_x W_w^{x_t} + W_h h_{t-1} + b)$$
$$g_{\text{RNN}}(x) = h_{-1}$$

where f is the activation function (either tanh or rectified linear unit; the choice is tuned), W_x and W_h are parameter matrices, b is a bias vector, and h_{-1} refers to the hidden vector of the last token.

Our fifth model is a special RNN which we call an *identity-RNN*. In the identity-RNN, the weight matrices are initialized to identity, the bias is initialized to zero, and the activation is the identity function. We divide the final output vector of the identity-RNN by the number of tokens in the sequence. Thus, before any updates to the parameters, the identity-RNN simply averages the word embeddings. We also regularize the identity-RNN parameters to their initial values. The idea is that, with high regularization, the identity-RNN is simply averaging word embeddings. However, it is a richer architecture and can take into account word order and hopefully improve upon the averaging baseline.

Our sixth and final model is the most expressive. We use long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997), a recurrent neural network (RNN) architecture designed to model sequences with long-distance dependencies. LSTMs have recently been shown to produce state-of-the-art results in a variety of sequence processing tasks (Chen et al., 2015a; Filippova et al., 2015; Xu et al., 2015c; Belinkov and Glass, 2015; Wang and Nyberg, 2015). We use the version from (Gers et al., 2003) which has the following equations:

$$\begin{aligned}
i_t &= \sigma\left(W_{xi}W_w^{x_t} + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)\\
f_t &= \sigma\left(W_{xf}W_w^{x_t} + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)\\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}W_w^{x_t} + W_{hc}h_{t-1} + b_c\right)\\
o_t &= \sigma\left(W_{xo}W_w^{x_t} + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)\\
h_t &= o_t \tanh(c_t)\\
g_{\text{LSTM}}(x) &= h_{-1}
\end{aligned}$$

where σ is the logistic sigmoid function. We found that the choice of whether or not to include the output gate had a significant impact on performance, so we used two versions of the LSTM model, one with the output gate and one without. For all models, we learn the word embeddings themselves, denoting the trainable word embedding parameters by W_w. We denote all other trainable parameters by W_c ("compositional parameters"), though the PARAGRAM-PHRASE model has no compositional parameters. We initialize W_w using some embeddings pretrained from large corpora.

4.3.1 *Training*

We mostly follow the approach of (Wieting et al., 2015). The training data consists of (possibly noisy) pairs taken directly from the original Paraphrase Database (PPDB) and we optimize a margin-based loss.

Our training data consists of a set X of phrase pairs $\langle x_1, x_2\rangle$, where x_1 and x_2 are assumed to be paraphrases. The objective function follows:

$$\min_{W_c, W_w} \frac{1}{|X|}\Bigg(\sum_{\langle x_1, x_2\rangle \in X} \max(0, \delta - \cos(g(x_1), g(x_2)) + \cos(g(x_1), g(t_1))) \\ + \max(0, \delta - \cos(g(x_1), g(x_2)) + \cos(g(x_2), g(t_2)))\Bigg) \\ + \lambda_c \|W_c\|^2 + \lambda_w \|W_{w_{initial}} - W_w\|^2$$

where g is the embedding function in use (e.g., g_{LSTM}), δ is the margin, λ_c and λ_w are regularization parameters, $W_{w_{initial}}$ is the initial word embedding matrix, and t_1 and t_2 are carefully-selected negative examples taken from a mini-batch during optimization. The intuition is that we want the two phrases to be more similar to each other ($\cos(g(x_1), g(x_2))$) than either is to their respective negative examples t_1 and t_2, by a margin of at least δ.

4.3.1.1 *Selecting Negative Examples*

To select t_1 and t_2 in Eq. 4.3.1, we tune the choice between two approaches. The first, MAX, simply chooses the most similar phrase in some set of phrases (other than those in the given phrase pair). For simplicity and to reduce the number of tunable parameters, we use the mini-batch for this set, but it could be a separate set. Formally, MAX corresponds to choosing t_1 for a given $\langle x_1, x_2\rangle$ as follows:

$$t_1 = \underset{t:\langle t, \cdot\rangle \in X_b \setminus \{\langle x_1, x_2\rangle\}}{\text{argmax}} \cos(g(x_1), g(t))$$

where $X_b \subseteq X$ is the current mini-batch. That is, we want to choose a negative example t_i that is similar to x_i according to the current model parameters. The downside of this approach is that we may occasionally choose a phrase t_i that is actually a true paraphrase of x_i.

The second strategy selects negative examples using MAX with probability 0.5 and selects them randomly from the mini-batch otherwise. We call this sampling strategy MIX. We tune over the strategy in our experiments.

4.4 EXPERIMENTS

4.4.1 *Data*

We experiment on 24 textual similarity datasets, covering many domains, including all datasets from every SemEval semantic textual similarity (STS) task (2012-2015). We also evaluate on the SemEval 2015 Twitter task (Xu et al., 2015b) and the SemEval 2014 Semantic Relatedness task (Marelli et al., 2014), as well as two tasks that use PPDB data (Wieting et al., 2015; Pavlick et al., 2015).

For tuning, we use two datasets that contain PPDB phrase pairs scored by human annotators on the strength of their paraphrase relationship. One is a large sample of 26,456 annotated phrase pairs developed by (Pavlick et al., 2015). The second, called Annotated-PPDB, was developed in our prior work in Chapter 3 and is a small set of 1,000 annotated phrase pairs that were filtered to focus on challenging paraphrase phenomena.

4.4.2 *Transfer Learning*

4.4.2.1 *Experimental Settings*

As training data, we used the XL section[7] of PPDB which contains 3,033,753 unique phrase pairs. However, for hyperparameter tuning we only used 100k examples sampled from PPDB XXL and trained for 5 epochs. Then after finding the hyperparameters that maximize Spearman's ρ on the Pavlick et al. PPDB task, we trained on the entire XL section of PPDB for 10 epochs. We used PARAGRAM-SL999 embeddings to initialize the word embedding matrix (W_w) for all models.

We chose the Pavlick et al. task for tuning because we wanted our entire procedure to only make use of PPDB and use no other resources. In particular, we did not want to use any STS tasks for training or hyperparameter tuning. We chose the Pavlick et al. dataset over Annotated-PPDB due to its larger size. But in practice the datasets are very similar and tuning on either produces similar results.

7 PPDB comes in different sizes (S, M, L, XL, XXL, and XXXL), where each larger size subsumes all smaller ones. The phrases are sorted by a confidence measure and so the smaller sets contain higher precision paraphrases.

To learn model parameters for all experiments in this section, we minimize Eq. 4.3.1. Our models have the following tunable hyperparameters:[8] λ_c, the L_2 regularizer on the compositional parameters W_c (not applicable for the word averaging model), the pool of phrases used to obtain negative examples (coupled with mini-batch size B, to reduce the number of tunable hyperparameters), λ_w, the regularizer on the word embeddings, and δ, the margin. We also tune over optimization method (either AdaGrad (Duchi et al., 2011) or Adam (Kingma and Ba, 2014)), learning rate (from $\{0.05, 0.005, 0.0005\}$), whether to clip the gradients with threshold 1 (Pascanu et al., 2012), and whether to use MIX or MAX sampling. For the classic RNN, we further tuned whether to use tanh or rectified linear unit activation functions; for the identity-RNN, we tuned λ_c over $\{1000, 100, 10, 1\}$ because we wanted higher regularization on the composition parameters; for the DANs we tuned over activation function (tanh or rectified linear unit) and the number of layers (either 1 or 2); for the LSTMs we tuned on whether to include an output gate. We fix the output dimensionalities of all models that require doing so to the dimensionality of our word embeddings (300).

4.4.2.2 *Results*

The results on all STS tasks as well as the SICK and Twitter tasks are shown in Table 14. We include results on the PPDB tasks in Table 15. In Table 14, we first show the median, 75th percentile, and highest score from the official task rankings. We then report the performance of our seven models: PARAGRAM-PHRASE (PP), identity-RNN (iRNN), projection (proj.), deep-averaging network (DAN), recurrent neural network (RNN), LSTM with output gate (o.g.), and LSTM without output gate (no o.g.). We compare to three baselines: skip-thought vectors[9] (Kiros et al., 2015), denoted "ST", averaged GloVe[10] vectors (Pennington et al., 2014), and averaged PARAGRAM-SL999 vectors from Chapter 3, denoted "PSL". Note that the GloVe vectors were used to initialize the PARAGRAM-SL999 vectors which were, in turn, used to initialize our PARAGRAM-PHRASE embeddings. We compare to skip-thought vectors because trained models are publicly available and they show impressive performance when used as features on several tasks including textual similarity.

8 For λ_c we searched over $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$, for b we searched over $\{25, 50, 100\}$, for λ_w we searched over $\{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$ as well as the setting in which we do not update W_w, and for δ we searched over $\{0.4, 0.6, 0.8\}$.

9 Note that we pre-processed the training data with the tokenizer from Stanford CoreNLP (Manning et al., 2014) rather than the included NLTK (Bird et al., 2009) tokenizer. We found that doing so significantly improves the performance of the skip-thought vectors.

10 We used the publicly available 300-dimensional vectors that were trained on the 840 billion token Common Crawl corpus, available at `http://nlp.stanford.edu/projects/glove/`.

Dataset	50%	75%	Max	PP	proj.	DAN	RNN	iRNN	LSTM (no o.g.)	LSTM (o.g.)	ST	GloVe	PSL
MSRpar	51.5	57.6	73.4	42.6	43.7	40.3	18.6	43.4	16.1	9.3	16.8	**47.7**	41.6
MSRvid	75.5	80.3	88.0	**74.5**	74.0	70.0	66.5	73.4	71.3	71.3	41.7	63.9	60.0
SMT-eur	44.4	48.1	56.7	47.3	**49.4**	43.8	40.9	47.1	41.8	44.3	35.2	46.0	42.4
OnWN	608	65.9	72.7	**70.6**	70.1	65.9	63.1	70.1	65.2	56.4	29.7	55.1	63.0
SMT-news	40.1	45.4	60.9	58.4	**62.8**	60.0	51.3	58.1	60.8	51.0	30.8	49.6	57.0
STS 2012 Average	54.5	59.5	70.3	58.7	**60.0**	56.0	48.1	58.4	51.0	46.4	30.8	52.5	52.8
headline	64.0	68.3	78.4	72.4	72.6	71.2	59.5	**72.8**	57.4	48.5	34.6	63.8	68.8
OnWN	52.8	64.8	84.3	67.7	68.0	64.1	54.6	**69.4**	68.5	50.4	10.0	49.0	48.0
FNWN	32.7	38.1	58.2	43.9	**46.8**	43.1	30.9	45.3	24.7	38.4	30.4	34.2	37.9
SMT	31.8	34.6	40.4	39.2	**39.8**	38.3	33.8	39.4	30.1	28.8	24.3	22.3	31.0
STS 2013 Average	45.3	51.4	65.3	55.8	**56.8**	54.2	44.7	56.7	45.2	41.5	24.8	42.3	46.4
deft forum	36.6	46.8	53.1	48.7	**51.1**	49.0	41.5	49.0	44.2	46.1	12.9	27.1	37.2
deft news	66.2	74.0	78.5	**73.1**	72.2	71.7	53.7	72.4	52.8	39.1	23.5	68.0	67.0
headline	67.1	75.4	78.4	69.7	**70.8**	69.2	57.5	70.2	57.5	50.9	37.8	59.5	65.3
images	75.6	79.0	83.4	**78.5**	78.1	76.9	67.6	78.2	68.5	62.9	51.2	61.0	62.0
OnWN	78.0	81.1	87.5	78.8	**79.5**	75.7	67.7	78.8	76.9	61.7	23.3	58.4	61.1
tweet news	64.7	72.2	79.2	76.4	75.8	74.2	58.0	**76.9**	58.7	48.2	39.9	51.2	64.7
STS 2014 Average	64.7	71.4	76.7	70.9	**71.3**	69.5	57.7	70.9	59.8	51.5	31.4	54.2	59.5
answers-forums	61.3	68.2	73.9	**68.3**	65.1	62.6	32.8	67.4	51.9	50.7	36.1	30.5	38.8
answers-students	67.6	73.6	78.8	**78.2**	77.8	78.1	64.7	78.2	71.5	55.7	33.0	63.0	69.2
belief	67.7	72.2	77.2	**76.2**	75.4	72.0	51.9	75.9	61.7	52.6	24.6	40.5	53.2
headline	74.2	80.8	84.2	74.8	**75.2**	73.5	65.3	75.1	64.0	56.6	43.6	61.8	69.0
images	80.4	84.3	87.1	**81.4**	80.3	77.5	71.4	81.1	70.4	64.2	17.7	67.5	69.9
STS 2015 Average	70.2	75.8	80.2	**75.8**	74.8	72.7	57.2	75.6	63.9	56.0	31.0	52.7	60.0
2014 SICK	71.4	79.9	82.8	71.6	**71.6**	70.7	61.2	71.2	63.9	59.0	49.8	65.9	66.4
2015 Twitter	49.9	52.5	61.9	52.9	52.8	**53.7**	45.1	52.9	47.6	36.1	24.7	30.3	36.3

Table 14: Results on SemEval textual similarity datasets (Pearson's $r \times 100$). The highest score in each row is in boldface (omitting the official task score columns).

The results in Table 14 show strong performance of our two simplest models: the PARAGRAM-PHRASE embeddings (PP) and our projection model (proj.). They outperform the other models on all but 5 of the 22 datasets. The iRNN model has the next best performance, while the LSTM models lag behind. These results stand in marked contrast to those in Table 15, which shows very similar performance across models on the in-domain PPDB tasks, with the LSTM models slightly outperforming the others. For the LSTM models, it is also interesting to note that removing the output gate results in stronger performance on the textual similarity tasks. Removing the output gate improves performance on 18 of the 22 datasets. The LSTM without output gate also performs reasonably well compared to our strong PARAGRAM-SL999 addition baseline, beating it on 12 of the 22 datasets.

Model	Pavlick et al. (oracle)	Pavlick et al. (test)	Annotated-PPDB (test)
PARAGRAM-PHRASE	60.3	60.0	53.5
projection	61.0	58.4	52.8
DAN	60.9	60.1	52.3
RNN	60.5	60.3	51.8
iRNN	60.3	60.0	**53.9**
LSTM (no o.g.)	**61.6**	**61.3**	53.4
LSTM (o.g.)	61.5	60.9	52.9
skip-thought	39.3	39.3	31.9
GloVe	44.8	44.8	25.3
PARAGRAM-SL999	55.3	55.3	40.4

Table 15: Results on the PPDB tasks (Spearman's $\rho \times 100$). For the task in (Pavlick et al., 2015), we include the oracle result (the max Spearman's ρ on the dataset), since this dataset was used for model selection for all other tasks, as well as test results where models were tuned on Annotated-PPDB.

4.4.3 PARAGRAM-PHRASE *XXL*

Since we found that PARAGRAM-PHRASE embeddings have such strong performance, we trained this model on more data from PPDB and also used more data for hyperparameter tuning. For tuning, we used all of PPDB XL and trained for 10 epochs, then trained our final model for 10 epochs on the entire phrase section of PPDB XXL, consisting of 9,123,575 unique phrase pairs.[11] We show the results of this improved model, which we call PARAGRAM-PHRASE XXL, in Table 16. We also report the median, 75th percentile, and maximum score from our suite of textual similarity tasks. PARAGRAM-PHRASE XXL matches or exceeds the best performance on 4 of the datasets (SMT-news, SMT, deft forum, and belief) and is within 3 points of the best performance on 8 out of 22. We have made this trained model available to the research community.[12]

4.4.4 *Using Representations in Learned Models*

We explore two natural questions regarding our representations learned from PPDB: (1) can these embeddings improve the performance of other models through initialization and regularization? (2) can they effectively be used as features for downstream tasks? To address these questions, we used three tasks: The SICK similarity task, the SICK entailment task,

11 We fixed batchsize to 100 and δ to 0.4, as these were the optimal values for the experiment in Table 14. Then, for λ_w we searched over $\{10^{-6}, 10^{-7}, 10^{-8}\}$, and tuned over MIX and MAX sampling. To optimize, we used AdaGrad with a learning rate of 0.05.

12 Available at `https://www.cs.cmu.edu/~jwieting`.

Dataset	50%	75%	Max	paragram-phrase-XXL
MSRpar	51.5	57.6	73.4	44.8
MSRvid	75.5	80.3	88.0	79.6
SMT-eur	44.4	48.1	56.7	49.5
OnWN	60.8	65.9	72.7	70.4
SMT-news	40.1	45.4	60.9	**63.3**
STS 2012 Average	54.5	59.5	70.3	61.5
headline	64.0	68.3	78.4	73.9
OnWN	52.8	64.8	84.3	73.8
FNWN	32.7	38.1	58.2	47.7
SMT	31.8	34.6	40.4	**40.4**
STS 2013 Average	45.3	51.4	65.3	58.9
deft forum	36.6	46.8	53.1	**53.4**
deft news	66.2	74.0	78.5	74.4
headline	67.1	75.4	78.4	71.5
images	75.6	79.0	83.4	80.4
OnWN	78.0	81.1	87.5	81.5
tweet news	64.7	72.2	79.2	77.4
STS 2014 Average	64.7	71.4	76.7	73.1
answers-forums	61.3	68.2	73.9	69.1
answers-students	67.6	73.6	78.8	78.0
belief	67.7	72.2	77.2	**78.2**
headline	74.2	80.8	84.2	76.4
images	80.4	84.3	87.1	83.4
STS 2015 Average	70.2	75.8	80.2	77.0
2014 SICK*	71.4	79.9	82.8	72.7
2015 Twitter	49.9	52.5	61.9	52.4

Table 16: Results on SemEval textual similarity datasets (Pearson's $r \times 100$) for paragram-phrase XXL embeddings. Results that match or exceed the best shared task system are shown in bold. *For the 2014 SICK task, the median, 75th percentile, and maximum include only the primary runs as the full set of results was not available.

and the Stanford Sentiment Treebank (SST) binary classification task (Socher et al., 2013). For the SICK similarity task, we minimize the objective function[13] from (Tai et al., 2015).

13 This objective function has been shown to perform very strongly on text similarity tasks, significantly better than squared or absolute error.

Given a score for a sentence pair in the range $[1, K]$, where K is an integer, with sentence representations h_L and h_R, and model parameters θ, they first compute:

$$\begin{aligned}
h_\times &= h_L \odot h_R, \;\; h_+ = |h_L - h_R|, \\
h_s &= \sigma\left(W^{(\times)} h_\times + W^{(+)} h_+ + b^{(h)}\right), \\
\hat{p}_\theta &= \text{softmax}\left(W^{(p)} h_s + b^{(p)}\right), \\
\hat{y} &= r^T \hat{p}_\theta,
\end{aligned}$$

where $r^T = [1\ 2\ \ldots\ K]$. They then define a sparse target distribution p that satisfies $y = r^T p$:

$$p_i = \begin{cases} y - \lfloor y \rfloor, & i = \lfloor y \rfloor + 1 \\ \lfloor y \rfloor - y + 1, & i = \lfloor y \rfloor \\ 0 & \text{otherwise} \end{cases}$$

for $1 \leqslant i \leqslant K$. Then they use the following loss, the regularized KL-divergence between p and $\hat{p}_\theta$:

$$J(\theta) = \frac{1}{m} \sum_{k=1}^{m} \text{KL}\left(p^{(k)} \,\middle\|\, \hat{p}_\theta^{(k)}\right),$$

where m is the number of training pairs and where we always use L_2 regularization on all compositional parameters[14] but omit these terms for clarity.

We use nearly the same model for the entailment task, with the only differences being that the final softmax layer has three outputs and the cost function is the negative log-likelihood of the class labels. For sentiment, since it is a binary sentence classification task, we first encoded the sentence and then used a fully-connected layer with a sigmoid activation followed by a softmax layer with two outputs. We used negative log-likelihood of the class labels as the cost function. All models use L_2 regularization on all parameters, except for the word embeddings, which are regularized back to their initial values with an L_2 penalty.

We first investigated how these models performed in the standard setting, without using any models trained using PPDB data. We tuned hyperparameters on the development set of each dataset[15] as well as on two optimization schemes: AdaGrad with learning rate of

14 Word embeddings are regularized toward their initial state.

15 For all models, we tuned batch-size over $\{25, 50, 100\}$, output dimension over $\{50, 150, 300\}$, λ_c over $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$, $\lambda_s = \lambda_c$, and λ_w over $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$ as well as the option of not updating the embeddings for all models except the word averaging model. We again fix the output di-

Task	word averaging	proj.	DAN	RNN	LSTM (no o.g.)	LSTM (o.g.)	w/ *universal* regularization
similarity (SICK)	**86.40**	85.93	85.96	73.13	85.45	83.41	**86.84**
entailment (SICK)	**84.6**	84.0	84.5	76.4	83.2	82.0	**85.3**
binary sentiment (SST)	83.0	83.0	83.4	86.5	86.6	**89.2**	86.9

Table 17: Results from supervised training of each compositional architecture on similarity, entailment, and sentiment tasks. The last column shows results regularizing to our *universal* parameters from the models in Table 14. The first row shows Pearson's $r \times 100$ and the last two show accuracy.

0.05 and Adam with a learning rate of 0.001. We trained the models for 10 epochs and initialized the word embeddings with PARAGRAM-SL999 embeddings.

The results are shown in Table 17. We find that using word averaging as the compositional architecture outperforms the other architectures for similarity and entailment. However, for sentiment classification, the LSTM is much stronger than the averaging models. This suggests that the superiority of a compositional architecture can vary widely depending on the evaluation, and motivates future work to compare these architectures on additional tasks.

These results are very competitive with the state of the art on these tasks. Recent strong results on the SICK similarity task include 86.86 using a convolutional neural network (He et al., 2015) and 86.76 using a tree-LSTM (Tai et al., 2015). For entailment, the best result we are aware of is 85.1 (Beltagy et al., 2015). On sentiment, the best previous result is 88.1 (Kim, 2014), which our LSTM surprisingly outperforms by a significant margin. We note that these experiments simply compare compositional architectures using only the provided training data for each task, tuning on the respective development sets. We did not use any PPDB data for these results, other than that used to train the initial PARAGRAM-SL999 embeddings. Our results appear to show that standard neural architectures can perform surprisingly well given strong word embeddings and thorough tuning over the hyperparameter space.

4.4.4.1 *Regularization and Initialization to Improve Textual Similarity Models*

In this setting, we initialize each respective model to the parameters learned from PPDB (calling them *universal* parameters) and augment Eq. 4.4.4 with three separate regularization terms with the following weights: λ_s which regularizes the classification parameters (the two layers used in the classification step after obtaining representations), λ_w for regu-

mensionalities of all models which require this specification, to the dimensionality of our word embeddings (300). Additionally, for the classic RNN, we further tuned whether to use tanh or rectified linear unit activation functions; for the DANs we tuned over activation function (tanh or rectified linear unit) and the number of layers (either 1 or 2).

larizing the word parameters toward the learned W_w from PPDB, and λ_c for regularizing the compositional parameters (for all models except for the word averaging model) back to their initial values.[16] In all cases, we regularize to the universal parameters using L_2 regularization.

The results are shown in the last column of Table 17, and we only show results for the best performing models on each task (word averaging for similarity/entailment, LSTM with output gate for sentiment). Interestingly, it seems that regularizing to our universal parameters significantly improves results for the similarity and entailment tasks which are competitive or better than the state-of-the-art, but harms the LSTM's performance on the sentiment classification task.

4.4.4.2 *Representations as Features*

Task	PARAGRAM-PHRASE			skip-thought	
	300	1200	2400	uni-skip	bi-skip
similarity (SICK)	82.15	82.85	**84.94**	84.77	84.05
entailment (SICK)	80.2	80.1	**83.1**	-	-
binary sentiment (SST)	**79.7**	78.8	79.4	-	-

Table 18: Results from supervised training on similarity, entailment, and sentiment tasks, except that we keep the sentence representations fixed to our PARAGRAM-PHRASE model. The first row shows Pearson's $r \times 100$ and the last two show accuracy, with boldface showing the highest score in each row.

We also investigate how our PARAGRAM-PHRASE embeddings perform as features for supervised tasks. We use a similar set-up as in (Kiros et al., 2015) and encode the sentences by averaging our PARAGRAM-PHRASE embeddings and then just learn the classification parameters without updating the embeddings. To provide a more apt comparison to skip-thought vectors, we also learned a linear projection matrix to increase dimensionality of our PARAGRAM-PHRASE embeddings. We chose 1200 and 2400 dimensions in order to both see the dependence of dimension on performance, and so that they can be compared fairly with skip-thought vectors. Note that 2400 dimensions is the same dimensionality as the uni-skip and bi-skip models in (Kiros et al., 2015).

The 300 dimension case corresponds to the PARAGRAM-PHRASE embeddings from Table 14. We tuned our higher dimensional models on PPDB as described previously in Section 4.4.2.2 before training on PPDB XL.[17] Then we trained the same models for the

16 We tuned λ_s over $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$, λ_c over $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$, and λ_w over $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$. All other hyperparameters were tuned as previously described.

17 Note that we fixed batch-size to 100, δ to 0.4, and used MAX sampling as these were the optimal parameters for the PARAGRAM-PHRASE embeddings. We tuned the other hyperparameters as described in Section 4.4.2.2 with the exception of λ_c which was tuned over $\{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$.

similarity, entailment, and sentiment tasks as described in Section 4.4.4 for 20 epochs. We again tuned λ_s over $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ and tuned over the two optimization schemes of AdaGrad with learning rate of 0.05 and Adam with a learning rate of 0.001. Note that we are not updating the word embeddings or the projection matrix during training.

The results are shown in Table 18. The similarity and entailment tasks show clear improvements as we project the embeddings into the 2400 dimensional space. In fact, our results outperform both types of skip-thought embeddings on the single task that we overlap. However, the sentiment task does not benefit from higher dimensional representations, which is consistent with our regularization experiments in which sentiment also did not show improvement. Therefore, it seems that our models learned from PPDB are more effective for *similarity* tasks than *classification* tasks, but this hypothesis requires further investigation.

4.5 DISCUSSION

It is interesting that the LSTM, with or without output gates, is outperformed by much simpler models on the similarity and entailment tasks studied in this paper. We now consider possible explanations for this trend.

The first hypothesis we test is based on length. Since PPDB contains short text snippets of a few words, the LSTM may not know how to handle the longer sentences that occur in our evaluation tasks. If this is true, the LSTM would perform much better on short text snippets and its performance would degrade as their length increases. To test this hypothesis, we took all 12,108 pairs from the 20 SemEval STS tasks and binned them by length.[18] We then computed the Pearson's r for each bin. The results are shown in Table 19 and show that while the LSTM models do perform better on the shortest text pairs, they are still outperformed, at all lengths, by the PARAGRAM-PHRASE model.[19]

We next consider whether the LSTM has worse generalization due to overfitting on the training data. To test this, we analyzed how the models performed on the training data (PPDB XL) by computing the average difference between the cosine similarity of the gold phrase pairs and the negative examples.[20] We found that all models had very similar scores:

18 For each pair, we computed the number of tokens in each of the two pieces of text, took the max, and then binned based on this value.

19 Note that for the analysis in Sections 4.5 and 4.6, the models used were selected from earlier experiments. They are not the same as those used to obtain the results in Table 14.

20 More precisely, for each gold pair $\langle g_1, g_2 \rangle$, and n_i, the respective negative example of each g_i, we computed $2 \cdot \cos(g_1, g_2) - \cos(n_1, g_1) - \cos(n_2, g_2)$ and averaged this value over all pairs.

Max Length	PARAGRAM-PHRASE	LSTM (no o.g.)	LSTM (o.g.)	PARAGRAM-SL999
$\leqslant 4$	**72.7**	63.4	58.8	66.3
5	**74.0**	54.5	48.4	65.0
6	**70.5**	52.6	48.2	50.1
7	**73.7**	56.9	50.6	56.4
8	**75.5**	60.2	52.4	60.1
9	**73.0**	58.0	48.8	58.8
$\geqslant 10$	**72.6**	55.6	53.8	58.4

Table 19: Performance (Pearson's $r \times 100$) as a function of the maximum number of tokens in the sentence pairs over all 20 SemEval STS datasets.

0.7535, 0.7572, 0.7565, and 0.7463 for PARAGRAM-PHRASE, projection, LSTM (o.g.), and LSTM (no o.g.). This, along with the similar performance of the models on the PPDB tasks in Table 15, suggests that overfitting is not the cause of the worse performance of the LSTM model.

Lastly, we consider whether the LSTM's weak performance was a result of insufficient tuning or optimization. We first note that we actually ran more hyperparameter tuning experiments for the LSTM models than either the PARAGRAM-PHRASE or projection models, since we tuned the decision to use an output gate. Secondly, we note that (Tai et al., 2015) had a similar LSTM result on the SICK dataset (Pearson's r of 85.28 to our 85.45) to show that our LSTM implementation/tuning procedure is able to match or exceed performance of another published LSTM result. Thirdly, the similar performance across models on the PPDB tasks (Table 15) suggests that no model had a large advantage during tuning; all found hyperparameters that comfortably beat the PARAGRAM-SL999 addition baseline. Finally, we point out that we tuned over learning rate and optimization strategy, as well as experimented with clipping gradients, in order to rule out optimization issues.

4.5.1 *Under-Trained Embeddings*

One limitation of our new PARAGRAM-PHRASE vectors is that many of our embeddings are under-trained. The number of unique tokens occurring in our training data, PPDB XL, is 37,366. However, the number of tokens appearing more than 100 times is just 7,113. Thus, one clear source of improvement for our model would be to address under-trained embeddings for tokens appearing in our test data.

In order to gauge the effect under-trained embeddings and unknown words have on our model, we calculated the fraction of words in each of our 22 SemEval datasets that do not occur at least 100 times in PPDB XL along with our performance deviation from the 75th percentile of each dataset. We found that this fraction had a Spearman's ρ of -45.1 with the deviation from the 75th percentile indicating that there is a significant negative correlation between the fraction of OOV words and performance on these STS tasks.

4.5.2 *Using More PPDB*

4.5.2.1 *Performance Versus Amount of Training Data*

Models in related work such as (Kiros et al., 2015) and (Li et al., 2015) require significant training time on GPUs, on the order of multiple weeks. Moreover, dependence of model performance upon training data size is unclear. To investigate this dependence for our PARAGRAM-PHRASE model, we trained on different amounts of data and plotted the performance. The results are shown in Figure 1. We start with PPDB XL which has 3,033,753 unique phrase pairs and then divide by two until there are fewer than 10 phrase pairs.[21] For each data point (each division by two), we trained a model with that number of phrase pairs for 10 epochs. We use the average Pearson correlation for all 22 datasets in Table 14 as the dependent variable in our plot.

We experimented with two different ways of selecting training data. The first ("Ordered") retains the order of the phrase pairs in PPDB, which ensures the smaller datasets contain higher confidence phrase pairs. The second ("Random") randomly permutes PPDB XL before constructing the smaller datasets. In both methods, each larger dataset contains the previous one plus as many new phase pairs.

We make three observations about the plot in Figure 1. The first is that performance continually increases as more training data is added. This is encouraging as our embeddings can continually improve with more data. Secondly, we note the sizable improvement (4 points) over the PARAGRAM-SL999 baseline by training on just 92 phrase pairs from PPDB. Finally, we note the difference between randomly permuting the training data and using the order from PPDB (which reflects the confidence that the phrases in each pair possess the paraphrase relationship). Performance of the randomly permuted data is usually slightly better than that of the ordered data, until the performance gap vanishes once half of PPDB

21 The smallest dataset contained 5 pairs.

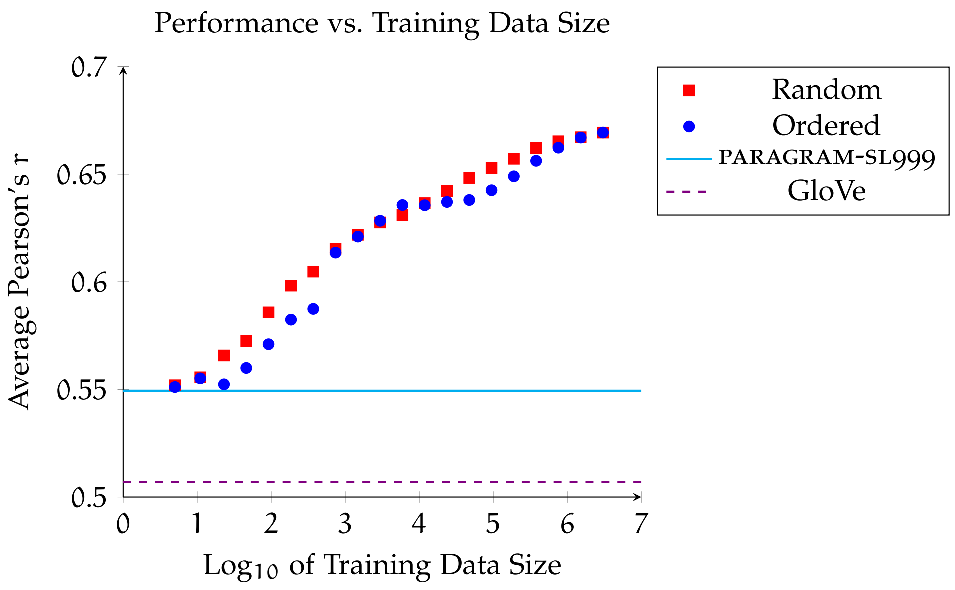


Figure 1: Performance of the PARAGRAM-PHRASE embeddings as measured by the average Pearson's r on 22 textual similarity datasets versus the amount of training data from PPDB on a log scale. Each datapoint contains twice as much training data as the previous one. Random and Ordered refer to whether we shuffled the XL paraphrase pairs from PPDB or kept them in order. We also show baselines of averaging PARAGRAM-SL999 and GloVe embeddings.

XL is used. We suspect this behavior is due to the *safe* phrase pairs that occur in the beginning of PPDB. These high-confidence phrase pairs usually have only slight differences and therefore are not as useful for training our model.

4.6 QUALITATIVE ANALYSIS

Word	PARAGRAM-PHRASE Nearest Neighbors	PARAGRAM-SL999 Nearest Neighbors
unlike	contrary, contrast, opposite, versa, conversely, opposed, contradiction	than, although, whilst, though, albeit, kinda, alike
2	2.0, two, both, ii, 2nd, couple, 02	2.0, 3, 1, b, ii, two, 2nd
ladies	girls, daughters, honorable, females, girl, female, dear	gentlemen, colleague, fellow, girls, mr, madam, dear
lookin	staring, looking, watching, look, searching, iooking, seeking	doin, goin, talkin, sayin, comin, outta, somethin
disagree	agree, concur, agreeing, differ, accept	disagreement, differ, dispute, difference, disagreements

Table 20: Nearest neighbors of PARAGRAM-PHRASE and PARAGRAM-SL999 word embeddings sorted by cosine similarity.

To explore other differences between our PARAGRAM-PHRASE vectors and the PARAGRAM-SL999 vectors that were used for initialization, we inspected lists of nearest neighbors in each vector space. When obtaining nearest neighbors, we restricted our search to the 10,000 most common tokens in PPDB XL to ensure that the PARAGRAM-PHRASE vectors were not too under-trained. Some informative neighbors are shown in Table 20. In the first four rows, we see that the PARAGRAM-PHRASE embeddings have neighbors with a strong paraphrasing relationship. They tend to avoid having neighbors that are antonyms or co-hyponyms such as *unlike* and *alike* or *2* and *3* which are an issue for the PARAGRAM-SL999 embeddings. In contrast to the first four rows, the last row shows a problematic effect of our bag-of-words composition function: *agree* is the nearest neighbor of *disagree*. The reason for this is that there are numerous pairs in PPDB XL such as *i disagree* and *i do not agree* that encourage *disagree* and *agree* to have high cosine similarity. A model that takes context into account could resolve this issue. The difficulty would be finding a model that does so while still generalizing well, as we found that our PARAGRAM-PHRASE embeddings generalize better than learning a weight matrix or using a recurrent neural network. We leave this for future work.

When we take a closer look at our PARAGRAM-PHRASE embeddings, we find that information-bearing content words, such as *poverty*, *kidding*, *humanitarian*, *18*, and *july* have the largest L_2 norms, while words such as *of*, *it*, *to*, *hereby* and *the* have the smallest. Pham et al. (2015) noted this same phenomenon in their closely-related compositional model. Interestingly, we found that this weighting explains much of the success of our model. In order to quantify exactly how much, we calculated a weight for each token in our working vocabulary[22] simply by summing up the absolute value of all components of its PARAGRAM-PHRASE vector. Then we multiplied each weight by its corresponding PARAGRAM-SL999 word vector. We computed the average Pearson's r over all 22 datasets in Table 14. The PARAGRAM-SL999 vectors have an average correlation of 54.94, the PARAGRAM-PHRASE vectors have 66.83, and the scaled PARAGRAM-SL999 vectors, where each is multiplied by its computed weight, have an average Pearson's r of 62.64. Therefore, it can be surmised that at least 64.76% of the improvement over the initial PARAGRAM-SL999 vectors is due to weighting tokens by their importance.[23]

We also investigated the connection between these multiplicative weights and word frequency. To do so, we calculated the frequency of all tokens in PPDB XL.[24] We then normalized these by the total number of tokens in PPDB XL and used the reciprocal of these scores as the multiplicative weights. Thus less frequent words have more weight than more frequent words. With this baseline weighting method, the average Pearson's r is 45.52, indi-

22 This corresponds to the 42,091 tokens that appear in the intersection of our PARAGRAM-SL999 vocabulary, the test sets of all STS tasks in our evaluation, and PPDB XL plus an unknown word token.

23 We also trained a model in which we only a learn a single multiplicative parameter for each word in our vocabulary, keeping the word embeddings fixed to the PARAGRAM-SL999 embeddings. We trained for 10 epochs on all phrase pairs in PPDB XL. The resulting average Pearson's r, after tuning on the Pavlick et al. PPDB task, was 62.06, which is slightly lower than using the absolute value of each PARAGRAM-PHRASE vector as its multiplicative weight.

24 Tokens that did not appear in PPDB XL were assigned a frequency of 1.

cating that the weights we obtain for these words are more sophisticated than mere word frequency. These weights are potentially useful for other applications that can benefit from modeling word importance, such as information retrieval.

4.7 CONCLUSION

We introduced an approach to create universal sentence embeddings and propose our model as the new baseline for embedding sentences, as it is simple, efficient, and performs strongly across a broad range of tasks and domains. Moreover, our representations do not require the use of any neural network architecture. The embeddings can be simply averaged for a given sentence in an NLP application to create its sentence embedding. We also find that our representations can improve general text similarity and entailment models when used as a prior and can achieve strong performance even when used as fixed representations in a classifier. Future work will focus on improving our embeddings by effectively handling undertrained words as well as by exploring new models that generalize even better to the large suite of text similarity tasks used in our experiments.

5 CHARAGRAM: IMPROVED GENERALIZATION BY EMBEDDING WORDS AND SENTENCES VIA CHARACTER nGRAMS

5.1 INTRODUCTION

Representing textual sequences such as words and sentences is a fundamental component of natural language understanding systems. Many functional architectures have been proposed to model compositionality in word sequences, ranging from simple averaging (Mitchell and Lapata, 2010; Iyyer et al., 2015) to functions with rich recursive structure (Socher et al., 2011; Tai et al., 2015; Bowman et al., 2016). Most work uses words as the smallest units in the compositional architecture, often using pretrained word embeddings or learning them specifically for the task of interest (Tai et al., 2015; He et al., 2015).

Some prior work has found benefit from using character-based compositional models that encode arbitrary character sequences into vectors. Examples include recurrent neural networks (RNNs) and convolutional neural networks (CNNs) on character sequences, showing improvements for several NLP tasks (Ling et al., 2015a; Kim et al., 2015; Ballesteros et al., 2015; dos Santos and Guimarães, 2015). By sharing subword information across words, character models have the potential to better represent rare words and morphological variants.

Our approach, CHARAGRAM, uses a much simpler functional architecture. We represent a character sequence by a vector containing counts of character n-grams, inspired by Huang et al. (2013). This vector is embedded into a low-dimensional space using a single nonlinear transformation. This can be interpreted as learning embeddings of character n-grams, which are learned so as to produce effective sequence embeddings when a summation is performed over the character n-grams in the sequence.

We consider three evaluations: word similarity, sentence similarity, and part-of-speech tagging. On multiple word similarity datasets, CHARAGRAM outperforms RNNs and CNNs, achieving state-of-the-art performance on SimLex-999 (Hill et al., 2015). When evaluated on a large suite of sentence-level semantic textual similarity tasks, CHARAGRAM embeddings again outperform the RNN and CNN architectures as well as the PARAGRAM-PHRASE embeddings of Chapter 4. We also consider English part-of-speech (POS) tagging using the bidirectional long short-term memory tagger of Ling et al. (2015a). The three architectures reach similar performance, though CHARAGRAM converges fastest to high accuracy.

We perform extensive analysis of our CHARAGRAM embeddings. We find large gains in performance on rare words, showing the empirical benefit of subword modeling. We also compare performance across different character n-gram vocabulary sizes, finding that the semantic tasks benefit far more from large vocabularies than the syntactic task. However, even for challenging semantic similarity tasks, we still see strong performance with only a few thousand character n-grams.

Nearest neighbors show that CHARAGRAM embeddings simultaneously address differences due to spelling variation, morphology, and word choice. Inspection of embeddings of particular character n-grams reveals etymological links; e.g., *die* is close to *mort*. We release our resources to the community in the hope that CHARAGRAM can provide a strong baseline for subword-aware text representation.

5.2 RELATED WORK

We first review work on using subword information in word embedding models. The simplest approaches append subword features to word embeddings, letting the model learn how to use the subword information for particular tasks. Some added knowledge-based morphological features to word representations (Alexandrescu and Kirchhoff, 2006; El-Desoky Mousa et al., 2013). Others learned embeddings jointly for subword units and words, defining simple compositional architectures (often based on addition) to create word embeddings from subword embeddings (Lazaridou et al., 2013; Botha and Blunsom, 2014; Qiu et al., 2014; Chen et al., 2015c).

A recent trend is to use richer functional architectures to convert character sequences into word embeddings. Luong et al. (2013) used recursive models to compose morphs into word embeddings, using unsupervised morphological analysis. Ling et al. (2015a) used a bidirectional long short-term memory (LSTM) RNN on characters to embed arbitrary word types, showing strong performance for language modeling and POS tagging. Ballesteros et al. (2015) used this model to represent words for dependency parsing. Several have used character-level RNN architectures for machine translation, whether for representing source or target words (Ling et al., 2015b; Luong and Manning, 2016), or for generating entire translations character-by-character (Chung et al., 2016).

Sutskever et al. (2011) and Graves (2013) used character-level RNNs for language modeling. Others trained character-level RNN language models to provide features for NLP tasks, including tokenization and segmentation (Chrupała, 2013; Evang et al., 2013), and text normalization (Chrupała, 2014).

CNNs with character n-gram filters have been used to embed arbitrary word types for several tasks, including language modeling (Kim et al., 2015), part-of-speech tagging (dos Santos and Zadrozny, 2014), named entity recognition (dos Santos and Guimarães, 2015), text classification (Zhang et al., 2015), and machine translation (Costa-Jussà and Fonollosa, 2016). Combinations of CNNs and RNNs on characters have also been explored (Józefowicz et al., 2016).

Most closely-related to our approach is the DSSM (instantiated variously as "deep semantic similarity model" or "deep structured semantic model") developed by Huang et al. (2013). For an information retrieval task, they represented words using feature vectors containing counts of character n-grams. Sperr et al. (2013) used a very similar technique to represent words in neural language models for machine translation. Our CHARAGRAM embeddings are based on this same idea. We show this strategy to be extremely effective when applied to both words and sentences, outperforming character LSTMs like those used by Ling et al. (2015a) and character CNNs like those from Kim et al. (2015).

5.3 MODELS

We now describe models that embed textual sequences using their characters, including our CHARAGRAM model and the baselines that we compare to. We denote a character-based textual sequence by $x = \langle x_1, x_2, ..., x_m \rangle$, which includes space characters between words as well as special start-of-sequence and end-of-sequence characters. We use x_i^j to denote the subsequence of characters from position i to position j inclusive, i.e., $x_i^j = \langle x_i, x_{i+1}, ..., x_j \rangle$, and we define $x_i^i = x_i$.

Our CHARAGRAM model embeds a character sequence x by adding the vectors of its character n-grams followed by an elementwise nonlinearity:

$$g_{\text{CHAR}}(x) = h\left(\mathbf{b} + \sum_{i=1}^{m+1} \sum_{j=1+i-k}^{i} \mathbb{I}\left[x_j^i \in V\right] W^{x_j^i}\right)$$

where h is a nonlinear function, $\mathbf{b} \in \mathbb{R}^d$ is a bias vector, k is the maximum length of any character n-gram, $\mathbb{I}[p]$ is an indicator function that returns 1 if p is true and 0 otherwise,

V is the set of character n-grams included in the model, and $W^{x_j^i} \in \mathbb{R}^d$ is the vector for character n-gram x_j^i.

The set V is used to restrict the model to a predetermined set (vocabulary) of character n-grams. Below, we compare several choices for defining this set. The number of parameters in the model is $d + d|V|$. This model is based on the letter n-gram hashing technique developed by Huang et al. (2013) for their DSSM approach. One can also view Eq. equation 5.3 (as they did) as first populating a vector of length $|V|$ with counts of character n-grams followed by a nonlinear transformation.

We compare the CHARAGRAM model to two other models. First we consider LSTM architectures (Hochreiter and Schmidhuber, 1997) over the character sequence x, using the version from Gers et al. (2003). We use a forward LSTM over the characters in x, then take the final LSTM hidden vector as the representation of x. Below we refer to this model as "charLSTM."

We also compare to convolutional neural network (CNN) architectures, which we refer to below as "charCNN." We use the architecture from Kim (2014) with a single convolutional layer followed by an optional fully-connected layer. We use filters of varying lengths of character n-grams, using two primary configurations of filter sets, one of which is identical to that used by Kim et al. (2015). Each filter operates over the entire sequence of character n-grams in x and we use max pooling for each filter. We tune over the choice of nonlinearity for both the convolutional filters and for the optional fully-connected layer. We give more details below about filter sets, n-gram lengths, and nonlinearities.

We note that using character n-gram convolutional filters is similar to our use of character n-grams in the CHARAGRAM model. The difference is that, in the CHARAGRAM model, the n-gram must match exactly for its vector to affect the representation, while in the CNN each filter will affect the representation of all sequences (depending on the nonlinearity being used). So the CHARAGRAM model is able to learn precise vectors for particular character n-grams with specific meanings, while there is pressure for the CNN filters to capture multiple similar patterns that recur in the data. Our qualitative analysis shows the specificity of the learned character n-gram vectors learned by the CHARAGRAM model.

5.4 EXPERIMENTS

We perform three sets of experiments. The goal of the first two (Section 5.4.1) is to produce embeddings for textual sequences such that the embeddings for paraphrases have high cosine similarity. Our third evaluation (Section 5.4.3) is a classification task, and follows the setup of the English part-of-speech tagging experiment from Ling et al. (2015a).

5.4.1 *Word and Sentence Similarity*

We compare the ability of our models to capture semantic similarity for both words and sentences. We train on noisy paraphrase pairs from the Paraphrase Database (PPDB; Ganitkevitch et al., 2013) with an L_2 regularized contrastive loss objective function, following the training procedure of Chapters 3 and 4 described below. For part-of-speech tagging, we follow the English Penn Treebank training procedure of Ling et al. (2015a).

For the similarity tasks, the training data consists of a set X of phrase pairs $\langle x_1, x_2 \rangle$ from the Paraphrase Database (PPDB; Ganitkevitch et al., 2013), where x_1 and x_2 are assumed to be paraphrases. We optimize a margin-based loss:

$$\min_{\theta} \frac{1}{|X|} \Bigg(\sum_{\langle x_1, x_2 \rangle \in X} \max(0, \delta - \cos(g(x_1), g(x_2)) \\ + \cos(g(x_1), g(t_1))) + \max(0, \delta - \cos(g(x_1), g(x_2)) \\ + \cos(g(x_2), g(t_2))) \Bigg) + \lambda \, \|\theta\|^2$$

where g is the embedding function in use, δ is the margin, the full set of parameters is contained in θ (e.g., for the CHARAGRAM model, $\theta = \langle W, \mathbf{b} \rangle$), λ is the L_2 regularization coefficient, and t_1 and t_2 are carefully selected negative examples taken from a mini-batch during optimization (discussed below). Intuitively, we want the two phrases to be more similar to each other ($\cos(g(x_1), g(x_2))$) than either is to their respective negative examples t_1 and t_2, by a margin of at least δ.

5.4.2 *Selecting Negative Examples*

To select t_1 and t_2 in Eq. equation 5.4.1, we tune the choice between two approaches. The first, MAX, simply chooses the most similar phrase in some set of phrases (other than those in the given phrase pair). For simplicity and to reduce the number of tunable parameters, we use the mini-batch for this set, but it could be a separate set. Formally, MAX corresponds to choosing t_1 for a given $\langle x_1, x_2 \rangle$ as follows:

$$t_1 = \operatorname*{argmax}_{t:\langle t,\cdot\rangle \in X_b \setminus \{\langle x_1, x_2 \rangle\}} \cos(g(x_1), g(t))$$

where $X_b \subseteq X$ is the current mini-batch. That is, we want to choose a negative example t_i that is similar to x_i according to the current model parameters. The downside of this approach is that we may occasionally choose a phrase t_i that is actually a true paraphrase of x_i.

The second strategy selects negative examples using MAX with probability 0.5 and selects them randomly from the mini-batch otherwise. We call this sampling strategy MIX. We tune over the choice of strategy in our experiments.

5.4.2.1 *Datasets*

For word similarity, we focus on two of the most commonly used datasets for evaluating semantic similarity of word embeddings: WordSim-353 (WS353) (Finkelstein et al., 2001) and SimLex-999 (SL999) (Hill et al., 2015). We also evaluate our best model on the Stanford Rare Word Similarity Dataset (Luong et al., 2013).

For sentence similarity, we evaluate on a diverse set of 22 textual similarity datasets, including all datasets from every SemEval semantic textual similarity (STS) task from 2012 to 2015. We also evaluate on the SemEval 2015 Twitter task (Xu et al., 2015b) and the SemEval 2014 SICK Semantic Relatedness task (Marelli et al., 2014). Given two sentences, the aim of the STS tasks is to predict their similarity on a 0-5 scale, where 0 indicates the sentences are on different topics and 5 indicates that they are completely equivalent.

Each STS task consists of 4-6 datasets covering a wide variety of domains, including newswire, tweets, glosses, machine translation outputs, web forums, news headlines, image and video captions, among others. Most submissions for these tasks use supervised models that are trained and tuned on provided training data or similar datasets from older tasks. Further details are provided in the official task descriptions (Agirre et al., 2012, 2013, 2014, 2015).

5.4.2.2 *Preliminaries*

For training data, we use pairs from PPDB. For word similarity experiments, we train on word pairs and for sentence similarity, we train on phrase pairs. PPDB comes in different sizes (S, M, L, XL, XXL, and XXXL), where each larger size subsumes all smaller ones. The pairs in PPDB are sorted by a confidence measure and so the smaller sets contain higher precision paraphrases.

Before training the CHARAGRAM model, we need to populate V, the vocabulary of character n-grams included in the model. We obtain these from the training data used for the final models in each setting, which is either the lexical or phrasal section of PPDB XXL. We tune over whether to include the full sets of character n-grams in these datasets or only those that appear more than once.

When extracting n-grams, we include spaces and add an extra space before and after each word or phrase in the training and evaluation data to ensure that the beginning and end of each word is represented. We note that strong performance can be obtained using far fewer character n-grams; we explore the effects of varying the number of n-grams and the n-gram orders in Section 5.4.5.

We used Adam (Kingma and Ba, 2014) with a learning rate of 0.001 to learn the parameters in the following experiments.

5.4.2.3 *Word Embedding Experiments*

TRAINING AND TUNING For hyperparameter tuning, we used one epoch on the lexical section of PPDB XXL, which consists of 770,007 word pairs. We used either WS353 or SL999 for model selection (reported below). We then took the selected hyperparameters and trained for 50 epochs to ensure that all models had a chance to converge.

We tuned all models thoroughly, tuning the activation functions for CHARAGRAM and charCNN, as well as the regularization strength, mini-batch size, and sampling type for all models. For all architectures, we tuned over the mini-batch size (25 or 50) and the type of sampling used (MIX or MAX). δ was set to 0.4 and the dimensionality d of each model was set to 300.

For the CHARAGRAM model, we tuned the activation function h (tanh or linear) and regularization coefficient λ (over $\{10^{-4}, 10^{-5}, 10^{-6}\}$). The n-gram vocabulary V contained all 100,283 character n-grams ($n \in \{2, 3, 4\}$) in the lexical section of PPDB XXL.

For charCNN and charLSTM, we randomly initialized 300 dimensional character embeddings for all unique characters in the training data and we tuned λ over $\{10^{-4}, 10^{-5}, 10^{-6}\}$. For charLSTM, we tuned over whether to include an output gate. For charCNN, we tuned the filter activation function (rectified linear or tanh) and tuned the activation for the fully-connected layer (tanh or linear).

Model	Tuned on	WS353	SL999
charCNN	SL999	26.31	30.64
	WS353	33.19	16.73
charLSTM	SL999	48.27	54.54
	WS353	51.43	48.83
CHARAGRAM	SL999	53.87	**63.33**
	WS353	**58.35**	60.00
inter-annotator agreement	-	75.6	78

Table 21: Word similarity results (Spearman's $\rho \times 100$) on WS353 and SL999. The inter-annotator agreement is the average Spearman's ρ between a single annotator with the average over all other annotators.

For charCNN, we experimented with two filter sets: one uses 175 filters for each n-gram size $\in \{2, 3, 4\}$, and the other uses the set of filters from Kim et al. (2015), consisting of 25 filters of size 1, 50 of size 2, 75 of size 3, 100 of size 4, 125 of size 5, and 150 of size 6. We also experimented with using dropout (Srivastava et al., 2014) on the inputs of the last layer of the charCNN model in place of L_2 regularization, as well as removing the last feedforward layer. Neither of these variations significantly improved performance on our suite of tasks for word or sentence similarity. However, using more filters does improve performance, seemingly linearly with the square of the number of filters.

ARCHITECTURE COMPARISON The results are shown in Table 21. The CHARAGRAM model outperforms both the charLSTM and charCNN models, and also outperforms recent strong results on SL999.

We also found that the charCNN and charLSTM models take far more epochs to converge than the CHARAGRAM model. We noted this trend across experiments and explore it further in Section 5.4.4.

Model	SL999
Hill et al. (2014a)	52
Schwartz et al. (2015)	56
Faruqui and Dyer (2015)	58
Chapter 3	66.7
CHARAGRAM (large)	**70.6**

Table 22: Spearman's $\rho \times 100$ on SL999. CHARAGRAM (large) refers to the CHARAGRAM model described in Section 5.4.5. This model contains 173,881 character embeddings, more than the 100,283 in the CHARAGRAM model used to obtain the results in Table 21.

COMPARISON TO PRIOR WORK We found that performance of CHARAGRAM on word similarity tasks can be improved by using more character n-grams. This is explored in Section 5.4.5. Our best result from these experiments was obtained with the largest model

Dataset	50%	75%	Max	charCNN	charLSTM	PARAGRAM-PHRASE	CHARAGRAM-PHRASE
STS 2012 Average	54.5	59.5	70.3	56.5	40.1	58.5	**66.1**
STS 2013 Average	45.3	51.4	65.3	47.7	30.7	**57.7**	57.2
STS 2014 Average	64.7	71.4	76.7	64.7	46.8	71.5	**74.7**
STS 2015 Average	70.2	75.8	80.2	66.0	45.5	75.7	**76.1**
2014 SICK	71.4	79.9	82.8	62.9	50.3	**72.0**	70.0
2015 Twitter	49.9	52.5	61.9	48.6	39.9	52.7	**53.6**
Average	59.7	65.6	73.6	59.2	41.9	66.2	**68.7**

Table 23: Results on SemEval textual similarity datasets (Pearson's $r \times 100$). The highest score in each row is in boldface (omitting the official task score columns).

we considered, which contains 173,881 n-gram embeddings. When using WS353 for model selection and training for 25 epochs, this model achieves 70.6 on SL999. To our knowledge, this is the best result reported on SL999 in this setting; Table 22 shows comparable recent results. Note that a higher SL999 number is reported in (Mrkšić et al., 2016), but the setting is not comparable to ours as they started with embeddings tuned on SL999.

Lastly, we evaluated our model on the Stanford Rare Word Similarity Dataset (Luong et al., 2013), using SL999 for model selection. We obtained a Spearman's ρ of 47.1, which outperforms the 41.8 result from Soricut and Och (2015) and is competitive with the 47.8 reported in Pennington et al. (2014), despite only using PPDB for training.

5.4.2.4 *Sentence Embedding Experiments*

TRAINING AND TUNING We did initial training of our models using one pass through PPDB XL, which consists of 3,033,753 unique phrase pairs. Following our work in Chapter 4, we use the annotated phrase pairs developed by Pavlick et al. (2015) as our validation set, using Spearman's ρ to rank the models. We then take the highest performing models and train on the 9,123,575 unique phrase pairs in the phrasal section of PPDB XXL for 10 epochs.

For all experiments, we fix the mini-batch size to 100, the margin δ to 0.4, and use MAX sampling. For the CHARAGRAM model, V contains all 122,610 character n-grams ($n \in \{2,3,4\}$) in the PPDB XXL phrasal section. The other tuning settings are the same as in Section 5.4.2.3.

For another baseline, we train the PARAGRAM-PHRASE model of Chapter 4, tuning its regularization strength over $\{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$. The PARAGRAM-PHRASE model simply uses word averaging as its composition function, but outperforms many more complex models.

In this section, we refer to our model as CHARAGRAM-PHRASE because the input is a character sequence containing multiple words rather than only a single word as in Section 5.4.2.3.

Dataset	50%	75%	Max	charCNN	charLSTM	PARAGRAM-PHRASE	CHARAGRAM-PHRASE
MSRpar	51.5	57.6	73.4	50.6	23.6	42.9	**59.7**
MSRvid	75.5	80.3	88.0	72.2	47.2	76.1	**79.6**
SMT-eur	44.4	48.1	56.7	50.9	38.5	45.5	**57.2**
OnWN	60.8	65.9	72.7	61.8	53.0	**70.7**	68.7
SMT-news	40.1	45.4	60.9	46.8	38.3	57.2	**65.2**
STS 2012 Average	54.5	59.5	70.3	56.5	40.1	58.5	**66.1**
headline	64.0	68.3	78.4	68.1	54.4	72.3	**75.0**
OnWN	52.8	64.8	84.3	54.4	33.5	**70.5**	67.8
FNWN	32.7	38.1	58.2	26.4	10.6	**47.5**	42.3
SMT	31.8	34.6	40.4	42.0	24.2	40.3	**43.6**
STS 2013 Average	45.3	51.4	65.3	47.7	30.7	**57.7**	57.2
deft forum	36.6	46.8	53.1	45.6	19.4	50.2	**62.7**
deft news	66.2	74.0	78.5	73.5	54.6	73.2	**77.0**
headline	67.1	75.4	78.4	67.4	53.7	69.1	**74.3**
images	75.6	79.0	83.4	68.7	53.6	**80.0**	77.6
OnWN	78.0	81.1	87.5	66.8	46.1	**79.9**	77.0
tweet news	64.7	72.2	79.2	66.2	53.6	76.8	**79.1**
STS 2014 Average	64.7	71.4	76.7	64.7	46.8	71.5	**74.7**
answers-forums	61.3	68.2	73.9	47.2	27.3	**67.4**	61.5
answers-students	67.6	73.6	78.8	75.0	63.1	78.3	**78.5**
belief	67.7	72.2	77.2	65.7	22.6	76.0	**77.2**
headline	74.2	80.8	84.2	72.2	61.7	74.5	**78.7**
images	80.4	84.3	87.1	70.0	52.8	82.2	**84.4**
STS 2015 Average	70.2	75.8	80.2	66.0	45.5	75.7	**76.1**
2014 SICK	71.4	79.9	82.8	62.9	50.3	**72.0**	70.0
2015 Twitter	49.9	52.5	61.9	48.6	39.9	52.7	**53.6**
Average	59.7	65.6	73.6	59.2	41.9	66.2	**68.7**

Table 24: Results on SemEval textual similarity datasets (Pearson's $r \times 100$). The highest score in each row is in boldface (omitting the official task score columns).

Since the vocabulary V is defined by the training data sequences, the CHARAGRAM-PHRASE model includes character n-grams that span multiple words, permitting it to capture some aspects of word order and word co-occurrence, which the PARAGRAM-PHRASE model is unable to do.

We encountered difficulties training the charLSTM and charCNN models for this task. We tried several strategies to improve their chance at convergence, including clipping gradients, increasing training data, and experimenting with different optimizers and learning rates. We found success by using the original (confidence-based) ordering of the PPDB phrase pairs for the initial epoch of learning, then shuffling them for subsequent epochs. This is similar to curriculum learning (Bengio et al., 2009). The higher-confidence phrase pairs tend to be shorter and have many overlapping words, possibly making them easier to learn from.

RESULTS An abbreviated version of the sentence similarity results is shown in Table 23; Full results are shown in Table 24. For comparison, we report performance for the median (50%), third quartile (75%), and top-performing (Max) systems from the shared tasks. We observe strong performance for the CHARAGRAM-PHRASE model. It always does better than the charCNN and charLSTM models, and outperforms the PARAGRAM-PHRASE model on 15 of the 22 tasks. Furthermore, CHARAGRAM-PHRASE matches or exceeds the top-performing task-tuned systems on 5 tasks, and is within 0.003 on 2 more. The charLSTM and charCNN models are significantly worse, with the charCNN being the better of the two and beating PARAGRAM-PHRASE on 4 of the tasks.

We emphasize that there are many other models that could be compared to, such as an LSTM over word embeddings. This and many other models were explored in Chapter 4. Their PARAGRAM-PHRASE model, which simply learns word embeddings within an averaging composition function, was among their best-performing models. We used this model in our experiments as a strongly-performing representative of their results.

Lastly, we note other recent work that considers a similar transfer learning setting. The FastSent model (Hill et al., 2016) uses the 2014 STS task as part of its evaluation and reports an average Pearson's r of 61.3, much lower than the 74.7 achieved by CHARAGRAM-PHRASE on the same datasets.

Model	Accuracy (%)
charCNN	97.02
charLSTM	96.90
CHARAGRAM	96.99
CHARAGRAM (2-layer)	**97.10**

Table 25: Results on part-of-speech tagging.

5.4.3 *POS Tagging Experiments*

We now consider part-of-speech (POS) tagging, since it has been used as a testbed for evaluating architectures for character-level word representations. It also differs from semantic similarity, allowing us to evaluate our architectures on a syntactic task. We replicate the POS tagging experimental setup of Ling et al. (2015a). Their model uses a bidirectional LSTM over character embeddings to represent words. They then use the resulting word representations in another bidirectional LSTM that predicts the tag for each word. We replace their character bidirectional LSTM with our three architectures: charCNN, charLSTM, and CHARAGRAM.

We use the Wall Street Journal portion of the Penn Treebank, using Sections 1-18 for training, 19-21 for tuning, and 22-24 for testing. We set the dimensionality of the character embeddings to 50 and that of the (induced) word representations to 150. For optimization, we use stochastic gradient descent with a mini-batch size of 100 sentences. The learning rate and momentum are set to 0.2 and 0.95 respectively. We train the models for 50 epochs, again to ensure that all models have an opportunity to converge.

The other settings for our models are mostly the same as for the word and sentence experiments (Section 5.4.1). We again use character n-grams with $n \in \{2, 3, 4\}$, tuning over whether to include all 54,893 in the training data or only those that occur more than once. However, there are two minor differences from the previous sections. First, we add a single binary feature to indicate if the token contains a capital letter. Second, our tuning considers rectified linear units as the activation function for the CHARAGRAM and charCNN architectures.[1]

The results are shown in Table 25. Performance is similar across models. We found that adding a second fully-connected 150 dimensional layer to the CHARAGRAM model improved results slightly.[2]

1 We did not consider ReLU for the similarity experiments because the final embeddings are used directly to compute cosine similarities, which led to poor performance when restricting the embeddings to be non-negative.

2 We also tried adding a second (300 dimensional) layer for the word and sentence embedding models and found that it hurt performance.

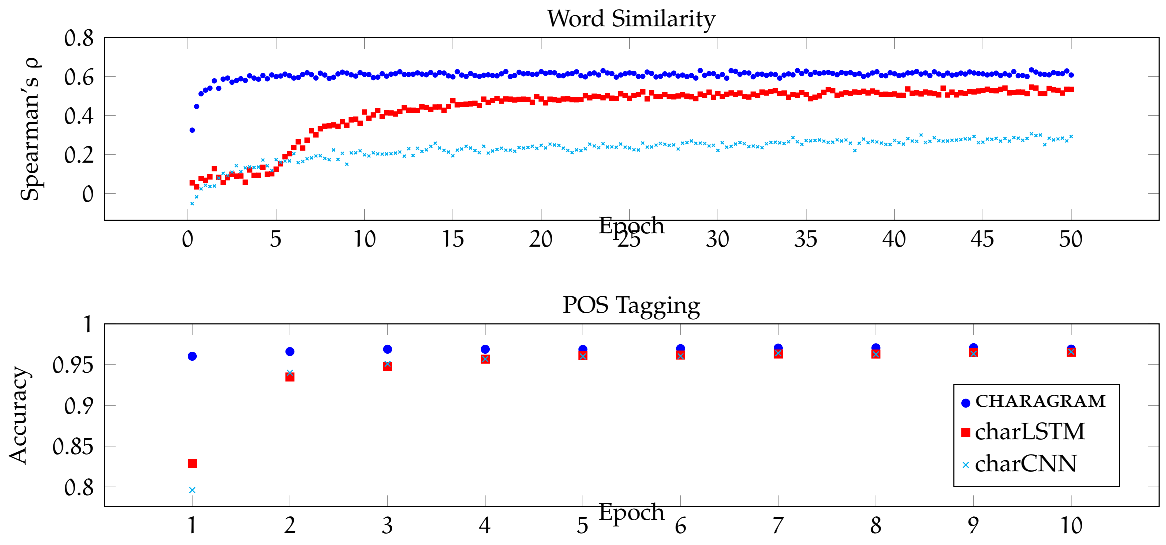


Figure 2: Plots of performance versus training epoch for word similarity and POS tagging.

5.4.4 *Convergence*

One observation we made during our experiments was that different models converged at significantly different rates. Figure 2 plots the performance of the word similarity and tagging tasks as a function of the number of examples processed during training. For word similarity, we plot the oracle Spearman's ρ on SL999, while for tagging we plot tagging accuracy on the validation set. We evaluate performance every quarter epoch (approximately every 194,252 word pairs) for word similarity and every epoch for tagging. We only show the first 10 epochs of training in the tagging plot.

The plots show that the CHARAGRAM model converges quickly to high performance. The charCNN and charLSTM models take many more epochs to converge. Even with tagging, which uses a very high learning rate, CHARAGRAM converges significantly faster than the others. For word similarity, it appears that charCNN and charLSTM are still slowly improving at the end of 50 epochs. This suggests that if training was done for a much longer period, and possibly on more data, the charLSTM or charCNN models could match and surpass the CHARAGRAM model. However, due to the large training sets available from PPDB and the computational requirements of these architectures, we were unable to explore the regime of training for many epochs. We conjecture that slow convergence could be the reason for the inferior performance of LSTMs for similarity tasks as reported in Chapter 4.

Task	# n-grams	2	2,3	2,3,4	2,3,4,5	2,3,4,5,6
POS Tagging	100	95.52	96.09	96.15	96.13	96.16
	1,000	96.72	96.86	96.97	97.02	97.03
	50,000	96.81	97.00	97.02	97.04	**97.09**
Word Similarity	100	6.2	7.0	7.7	9.1	8.8
	1,000	15.2	33.0	38.7	43.2	43.9
	50,000	14.4	52.4	67.8	69.2	**69.5**
Sentence Similarity	100	40.2	33.8	32.5	31.2	29.8
	1,000	50.1	60.3	58.6	56.6	55.6
	50,000	45.7	64.7	**66.6**	63.0	61.3

Table 26: Results of using different numbers and different combinations of character n-grams.

5.4.5 *Model Size Experiments*

The default setting for our charagram and charagram-phrase models is to use all character bigram, trigrams, and 4-grams that occur in the training data at least C times, tuning C over the set $\{1, 2\}$. This results in a large number of parameters, which could be seen as an unfair advantage over the comparatively smaller charCNN and charLSTM models, which have up to 881,025 and 763,200 parameters respectively in the similarity experiments.[3]

On the other hand, for a given training example, very few parameters in the charagram model are actually used. For the charCNN and charLSTM models, by contrast, all parameters are used except the character embeddings for those characters that are not present in the example. For a sentence with 100 characters, and when using the 300-dimensional charagram model with bigrams, trigrams, and 4-grams, there are approximately 90,000 parameters in use for this sentence, far fewer than those used by the charCNN and charLSTM for the same sentence.

We performed a series of experiments to investigate how the charagram and charagram-phrase models perform with different numbers and lengths of character n-grams. For a given k, we took the top k most frequent character n-grams for each value of n in use. We experimented with k values in $\{100, 1000, 50000\}$. If there were fewer than k unique character n-grams for a given n, we used all of them. For these experiments, we did very little tuning, setting the regularization strength to 0 and only tuning over the activation function. We repeated this experiment for all three of our tasks. For word similarity, we report performance on SL999 after training for 5 epochs on the lexical section of PPDB XXL. For sentence similarity, we report the average Pearson's r over all 22 datasets after training for 5 epochs on the phrasal section of PPDB XL. For tagging, we report accuracy on the validation set after training for 50 epochs. The results are shown in Table 26.

3 This includes 134 character embeddings.

When using extremely small models with only 100 n-grams of each order, we still see relatively strong performance on POS tagging. However, the semantic similarity tasks require far more n-grams to yield strong performance. Using 1000 n-grams clearly outperforms 100, and 50,000 n-grams performs best.

5.5 ANALYSIS

5.5.1 *Quantitative Analysis*

One of our primary motivations for character-based models is to address the issue of out-of-vocabulary (OOV) words, which were found to be one of the main sources of error for the PARAGRAM-PHRASE model from Chapter 4. They reported a negative correlation (Pearson's r of -0.45) between OOV rate and performance. We took the 12,108 sentence pairs in all 20 SemEval STS tasks and binned them by the total number of unknown words in the pairs.[4] We computed Pearson's r over each bin. The results are shown in Table 27.

Number of Unknown Words	N	PARAGRAM-PHRASE	CHARAGRAM-PHRASE
0	11,292	71.4	**73.8**
1	534	68.8	**78.8**
2	194	66.4	**72.8**
$\geqslant 1$	816	68.6	**77.9**
$\geqslant 0$	12,108	71.0	**74.0**

Table 27: Performance (Pearson's $r \times 100$) as a function of the number of unknown words in the sentence pairs over all 20 SemEval STS datasets. N is the number of sentence pairs.

The CHARAGRAM-PHRASE model has better performance for each number of unknown words. The PARAGRAM-PHRASE model degrades when more unknown words are present, presumably because it is forced to use the same unknown word embedding for all unknown words. The CHARAGRAM-PHRASE model has no notion of unknown words, as it can embed any character sequence.

We next investigated the sensitivity of the two models to length, as measured by the maximum of the lengths of the two sentences in a pair. We binned all of the 12,108 sentence pairs in the 20 SemEval STS tasks by length and then again found the Pearson's r for both the PARAGRAM-PHRASE and CHARAGRAM-PHRASE models. The results are shown in Table 28.

4 Unknown words were defined as those not present in the 1.7 million unique (case-insensitive) tokens that comprise the vocabulary for the GloVe embeddings available at `http://nlp.stanford.edu/projects/glove/`. The PARAGRAM-SL999 embeddings, used to initialize the PARAGRAM-PHRASE model, use this same vocabulary.

Max Length	N	PARAGRAM-PHRASE	CHARAGRAM-PHRASE
$\leqslant 4$	71	67.9	**72.9**
5	216	71.1	**71.9**
6	572	67.0	**69.7**
7	1,097	71.5	**74.0**
8	1,356	74.2	**74.5**
9	1,266	71.7	**72.7**
10	1,010	70.7	**74.2**
11-15	3,143	71.8	**73.7**
16-20	1,559	73.0	**75.1**
$\geqslant 21$	1,818	74.5	**75.4**

Table 28: Performance (Pearson's $r \times 100$) as a function of the maximum number of tokens in the sentence pairs over all 20 SemEval STS datasets. N is the number of sentence pairs.

We find that both models are robust to sentence length, achieving the highest correlations on the longest sentences. We also find that CHARAGRAM-PHRASE outperforms PARAGRAM-PHRASE at all sentence lengths.

5.5.2 *Qualitative Analysis*

bigram	CHARAGRAM-PHRASE	PARAGRAM-PHRASE
not capable	incapable, unable, incapacity	not, capable, stalled
not able	unable, incapable, incapacity	not, able, stalled
not possible	impossible impracticable unable	not, stalled, possible
not sufficient	insufficient, sufficient, inadequate	not, sufficient, stalled
not easy	easy, difficult, tough	not, stalled, easy

Table 29: Nearest neighboring words of selected bigrams under CHARAGRAM-PHRASE and PARAGRAM-PHRASE embeddings.

Aside from OOVs, the PARAGRAM-PHRASE model lacks the ability to model word order or cooccurrence, since it simply averages the words in the sequence. We were interested to see whether CHARAGRAM-PHRASE could handle negation, since it does model limited information about word order (via character n-grams that span multiple words in the sequence). We made a list of "not" bigrams that could be represented by a single word, then embedded each bigram using both models and did a nearest-neighbor search over a working vocabulary.[5] The results, in Table 29, show how the CHARAGRAM-PHRASE embeddings model negation. In all cases but one, the nearest neighbor is a paraphrase for the bigram and the next neighbors are mostly paraphrases as well. The PARAGRAM-PHRASE model, unsurprisingly, is incapable of modeling negation. In all cases, the nearest neighbor is *not*,

5 This contained all words in PPDB-XXL, our evaluations, and in two other datasets: the Stanford Sentiment task (Socher et al., 2013) and the SNLI dataset (Bowman et al., 2015), resulting in 93,217 unique (up-to-casing) tokens.

Word	CHARAGRAM-PHRASE
vehicals	vehical, vehicles, vehicels, vehicular, cars, vehicle, automobiles, car
serious-looking	serious, grave, acute, serious-minded, seriousness, gravity, serious-faced
near-impossible	impossible, hard/impossible, audacious-impossible, impractical, unable
growths	growth, grow, growing, increases, grows, increase, rise, growls, rising
litered	liter, litering, lited, liters, literate, literature, literary, literal, lite, obliterated
journeying	journey, journeys, voyage, trip, roadtrip, travel, tourney, voyages, road-trip
babyyyyyy	babyyyyyyy, baby, babys, babe, baby.i, babydoll, babycake, darling
adirty	dirty, dirtyyyyyy, filthy, down-and-dirty, dirtying, dirties, ugly, dirty-blonde
refunding	refunds, refunded, refund, repayment, reimbursement, rebate, repay reimbursements, reimburse, repaying, repayments, rebates, rebating, reimburses
professors	professor, professorships, professorship, teachers, professorial, teacher prof., teaches, lecturers, teachings, instructors, headteachers, teacher-student
huge	enormous, tremendous, large, big, vast, overwhelming, immense, giant formidable, considerable, massive, huger, large-scale, great, daunting

Table 30: Nearest neighbors of CHARAGRAM-PHRASE embeddings. Above the double horizontal line are nearest neighbors of words that were not in our training data, and below it are nearest neighbors of words that were in our training data.

as this word carries much more weight than the word it modifies. The remaining nearest neighbors are either the modified word or *stalled*.

We did two additional nearest neighbor explorations with our CHARAGRAM-PHRASE model. In the first, we collected the nearest neighbors for words that were not in the training data (i.e. PPDB XXL), but were in our working vocabulary. This consisted of 59,660 words. In the second, we collected nearest neighbors of words that were in our training data which consisted of 37,765 tokens.

A sample of the nearest neighbors is shown in Table 30. Several kinds of similarity are being captured simultaneously by the model. One kind is similarity in terms of spelling variation, including misspellings (*vehicals*, *vehicels*, and *vehicles*) and repetition for emphasis (*baby* and *babyyyyyyy*). Another kind is similarity in terms of morphological variants of a shared root (e.g., *journeying* and *journey*). We also see that the model has learned many strong synonym relationships without significant amounts of overlapping n-grams (e.g., *vehicles*, *cars*, and *automobiles*). We find these characteristics for words both in and out of the training data. Words in the training data, which tend to be more commonly used, do tend to have higher precision in their nearest neighbors (e.g., see neighbors for *huge*). We noted occasional mistakes for words that share a large number of n-grams but are not paraphrases (see nearest neighbors for *litered* which is likely a misspelling of *littered*).

Lastly, since our model learns embeddings for character n-grams, we include an analysis of character n-gram nearest neighbors in Table 31. These n-grams appear to be grouped into themes, such as death (first row), food (second row), and speed (third row), but have

n-gram	n-gram Embedding
die	_dy, _die, dead,_dyi, rlif, mort, ecea, rpse, d_aw
foo	_foo,_eat, meal, alim, trit, feed, grai,_din, nutr, toe
pee	peed, hast, spee, fast, mpo_, pace, _vel, loci, ccel
aiv	waiv, aive, boli, epea, ncel, abol, lift, bort, bol
ngu	ngue, uist, ongu, tong, abic, gual, fren, ocab, ingu
_2	2_, _02, _02_,_tw,_dua,_xx,_ii_, xx, 0_14, d_.2

Table 31: Nearest neighbors of character n-gram embeddings from our trained CHARAGRAM-PHRASE model. The underscore indicates a space, which signals the beginning or end of a word.

different granularities. The n-grams in the last row appear in paraphrases of 2, whereas the second-to-last row shows n-grams in words like *french* and *vocabulary*, which can broadly be classified as having to do with language.

5.6 CONCLUSION

We performed a careful empirical comparison of character-based compositional architectures on three NLP tasks. While most prior work has considered machine translation, language modeling, and syntactic analysis, we showed how character-level modeling can improve semantic similarity tasks, both quantitatively and with extensive qualitative analysis. We found a consistent trend: the simplest architecture converges fastest to high performance. These results, coupled with those from Chapter 4, suggest that practitioners should begin with simple architectures rather than moving immediately to RNNs and CNNs. We release our code and trained models so they can be used by the NLP community for general-purpose, character-based text representation.

Part II

AUTOMATICALLY CREATING PARAPHRASE CORPORA FOR IMPROVED SENTENCE EMBEDDING AND GENERATION TASKS

6

REVISITING RECURRENT NETWORKS FOR PARAPHRASTIC SENTENCE EMBEDDINGS

6.1 INTRODUCTION

Modeling sentential compositionality is a fundamental aspect of natural language semantics. Researchers have proposed a broad range of compositional functional architectures (Mitchell and Lapata, 2008; Socher et al., 2011; Kalchbrenner et al., 2014) and evaluated them on a large variety of applications. Our goal is to learn a general-purpose sentence embedding function that can be used unmodified for measuring semantic textual similarity (STS) (Agirre et al., 2012) and can also serve as a useful initialization for downstream tasks. We wish to learn this embedding function such that sentences with high semantic similarity have high cosine similarity in the embedding space. In particular, we focus on the setting of Chapter 4, in which models are trained on noisy paraphrase pairs and evaluated on both STS and supervised semantic tasks.

Surprisingly, in Chapter 4, we found that simple embedding functions—those based on averaging word vectors—outperform more powerful architectures based on long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997). In this chapter, we revisit their experimental setting and present several techniques that together improve the performance of the LSTM to be superior to word averaging.

We first change data sources: rather than train on noisy phrase pairs from the Paraphrase Database (PPDB; Ganitkevitch et al., 2013), we use noisy *sentence* pairs obtained automatically by aligning Simple English to standard English Wikipedia (Coster and Kauchak, 2011). Even though this data was intended for use by text simplification systems, we find it to be efficient and effective for learning sentence embeddings, outperforming much larger sets of examples from PPDB.

We then show how we can modify and regularize the LSTM to further improve its performance. The main modification is to simply average the hidden states instead of using the final one. For regularization, we experiment with two kinds of dropout and also with randomly scrambling the words in each input sequence. We find that these techniques help in the transfer learning setting and on two supervised semantic similarity datasets as well. Further gains are obtained on the supervised tasks by initializing with our models from the transfer setting.

Inspired by the strong performance of both averaging and LSTMs, we introduce a novel recurrent neural network architecture which we call the GATED RECURRENT AVERAGING NETWORK (GRAN). The GRAN outperforms averaging and the LSTM in both the transfer and supervised learning settings, forming a promising new recurrent architecture for semantic modeling.

6.2 RELATED WORK

Modeling sentential compositionality has received a great deal of attention in recent years. A comprehensive survey is beyond the scope of this paper, but we mention popular functional families: neural bag-of-words models (Kalchbrenner et al., 2014), deep averaging networks (DANs) (Iyyer et al., 2015), recursive neural networks using syntactic parses (Socher et al., 2011, 2012, 2013; Irsoy and Cardie, 2014), convolutional neural networks (Kalchbrenner et al., 2014; Kim, 2014; Hu et al., 2014a), and recurrent neural networks using long short-term memory (Tai et al., 2015; Ling et al., 2015a; Liu et al., 2015). Simple operations based on vector addition and multiplication typically serve as strong baselines (Mitchell and Lapata, 2008, 2010; Blacoe and Lapata, 2012).

Most work cited above uses a supervised learning framework, so the composition function is learned discriminatively for a particular task. In this paper, we are primarily interested in creating general purpose, domain independent embeddings for word sequences. Several others have pursued this goal (Socher et al., 2011; Le and Mikolov, 2014; Pham et al., 2015; Kiros et al., 2015; Hill et al., 2016; Arora et al., 2017; Pagliardini et al., 2017), though usually with the intent to extract useful features for supervised sentence tasks rather than to capture semantic similarity.

An exception is our work in Chapter 4. We closely follow this experimental setup and directly address some outstanding questions in our experimental results. Here we briefly summarize our main findings and attempts at explaining them. We made the surprising discovery that word averaging outperforms LSTMs by a wide margin in the transfer learning setting. We proposed several hypotheses for why this occurs. We first considered that the LSTM was unable to adapt to the differences in sequence length between phrases in training and sentences in test. This was ruled out by showing that neither model showed any strong correlation between sequence length and performance on the test data.

We next examined whether the LSTM was overfitting on the training data, but then showed that both models achieve similar values of the training objective and similar performance on *in-domain* held-out test sets. Lastly, we considered whether their hyperparameters were inadequately tuned, but extensive hyperparameter tuning did not change the story. Therefore, the reason for the performance gap, and how to correct it, was left as an open problem. This chapter takes steps toward addressing that problem.

6.3 MODELS AND TRAINING

6.3.1 *Models*

Our goal is to embed a word sequence s into a fixed-length vector. We focus on three compositional models in this paper, all of which use words as the smallest unit of compositionality. We denote the tth word in s as s_t, and we denote its word embedding by x_t.

Our first two models have been well-studied in prior work, so we describe them briefly. The first, which we call WORD, simply averages the embeddings x_t of all words in s. The only parameters learned in this model are those in the word embeddings themselves, which are stored in the word embedding matrix W_w. This model was found in Chapter 4 to perform very strongly for semantic similarity tasks.

Our second model uses a long short-term memory (LSTM) recurrent neural network (Hochreiter and Schmidhuber, 1997) to embed s. We use the LSTM variant from Gers et al. (2003) including its "peephole" connections. We consider two ways to obtain a sentence embedding from the LSTM. The first uses the final hidden vector, which we denote h_{-1}. The second, denoted LSTMAVG, averages all hidden vectors of the LSTM. In both variants, the learnable parameters include both the LSTM parameters W_c and the word embeddings W_w.

Inspired by the success of the two models above, we propose a third model, which we call the GATED RECURRENT AVERAGING NETWORK (GRAN). The GATED RECURRENT AVERAGING NETWORK combines the benefits of WORD and LSTMs. In fact it reduces to WORD if the output of the gate is all ones. We first use an LSTM to generate a hidden vector, h_t, for each word s_t in s. Then we use h_t to compute a gate that will be elementwise-multiplied with x_t, resulting in a new, gated hidden vector a_t for each step t:

$$a_t = x_t \odot \sigma(W_x x_t + W_h h_t + b)$$

where W_x and W_h are parameter matrices, b is a parameter vector, and σ is the elementwise logistic sigmoid function. After all a_t have been generated for a sentence, they are averaged to produce the embedding for that sentence. This model includes as learnable parameters those of the LSTM, the word embeddings, and the additional parameters in Eq. equation 6.3.1. For both the LSTM and GRAN models, we use W_c to denote the "compositional" parameters, i.e., all parameters other than the word embeddings.

The motivation for the GRAN is that we are contextualizing the word embeddings prior to averaging. The gate can be seen as an attention, attending to the prior context of the sentence.[1]

We also experiment with four other variations of this model, though they generally were more complex and showed inferior performance. In the first, GRAN-2, the gate is applied to h_t (rather than x_t) to produce a_t, and then these a_t are averaged as before.

GRAN-3 and GRAN-4 use two gates: one applied to x_t and one applied to a_{t-1}. We tried two different ways of computing these gates: for each gate i, $\sigma(W_{x_i} x_t + W_{h_i} h_t + b_i)$ (GRAN-3) or $\sigma(W_{x_i} x_t + W_{h_i} h_t + W_{a_i} a_{t-1} + b_i)$ (GRAN-4). The sum of these two terms comprised a_t. In this model, the last average hidden state, a_{-1}, was used as the sentence embedding after dividing it by the length of the sequence. In these models, we are additionally keeping a running average of the embeddings that is being modified by the context at every time step. In GRAN-4, this running average is also considered when producing the contextualized word embedding.

Lastly, we experimented with a fifth GRAN, GRAN-5, in which we use two gates, calculated by $\sigma(W_{x_i} x_t + W_{h_i} h_t + b_i)$ for each gate i. The first is applied to x_t and the second is applied to h_t. The output of these gates is then summed. Therefore GRAN-5 can be reduced to either word-averaging or averaging LSTM states, depending on the behavior of the gates. If the first gate is all ones and the second all zeros throughout the sequence, the model is equivalent to word-averaging. Conversely, if the first gate is all zeros and the

1 We tried a variant of this model without the gate. We obtain a_t from $f(W_x x_t + W_h h_t + b)$, where f is a nonlinearity, tuned over tanh and ReLU. The performance of the model is significantly worse than the GRAN in all experiments.

second is all ones throughout the sequence, the model is equivalent to averaging the LSTM states. Further analysis of these models is included in Section 6.4.

6.3.2 *Training*

We follow the training procedure of Chapters 3 and 4, described below. The training data consists of a set S of phrase or sentence pairs $\langle s_1, s_2 \rangle$ from either the Paraphrase Database (PPDB; Ganitkevitch et al., 2013) or the aligned Wikipedia sentences (Coster and Kauchak, 2011) where s_1 and s_2 are assumed to be paraphrases. We optimize a margin-based loss:

$$\min_{W_c, W_w} \frac{1}{|S|} \Bigg(\sum_{\langle s_1, s_2 \rangle \in S} \max(0, \delta - \cos(g(s_1), g(s_2)) \\ + \cos(g(s_1), g(t_1))) + \max(0, \delta - \cos(g(s_1), g(s_2)) \\ + \cos(g(s_2), g(t_2))) \Bigg) + \lambda_c \|W_c\|^2 + \lambda_w \left\|W_{w_{initial}} - W_w\right\|^2$$

where g is the model in use (e.g., WORD or LSTM), δ is the margin, λ_c and λ_w are regularization parameters, $W_{w_{initial}}$ is the initial word embedding matrix, and t_1 and t_2 are carefully-selected negative examples taken from a mini-batch during optimization. The intuition is that we want the two phrases to be more similar to each other ($\cos(g(s_1), g(s_2))$) than either is to their respective negative examples t_1 and t_2, by a margin of at least δ.

6.3.2.1 *Selecting Negative Examples*

To select t_1 and t_2 in Eq. equation 8.4, we simply choose the most similar phrase in some set of phrases (other than those in the given phrase pair). For simplicity we use the mini-batch for this set, but it could be a different set. That is, we choose t_1 for a given $\langle s_1, s_2 \rangle$ as follows:

$$t_1 = \underset{t:\langle t, \cdot \rangle \in S_b \setminus \{\langle s_1, s_2 \rangle\}}{\operatorname{argmax}} \cos(g(s_1), g(t))$$

where $S_b \subseteq S$ is the current mini-batch. That is, we want to choose a negative example t_i that is similar to s_i according to the current model. The downside is that we may occasionally choose a phrase t_i that is actually a true paraphrase of s_i.

6.4 EXPERIMENTS

Our experiments are designed to address the empirical question posed in Chapter 4: why do LSTMs underperform WORD for transfer learning? In Sections 6.4.1.2-6.4.2, we make progress on this question by presenting methods that bridge the gap between the two models in the transfer setting. We then apply these same techniques to improve performance in the supervised setting, described in Section 6.4.3. In both settings we also evaluate our novel GRAN architecture, finding it to consistently outperform both WORD and the LSTM.

6.4.1 *Transfer Learning*

6.4.1.1 *Datasets and Tasks*

We train on large sets of noisy paraphrase pairs and evaluate on a diverse set of 22 textual similarity datasets, including all datasets from every SemEval semantic textual similarity (STS) task from 2012 to 2015. We also evaluate on the SemEval 2015 Twitter task (Xu et al., 2015b) and the SemEval 2014 SICK Semantic Relatedness task (Marelli et al., 2014). Given two sentences, the aim of the STS tasks is to predict their similarity on a 0-5 scale, where 0 indicates the sentences are on different topics and 5 indicates that they are completely equivalent. We report the average Pearson's r over these 22 sentence similarity tasks.

Each STS task consists of 4-6 datasets covering a wide variety of domains, including newswire, tweets, glosses, machine translation outputs, web forums, news headlines, image and video captions, among others. Further details are provided in the official task descriptions (Agirre et al., 2012, 2013, 2014, 2015).

6.4.1.2 *Experiments with Data Sources*

We first investigate how different sources of training data affect the results. We try two data sources. The first is phrase pairs from the Paraphrase Database (PPDB). PPDB comes in different sizes (S, M, L, XL, XXL, and XXXL), where each larger size subsumes all smaller ones. The pairs in PPDB are sorted by a confidence measure and so the smaller sets contain higher precision paraphrases. PPDB is derived automatically from naturally-occurring bilingual text, and versions of PPDB have been released for many languages without the need for any manual annotation (Ganitkevitch and Callison-Burch, 2014).

	Word	LSTM	LSTMAvg
PPDB	67.7	54.2	64.2
SimpWiki	68.4	59.3	67.5

Table 32: Test results on SemEval semantic textual similarity datasets (Pearson's $r \times 100$) when training on different sources of data: phrase pairs from PPDB or simple-to-standard English Wikipedia sentence pairs from Coster and Kauchak (2011).

The second source of data is a set of sentence pairs automatically extracted from Simple English Wikipedia and English Wikipedia articles by Coster and Kauchak (2011). This data was extracted for developing text simplification systems, where each instance pairs a simple and complex sentence representing approximately the same information. Though the data was obtained for simplification, we use it as a source of training data for learning paraphrastic sentence embeddings. The dataset, which we call SimpWiki, consists of 167,689 sentence pairs.

To ensure a fair comparison, we select a sample of pairs from PPDB XL such that the number of tokens is approximately the same as the number of tokens in the SimpWiki sentences.[2]

We use paragram-sl999 embeddings from Chapter 3 to initialize the word embedding matrix (W_w) for all models. For all experiments, we fix the mini-batch size to 100, and λ_c to 0. We tune the margin δ over $\{0.4, 0.6, 0.8\}$ and λ_w over $\{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 0\}$. We train Word for 7 epochs, and the LSTM for 3, since it converges much faster and does not benefit from 7 epochs. For optimization we use Adam (Kingma and Ba, 2014) with a learning rate of 0.001. We use the 2016 STS tasks (Agirre et al., 2016) for model selection, where we average the Pearson's r over its 5 datasets. We refer to this type of model selection as *test*. For evaluation, we report the average Pearson's r over the 22 other sentence similarity tasks.

The results are shown in Table 32. We first note that, when training on PPDB, we find the same result as Chapter 4: Word outperforms the LSTM by more than 13 points. However, when training both on sentence pairs, the gap shrinks to about 9 points. It appears that part of the inferior performance for the LSTM in prior work was due to training on phrase pairs rather than on sentence pairs. The Word model also benefits from training on sentences, but not nearly as much as the LSTM.[3]

2 The PPDB data consists of 1,341,188 phrase pairs and contains 3 more tokens than the SimpWiki data.

3 We experimented with adding EOS tags at the end of training and test sentences, SOS tags at the start of training and test sentences, adding both, and adding neither. We treated adding these tags as hyperparameters and tuned over these four settings along with the other hyperparameters in the original experiment. Interestingly, we found that adding these tags, especially EOS, had a large effect on the LSTM when training on SimpWiki,

Our hypothesis explaining this result is that in PPDB, the "phrase pairs" are short fragments of text which are not necessarily constituents or phrases in any syntactic sense. Therefore, the sentences in the STS test sets are quite different from the fragments seen during training. We hypothesize that while word-averaging is relatively unaffected by this difference, the recurrent models are much more sensitive to overall characteristics of the word sequences, and the difference between train and test matters much more.

These results also suggest that the SimpWiki data, even though it was developed for text simplification, may be useful for other researchers working on semantic textual similarity tasks.

6.4.1.3 *Experiments with LSTM Variations*

We next compare LSTM and LSTMAvg. The latter consists of averaging the hidden vectors of the LSTM rather than using the final hidden vector as in prior work (Chapter 4). We hypothesize that the LSTM may put more emphasis on the words at the end of the sentence than those at the beginning. By averaging the hidden states, the impact of all words in the sequence is better taken into account. Averaging also makes the LSTM more like Word, which we know to perform strongly in this setting.

The results on Word and the LSTM models are shown in Table 32. When training on PPDB, moving from LSTM to LSTMAvg improves performance by 10 points, closing most of the gap with Word. We also find that LSTMAvg improves by moving from PPDB to SimpWiki, though in both cases it still lags behind Word.

6.4.2 *Experiments with Regularization*

We next experiment with various forms of regularization. In Chapters 4 and 5, we only used L2 regularization. In Chapter 4 also regularized the word embeddings back to their initial values. Here we use L2 regularization as well as several additional regularization methods we describe below.

We try two forms of dropout. The first is just standard dropout (Srivastava et al., 2014) on the word embeddings. The second is "word dropout", which drops out entire word embeddings with some probability (Iyyer et al., 2015).

improving performance by 6 points. When training on PPDB, adding EOS tags only improved performance by 1.6 points.
The addition of the tags had a smaller effect on LSTMAvg. Adding EOS tags improved performance by 0.3 points on SimpWiki and adding SOS tags on PPDB improved performance by 0.9 points.

We also experiment with scrambling the inputs. For a given mini-batch, we go through each sentence pair and, with some probability, we shuffle the words in each sentence in the pair. When scrambling a sentence pair, we always shuffle both sentences in the pair. We do this before selecting negative examples for the mini-batch. The motivation for scrambling is to make it more difficult for the LSTM to memorize the sequences in the training data, forcing it to focus more on the identities of the words and less on word order. Hence it will be expected to behave more like the word averaging model.[4]

We also experiment with combining scrambling and dropout. In this setting, we tune over scrambling with either word dropout or dropout.

The settings for these experiments are largely the same as those of the previous section with the exception that we tune λ_w over a smaller set of values: $\{10^{-5}, 0\}$. When using L_2 regularization, we tune λ_c over $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. When using dropout, we tune the dropout rate over $\{0.2, 0.4, 0.6\}$. When using scrambling, we tune the scrambling rate over $\{0.25, 0.5, 0.75\}$. We also include a bidirectional model ("Bi") for both LSTMAvg and the GATED RECURRENT AVERAGING NETWORK. We tune over two ways to combine the forward and backward hidden states; the first simply adds them together and the second uses a single feedforward layer with a tanh activation.

We try two approaches for model selection. The first, *test*, is the same as was done in Section 6.4.1.2, where we use the average Pearson's r on the 5 2016 STS datasets. The second tunes based on the average Pearson's r of all 22 datasets in our evaluation. We refer to this as *oracle*.

The results are shown in Table 33. They show that dropping entire word embeddings and scrambling input sequences is very effective in improving the result of the LSTM, while neither type of dropout improves WORD. Moreover, averaging the hidden states of the LSTM is the most effective modification to the LSTM in improving performance. All of these modifications can be combined to significantly improve the LSTM, finally allowing it to overtake WORD.

In Table 34, we compare the various GRAN architectures. We find that the GRAN provides a small improvement over the best LSTM configuration, possibly because of its similarity to WORD. It also outperforms the other GRAN models, despite being the simplest.

In Table 35, we show results on all individual STS evaluation datasets after using STS 2016 for model selection (unidirectional models only). The LSTMAvg and GATED RECUR-

4 We also tried some variations on scrambling that did not yield significant improvements: scrambling after obtaining the negative examples, partially scrambling by performing n swaps where n comes from a Poisson distribution with a tunable λ, and scrambling individual sentences with some probability instead of always scrambling both in the pair.

Model	Regularization	Oracle	2016 STS
WORD	none	68.5	68.4
	dropout	68.4	68.3
	word dropout	68.3	68.3
LSTM	none	60.6	59.3
	L_2	60.3	56.5
	dropout	58.1	55.3
	word dropout	66.2	65.3
	scrambling	66.3	65.1
	dropout, scrambling	68.4	68.4
LSTMAVG	none	67.7	67.5
	dropout, scrambling	69.2	68.6
BiLSTMAVG	dropout, scrambling	**69.4**	**68.7**

Table 33: Results on SemEval textual similarity datasets (Pearson's $r \times 100$) when experimenting with different regularization techniques.

Model	Oracle	STS 2016
GRAN (no reg.)	68.0	68.0
GRAN	69.5	**68.9**
GRAN-2	68.8	68.1
GRAN-3	69.0	67.2
GRAN-4	68.6	68.1
GRAN-5	66.1	64.8
BiGRAN	**69.7**	68.4

Table 34: Results on SemEval textual similarity datasets (Pearson's $r \times 100$) for the GRAN architectures. The first row, marked as (no reg.) is the GRAN without any regularization. The other rows show the result of the various GRAN models using dropout and scrambling.

RENT AVERAGING NETWORK are more closely correlated in performance, in terms of Spearman's ρ and Pearson'r r, than either is to WORD. But they do differ significantly in some datasets, most notably in those comparing machine translation output with its reference. Interestingly, both the LSTMAVG and GATED RECURRENT AVERAGING NETWORK significantly outperform WORD in the datasets focused on comparing glosses like *OnWN* and *FNWN*. Upon examination, we found that these datasets, especially 2013 *OnWN*, contain examples of low similarity with high word overlap. For example, the pair ⟨*the act of preserving or protecting something., the act of decreasing or reducing something.*⟩ from 2013 *OnWN* has a gold similarity score of 0.4. It appears that WORD was fooled by the high amount of word overlap in such pairs, while the other two models were better able to recognize the semantic differences.

Dataset	LSTMAvg	Word	GRAN
MSRpar	**49.0**	45.9	47.7
MSRvid	84.3	85.1	**85.2**
SMT-eur	**51.2**	47.5	49.3
OnWN	**71.5**	71.2	71.5
SMT-news	**68.0**	58.2	58.7
STS 2012 Average	**64.8**	61.6	62.5
headline	**77.3**	76.9	76.1
OnWN	81.2	72.8	**81.4**
FNWN	53.2	50.2	**55.6**
SMT	**40.7**	38.0	40.3
STS 2013 Average	63.1	59.4	**63.4**
deft forum	**56.6**	55.6	55.7
deft news	78.0	**78.5**	77.1
headline	74.5	**75.1**	72.8
images	84.7	85.6	**85.8**
OnWN	84.9	81.4	**85.1**
tweet news	76.3	**78.7**	78.7
STS 2014 Average	75.8	75.8	**75.9**
answers-forums	71.8	70.6	**73.1**
answers-students	71.1	**75.8**	72.9
belief	75.3	76.8	**78.0**
headline	79.5	**80.3**	78.6
images	85.8	**86.0**	85.8
STS 2015 Average	76.7	**77.9**	77.7
2014 SICK	71.3	72.4	**72.9**
2015 Twitter	**52.1**	**52.1**	50.2

Table 35: Results on SemEval textual similarity datasets (Pearson's $r \times 100$). The highest score in each row is in boldface.

6.4.3 *Supervised Text Similarity*

We also investigate if these techniques can improve LSTM performance on supervised semantic textual similarity tasks. We evaluate on two supervised datasets. For the first, we start with the 20 SemEval STS datasets from 2012-2015 and then use 40% of each dataset for training, 10% for validation, and the remaining 50% for testing. There are 4,481 examples in training, 1,207 in validation, and 6,060 in the test set. The second is the SICK 2014 dataset, using its standard training, validation, and test sets. There are 4,500 sentence pairs in the training set, 500 in the development set, and 4,927 in the test set. The SICK task is an easier learning problem since the training examples are all drawn from the same distribution, and they are mostly shorter and use simpler language. As these are supervised tasks, the sentence pairs in the training set contain manually-annotated semantic similarity scores.

We minimize the loss function[5] from Tai et al. (2015). Given a score for a sentence pair in the range $[1, K]$, where K is an integer, with sentence representations h_L and h_R, and model parameters θ, they first compute:

$$\begin{aligned}
h_\times &= h_L \odot h_R, \;\; h_+ = |h_L - h_R|, \\
h_s &= \sigma\left(W^{(\times)} h_\times + W^{(+)} h_+ + b^{(h)}\right), \\
\hat{p}_\theta &= \text{softmax}\left(W^{(p)} h_s + b^{(p)}\right), \\
\hat{y} &= r^T \hat{p}_\theta,
\end{aligned}$$

where $r^T = [1\ 2\ \ldots\ K]$. They then define a sparse target distribution p that satisfies $y = r^T p$:

$$p_i = \begin{cases} y - \lfloor y \rfloor, & i = \lfloor y \rfloor + 1 \\ \lfloor y \rfloor - y + 1, & i = \lfloor y \rfloor \\ 0 & \text{otherwise} \end{cases}$$

for $1 \leqslant i \leqslant K$. Then they use the following loss, the regularized KL-divergence between p and $\hat{p}_\theta$:

$$J(\theta) = \frac{1}{m} \sum_{k=1}^{m} \text{KL}\left(p^{(k)} \;\middle\|\; \hat{p}_\theta^{(k)}\right),$$

where m is the number of training pairs.

5 This objective function has been shown to perform very strongly on text similarity tasks, significantly better than squared or absolute error.

Model	Regularization	STS	SICK	Avg.
WORD	none	79.2	85.2	82.2
	dropout	80.7	84.5	82.6
	word dropout	79.3	81.8	80.6
LSTM	none	68.4	80.9	74.7
	dropout	69.6	81.3	75.5
	word dropout	68.0	76.4	72.2
	scrambling	74.2	84.4	79.3
	dropout, scrambling	75.0	84.2	79.6
LSTMAVG	none	69.0	79.5	74.3
	dropout	69.2	79.4	74.3
	word dropout	65.6	76.1	70.9
	scrambling	76.5	83.2	79.9
	dropout, scrambling	76.5	84.0	80.3
GRAN	none	79.7	85.2	82.5
	dropout	79.7	84.6	82.2
	word dropout	77.3	83.0	80.2
	scrambling	81.4	**85.3**	**83.4**
	dropout, scrambling	**81.6**	85.1	**83.4**

Table 36: Results from supervised training on the STS and SICK datasets (Pearson's $r \times 100$). The last column is the average result on the two datasets.

Model	STS	SICK	Avg.
GRAN	**81.6**	85.3	**83.5**
GRAN-2	77.4	85.1	81.3
GRAN-3	81.3	85.4	83.4
GRAN-4	80.1	**85.5**	82.8
GRAN-5	70.9	83.0	77.0

Table 37: Results from supervised training on the STS and SICK datasets (Pearson's $r \times 100$) for the GRAN architectures. The last column is the average result on the two datasets.

We experiment with the LSTM, LSTMAVG, and WORD models with dropout, word dropout, and scrambling tuning over the same hyperparameter as in Section 6.4.2. We again regularize the word embeddings back to their initial state, tuning λ_w over $\{10^{-5}, 0\}$. We used the validation set for each respective dataset for model selection.

The results are shown in Table 36. The GATED RECURRENT AVERAGING NETWORK has the best performance on both datasets. Dropout helps the word-averaging model in the STS task, unlike in the transfer learning setting. The LSTM benefits slightly from dropout, scrambling, and averaging on their own individually with the exception of word dropout on both datasets and averaging on the SICK dataset. However, when combined, these modifications are able to significantly improve the performance of the LSTM, bringing it much closer in performance to WORD. This experiment indicates that these modifications when training

LSTMs are beneficial outside the transfer learning setting, and can potentially be used to improve performance for the broad range of problems that use LSTMs to model sentences.

In Table 37 we compare the various GRAN architectures under the same settings as the previous experiment. We find that the GRAN still has the best overall performance.

#	Sentence 1	Sentence 2	LAVG	WORD	Gold
1	the lamb is looking at the camera.	a cat looking at the camera.	**3.42**	4.13	0.8
2	he also said shockey is "living the dream life of a new york athlete.	"jeremy's a good guy," barber said, adding:"jeremy is living the dream life of the new york athlete.	**3.55**	4.22	2.75
3	bloomberg chips in a billion	bloomberg gives $1.1 b to university	**3.99**	3.04	4.0
4	in other regions, the sharia is imposed.	in other areas, sharia law is being introduced by force.	**4.44**	3.72	4.75
5	three men in suits sitting at a table.	two women in the kitchen looking at a object.	3.33	**2.79**	0.0
6	we never got out of it in the first place!	where does the money come from in the first place?	4.00	**3.33**	0.8
7	two birds interacting in the grass.	two dogs play with each other outdoors.	3.44	**2.81**	0.2

Table 38: Illustrative sentence pairs from the STS datasets showing errors made by LSTMAVG and WORD. The last three columns show the gold similarity score, the similarity score of LSTMAVG, and the similarity score of WORD. Boldface indicates smaller error compared to gold scores.

We also experiment with initializing the supervised models using our pretrained sentence model parameters, for the WORD model (no regularization), LSTMAVG (dropout, scrambling), and GATED RECURRENT AVERAGING NETWORK (dropout, scrambling) models from Table 33 and Table 34. We both initialize and then regularize back to these initial values, referring to this setting as "universal".[6]

The results are shown in Table 39. Initializing and regularizing to the pretrained models significantly improves the performance for all three models, justifying our claim that these models serve a dual purpose: they can be used a black box semantic similarity function, and they possess rich knowledge that can be used to improve the performance of downstream tasks.

6.5 ANALYSIS

6.5.1 *Error Analysis*

We analyze the predictions of WORD and the recurrent networks, represented by LSTMAVG, on the 20 STS datasets. We choose LSTMAVG as it correlates slightly less strongly with WORD

6 In these experiments, we tuned λ_w over $\{10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 0\}$ and λ_c over $\{10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 0\}$.

Model	Regularization	STS	SICK
Word	dropout	80.7	84.5
	dropout, universal	**82.9**	85.6
LSTMAvg	dropout, scrambling	76.5	84.0
	dropout, scrambling, universal	81.3	85.2
GRAN	dropout, scrambling	81.6	85.1
	dropout, scrambling, universal	82.7	**86.0**

Table 39: Impact of initializing and regularizing toward universal models (Pearson's $r \times 100$) in supervised training.

than the GRAN on the results over all SemEval datasets used for evaluation. We scale the models' cosine similarities to lie within $[0, 5]$, then compare the predicted similarities of LSTMAvg and Word to the gold similarities. We analyzed instances in which each model would tend to overestimate or underestimate the gold similarity relative to the other. These are illustrated in Table 38.

We find that Word tends to overestimate the semantic similarity of a sentence pair, relative to LSTMAvg, when the two sentences have a lot of word or synonym overlap, but have either important differences in key semantic roles or where one sentence has significantly more content than the other. These phenomena are shown in examples 1 and 2 in Table 38. Conversely, Word tends to underestimate similarity when there are one-word-to-multiword paraphrases between the two sentences as shown in examples 3 and 4.

LSTMAvg tends to overestimate similarity when the two inputs have similar sequences of syntactic categories, but the meanings of the sentences are different (examples 5, 6, and 7). Instances of LSTMAvg underestimating the similarity relative to Word are relatively rare, and those that we found did not have any systematic patterns.

6.5.2 GRAN *Gate Analysis*

We also investigate what is learned by the gating function of the GATED RECURRENT AVERAGING NETWORK. We are interested to see whether its estimates of importance correlate with those of traditional syntactic and (shallow) semantic analysis.

We use the oracle trained GATED RECURRENT AVERAGING NETWORK from Table 34 and calculate the L_1 norm of the gate after embedding 10,000 sentences from English Wikipedia.[7] We also automatically tag and parse these sentences using the Stanford dependency parser (Man-

[7] We selected only sentences of less than or equal to 15 tokens to ensure more accurate parsing.

POS		Dep. Label	
top 10	bot. 10	top 10	bot. 10
NNP	TO	number	possessive
NNPS	WDT	nn	cop
CD	POS	num	det
NNS	DT	acomp	auxpass
VBG	WP	appos	prep
NN	IN	pobj	cc
JJ	CC	vmod	mark
UH	PRP	dobj	aux
VBN	EX	amod	expl
JJS	WRB	conj	neg

Table 40: POS tags and dependency labels with highest and lowest average GATED RECURRENT AVERAGING NETWORK gate L_1 norms. The lists are ordered from highest norm to lowest in the top 10 columns, and lowest to highest in the bottom 10 columns.

Dep. Label	Weight
xcomp	170.6
acomp	167.1
root	157.4
amod	143.1
advmod	121.6

Table 41: Average L_1 norms for adjectives (JJ) with selected dependency labels.

ning et al., 2014). We then compute the average gate L_1 norms for particular part-of-speech tags, dependency arc labels, and their conjunction.

Table 40 shows the highest/lowest average norm tags and dependency labels. The network prefers nouns, especially proper nouns, as well as cardinal numbers, which is sensible as these are among the most discriminative features of a sentence.

Analyzing the dependency relations, we find that nouns in the object position tend to have higher weight than nouns in the subject position. This may relate to topic and focus; the object may be more likely to be the "new" information related by the sentence, which would then make it more likely to be matched by the other sentence in the paraphrase pair.

We find that the weights of adjectives depend on their position in the sentence, as shown in Table 41. The highest norms appear when an adjective is an xcomp, acomp, or root; this typically means it is residing in an object-like position in its clause. Adjectives that modify a noun (amod) have medium weight, and those that modify another adjective or verb (advmod) have low weight.

Dep. Label	Weight
pcomp	190.0
amod	178.3
xcomp	176.8
vmod	170.6
root	161.8
auxpass	125.4
prep	121.2

Table 42: Average L_1 norms for words with the tag VBG with selected dependency labels.

Lastly, we analyze words tagged as VBG, a highly ambiguous tag that can serve many syntactic roles in a sentence. As shown in Table 42, we find that when they are used to modify a noun (amod) or in the object position of a clause (xcomp, pcomp) they have high weight. Medium weight appears when used in verb phrases (root, vmod) and low weight when used as prepositions or auxiliary verbs (prep, auxpass).

6.6 CONCLUSION

We showed how to modify and regularize LSTMs to improve their performance for learning paraphrastic sentence embeddings in both transfer and supervised settings. We also introduced a new recurrent network, the GATED RECURRENT AVERAGING NETWORK, that improves upon both WORD and LSTMs for these tasks, and we release our code and trained models.

Furthermore, we analyzed the different errors produced by WORD and the recurrent methods and found that the recurrent methods were learning composition that wasn't being captured by WORD. We also investigated the GRAN in order to better understand the compositional phenomena it was learning by analyzing the L_1 norm of its gate over various inputs.

Future work will explore additional data sources, including from aligning different translations of novels (Barzilay and McKeown, 2001), aligning new articles of the same topic (Dolan et al., 2004), or even possibly using machine translation systems to translate bilingual text into paraphrastic sentence pairs. Our new techniques, combined with the promise of new data sources, offer a great deal of potential for improved universal paraphrastic sentence embeddings.

7 LEARNING PARAPHRASTIC SENTENCE EMBEDDINGS FROM BACK-TRANSLATED BITEXT

7.1 INTRODUCTION

Pretrained word embeddings have received a great deal of attention from the research community, but there is much less work on developing pretrained embeddings for *sentences*. Here we target sentence embeddings that are "paraphrastic" in the sense that two sentences with similar meanings are close in the embedding space. In 4, we developed paraphrastic sentence embeddings that are useful for semantic textual similarity tasks and can also be used as initialization for supervised semantic tasks.

To learn these sentence embeddings, we used the Paraphrase Database (PPDB) (Ganitkevitch et al., 2013). PPDB contains a large set of paraphrastic textual fragments extracted automatically from bilingual text ("bitext"), which is readily available for languages and domains. Versions of PPDB have been released for several languages (Ganitkevitch and Callison-Burch, 2014).

However, in 6, we showed that the fragmental nature of PPDB's pairs can be problematic, especially for recurrent networks. Better performance can be achieved with a smaller set of sentence pairs derived from aligning Simple English and standard English Wikipedia (Coster and Kauchak, 2011). While effective, this type of data is inherently limited in size and scope, and not available for languages other than English.

PPDB is appealing in that it only requires bitext. We would like to retain this property but develop a data resource with sentence pairs rather than phrase pairs. We turn to neural machine translation (NMT) (Sutskever et al., 2014; Bahdanau et al., 2015; Sennrich et al., 2016b), which has matured recently to yield strong performance especially in terms of producing grammatical outputs.

In this chapter, we build NMT systems for three language pairs, then use them to back-translate the non-English side of the training bitext. The resulting data consists of sentence pairs containing an English reference and the output of an X-to-English NMT system. Table 43 shows examples. We use this data for training paraphrastic sentence embeddings,

R:	We understand that has already commenced, but there is a long way to go.
T:	This situation has already commenced, but much still needs to be done.
R:	The restaurant is closed on Sundays. No breakfast is available on Sunday mornings.
T:	The restaurant stays closed Sundays so no breakfast is served these days.
R:	Improved central bank policy is another huge factor.
T:	Another crucial factor is the improved policy of the central banks.

Table 43: Illustrative examples of references (R) paired with back-translations (T).

yielding results that are much stronger than when using PPDB and competitive with the Simple English Wikipedia data.

Since bitext is abundant and available for many language pairs and in many data domains, we also develop several methods of filtering the data, including based on sentence length, quality measures, and measures of difference between the reference and its back-translation. We find length to be an effective filtering method, showing that very short length ranges—where the translation is 1 to 10 words—are best for learning.

In studying quality measures for filtering, we train a classifier to predict if a sentence is a reference or a back-translation, then score sentences by the classifier score. This investigation allows us to examine the kinds of phenomena that best distinguish NMT output from references in this controlled setting of translating the bitext training data. NMT output has more repetitions of both words and longer n-grams, and uses fewer rare words than the references.

We release our generated sentence pairs to the research community with the hope that the data can inspire others to develop additional filtering methods, to experiment with richer architectures for sentence embeddings, and to further analyze the differences between neural machine translations and references.

7.2 RELATED WORK

We describe related work in learning general-purpose sentence embeddings, work in automatically generating or discovering paraphrases, and finally prior work in leveraging neural machine translation for embedding learning.

PARAPHRASTIC SENTENCE EMBEDDINGS. Our learning and evaluation setting is the same as that considered in Chapters 4 and 5, in which the goal is to learn paraphrastic sentence embeddings that can be used for downstream tasks. They trained models on PPDB and evaluated them using a suite of semantic textual similarity (STS) tasks and supervised semantic tasks. Others have begun to consider this setting as well (Arora et al., 2017).

Other work in learning general purpose sentence embeddings has used autoencoders (Socher et al., 2011; Hill et al., 2016), encoder-decoder architectures (Kiros et al., 2015), or other learning frameworks (Le and Mikolov, 2014; Pham et al., 2015). Chapter 4 and Hill et al. (2016) provide many empirical comparisons to this prior work. For conciseness, we compare only to the strongest configurations from their results.

PARAPHRASE GENERATION AND DISCOVERY. There is a rich history of research in generating or finding naturally-occurring sentential paraphrases (Barzilay and McKeown, 2001; Dolan et al., 2004; Dolan and Brockett, 2005; Quirk et al., 2004; Coster and Kauchak, 2011; Xu et al., 2014, 2015b).

The most relevant work uses bilingual corpora, e.g., Bannard and Callison-Burch (2005), culminating in PPDB. Our goals are highly similar to those of the PPDB project, which has also been produced for many languages (Ganitkevitch and Callison-Burch, 2014) since it only relies on the availability of bilingual text.

Prior work has shown that PPDB can be used for learning embeddings for words and phrases (Faruqui et al., 2015; Wieting et al., 2015). However, when learning sentence embeddings in Chapter 6, we showed that PPDB is not as effective as sentential paraphrases, especially for recurrent networks. These results are intuitive because the phrases in PPDB are short and often cut across constituent boundaries. For sentential paraphrases, we used a dataset developed for text simplification by Coster and Kauchak (2011) in Chapter 6. It was created by aligning sentences from Simple English and standard English Wikipedia. We compare our data to both PPDB and this Wikipedia dataset.

NEURAL MACHINE TRANSLATION FOR PARAPHRASTIC EMBEDDING LEARNING. Sutskever et al. (2014) trained NMT systems and visualized part of the space of the source language encoder for their English→French system. Hill et al. (2016) evaluated the encoders of English-to-X NMT systems as sentence representations, finding them to perform poorly compared to several other methods based on unlabeled data. Mallinson et al. (2017) adapted trained NMT models to produce sentence similarity scores in semantic evaluations. They used pairs of NMT systems, one to translate an English sentence into multiple foreign translations and the other to then translate back to English. Other work has used neural MT architectures and training settings to obtain better word embeddings (Hill et al., 2014a,b).

Our approach differs in that we only use the NMT system to generate training data for training sentence embeddings, rather than use it as the source of the model. This permits us to decouple decisions made in designing the NMT architecture from decisions about which models we will use for learning sentence embeddings. Thus we can benefit from orthogonal work in designing neural architectures to embed sentences.

7.3 NEURAL MACHINE TRANSLATION

We now describe the NMT systems we use for generating data for learning sentence embeddings. In our experiments, we use three encoder-decoder NMT models: Czech→English, French→English, and German→English.

We used Groundhog[1] as the implementation of the NMT systems for all experiments. We generally followed the settings and training procedure from previous work (Bahdanau et al., 2015; Sennrich et al., 2016b). As such, all networks have a hidden layer size of 1000 and an embedding layer size of 620. During training, we used Adadelta (Zeiler, 2012), a minibatch size of 80, and the training set was reshuffled between epochs. We trained a network for approximately 7 days on a single GPU (TITAN X), then the embedding layer was fixed and training continued, as suggested by Jean et al. (2015), for 12 hours. Additionally, the softmax was calculated over a filtered list of candidate translations. Following Jean et al. (2015), during decoding, we restrict the softmax layers' output vocabulary to include: the 10000 most common words, the top 25 unigram translations, and the gold translations' unigrams.

All systems were trained on the available training data from the WMT15 shared translation task (15.7 million, 39.2 million, and 4.2 million sentence pairs for CS→EN, FR→EN, and DE→EN, respectively). The training data included: Europarl v7 (Koehn, 2005), the Common Crawl corpus, the UN corpus (Eisele and Chen, 2010), News Commentary v10, the 10^9 French-English corpus, and CzEng 1.0 (Bojar et al., 2016). A breakdown of the sizes of these corpora can be found in Table 44. The data was pre-processed using standard pre-processing scripts found in Moses (Koehn et al., 2007). Rare words were split into sub-word units, following Sennrich et al. (2016c). BLEU scores on the WMT2015 test set for each NMT system can be seen in Table 45.

To produce paraphrases we use "back-translation", i.e., we use our X→English NMT systems to translate the non-English sentence in each training sentence pair into English.

1 Available at https://github.com/sebastien-j/LV_groundhog.

	Czech	French	German
Europarl	650,000	2,000,000	2,000,000
Common Crawl	160,000	3,000,000	2,000,000
News Commentary	150,000	200,000	200,000
UN	-	12,000,000	-
10^9 French-English	-	22,000,000	-
CzEng	14,700,000	-	-

Table 44: Dataset sizes (numbers of sentence pairs) for data domains used for training NMT systems.

Language	% BLEU
Czech→English	19.7
French→English	20.1
German→English	28.2

Table 45: BLEU scores on the WMT2015 test set.

We directly use the bitext on which the models were trained. This could potentially lead to pairs in which the reference and translation match exactly, if the model has learned to memorize the reference translations seen during training. However, in practice, since we have so much bitext to draw from, we can easily find data in which they do not match exactly.

Thus our generated data consists of pairs of English references from the bitext along with the NMT-produced English back-translations. We use beam search with a width of 50 to generate multiple translations for each non-English sentence, each of which is a candidate paraphrase for the English reference.

Example outputs of this process are in Table 43, showing some rich paraphrase phenomena in the data. These examples show non-trivial phrase substitutions ("there is a long way to go" and "much still needs to be done"), sentences being merged and simplified, and sentences being rearranged. For examples of erroneous paraphrases that can be generated by this process, see Table 53.

7.4 MODELS AND TRAINING

Our goal is to compare our paraphrase dataset to other datasets by using each to train sentence embeddings, keeping the models and learning procedure fixed. So we select models and a loss function from Chapters 4 and 6.

7.4.1 *Models*

We wish to embed a word sequence s into a fixed-length vector. We denote the tth word in s as s_t, and we denote its word embedding by x_t. We focus on two models in this paper. The first model, which we call WORD, simply averages the embeddings x_t of all words in s. The only parameters learned in this model are those in the word embeddings themselves, which are stored in the word embedding matrix W_w. This model was found in Chapter 4 to perform very strongly for semantic similarity tasks.

The second model, the GATED RECURRENT AVERAGING NETWORK (GRAN) from Chapter 6, combines the benefits of WORD and long short-term memory (LSTM) recurrent neural networks (Hochreiter and Schmidhuber, 1997). It first uses an LSTM to generate a hidden vector, h_t, for each word s_t in s. Then h_t is used to compute a gate that is elementwise-multiplied with x_t, resulting in a new hidden vector a_t for each step t:

$$a_t = x_t \odot \sigma(W_x x_t + W_h h_t + b)$$

where W_x and W_h are parameter matrices, b is a parameter vector, and σ is the elementwise logistic sigmoid function. After all a_t have been generated for a sentence, they are averaged to produce the embedding for that sentence. The GRAN reduces to WORD if the output of the gate is always 1. This model includes as learnable parameters those of the LSTM, the word embeddings, and the additional parameters in Eq. equation 7.4.1. We use W_c to denote the "compositional" parameters, i.e., all parameters other than the word embeddings.

7.4.2 *Training*

We follow the training procedure from Chapters 3 and 4. The training data is a set S of paraphrastic pairs $\langle s_1, s_2\rangle$ and we optimize a margin-based loss:

$$\begin{aligned}
&\min_{W_c, W_w} \frac{1}{|S|}\Bigg(\sum_{\langle s_1, s_2\rangle \in S} \max(0, \delta - \cos(g(s_1), g(s_2)) \\
&+ \cos(g(s_1), g(t_1))) + \max(0, \delta - \cos(g(s_1), g(s_2)) \\
&+ \cos(g(s_2), g(t_2)))\Bigg) + \lambda_c \left\|W_c\right\|^2 + \lambda_w \left\|W_{w_{initial}} - W_w\right\|^2
\end{aligned}$$

where g is the model (WORD or GRAN), δ is the margin, λ_c and λ_w are regularization parameters, $W_{w_{initial}}$ is the initial word embedding matrix, and t_1 and t_2 are "negative examples" taken from a mini-batch during optimization. The intuition is that we want the two texts to be more similar to each other ($\cos(g(s_1), g(s_2))$) than either is to their respective negative examples t_1 and t_2, by a margin of at least δ. To select t_1 and t_2, we choose the most similar sentence in some set (other than those in the given pair). For simplicity we use the mini-batch for this set, i.e., we choose t_1 for a given $\langle s_1, s_2\rangle$ as follows:

$$t_1 = \operatorname*{argmax}_{t:\langle t,\cdot\rangle \in S_b \setminus \{\langle s_1, s_2\rangle\}} \cos(g(s_1), g(t))$$

where $S_b \subseteq S$ is the current mini-batch. That is, we want to choose a negative example t_i that is similar to s_i according to the current model. The downside is that we may occasionally choose a phrase t_i that is actually a true paraphrase of s_i.

7.5 EXPERIMENTS

We now investigate how best to use our generated paraphrase data for training universal paraphrastic sentence embeddings. We consider 10 data sources: Common Crawl (CC), Europarl (EP), and News Commentary (News) from all 3 language pairs, as well as the 10^9 French-English data (Giga). We extract 150,000 reference/back-translation pairs from each data source. We use 100,000 of these to mine for training data for our sentence embedding models, and the remaining 50,000 are used as train/validation/test data for the reference classification and language models described below.

7.5.1 *Evaluation*

We evaluate the quality of a paraphrase dataset by using the experimental setting from Chapter 4. We use the paraphrases as training data to create paraphrastic sentence embeddings, then evaluate the embeddings on the SemEval semantic textual similarity (STS) tasks from 2012 to 2015 (Agirre et al., 2012, 2013, 2014, 2015), the SemEval 2015 Twitter task (Xu et al., 2015b), and the SemEval 2014 SICK Semantic Relatedness task (Marelli et al., 2014).

Given two sentences, the aim of the STS tasks is to predict their similarity on a 0-5 scale, where 0 indicates the sentences are on different topics and 5 indicates that they are completely equivalent. As our test set, we report the average Pearson's r over these 22 sentence similarity tasks. As development data, we use the 2016 STS tasks (Agirre et al., 2016), where the tuning criterion is the average Pearson's r over its 5 datasets.

7.5.2 *Experimental Setup*

For fair comparison among different datasets and dataset filtering methods described below, we use only 24,000 training examples for nearly all experiments. Different filtering methods produce different amounts of training data, and using 24,000 examples allows us to keep the amount of training data constant across filtering methods. It also allows us to complete these several thousand experiments in a reasonable amount of time. In Section 7.5.8 below, we discuss experiments that scale up to larger amounts of training data.

We use PARAGRAM-SL999 embeddings from Chapter 3 to initialize the word embedding matrix (W_w) for both models. For all experiments, we fix the mini-batch size to 100, λ_w to 0, λ_c to 0, and the margin δ to 0.4. We train WORD for 20 epochs, and the GRAN for 3, since it converges much faster. For optimization we use Adam (Kingma and Ba, 2014) with a learning rate of 0.001.

We compare to two data resources used in previous work to learn paraphrastic sentence embeddings. The first is phrase pairs from PPDB, used in Chapters 4 and 5. PPDB comes in different sizes (S, M, L, XL, XXL, and XXXL), where each larger size subsumes all smaller ones. The pairs in PPDB are sorted by a confidence measure and so the smaller sets contain higher precision paraphrases. We use PPDB XL in this paper, which consists of fairly high precision paraphrases. The other data source is the aligned Simple English / standard English Wikipedia data developed by Coster and Kauchak (2011) and used for learning paraphrastic sentence embeddings in Chapter 6. We refer to this data source as "SimpWiki". We refer to our back-translated data as "NMT".

7.5.3 *Dataset Comparison*

We first compare datasets, randomly sampling 24,000 sentence pairs from each of PPDB, SimpWiki, and each of our NMT datasets. The only hyperparameter to tune for this experiment is the stopping epoch, which we tune based on our development set. The results are shown in Table 46.

Lang.	Data	GRAN	WORD
SimpWiki		67.2	65.8
PPDB		64.5	65.8
CS	CC	65.5	65.4
	EP	66.5	65.1
	News	67.2	65.1
FR	CC	67.3	66.1
	EP	**67.8**	65.7
	Giga	67.4	65.9
	News	67.0	65.2
DE	CC	66.5	**66.2**
	EP	67.2	65.6
	News	66.5	64.7

Table 46: Test results (average Pearson's $r \times 100$ over 22 STS datasets) using a random selection of 24,000 examples from each data source considered in this paper.

We find that the NMT datasets are all effective as training data, outperforming PPDB in all cases when using the GRAN. There are exceptions when using WORD, for which PPDB is quite strong. This is sensible because WORD is not sensitive to word order, so the fragments in PPDB do not cause problems. However, when using the GRAN, which is sensitive to word order, the NMT data is consistently better than PPDB. It often exceeds the performance of training on the SimpWiki data, which consists entirely of human-written sentences.

7.5.4 *Filtering Methods*

Above we showed that the NMT data is better than PPDB when using a GRAN and often as good as SimpWiki. Since we have access to so much more NMT data than SimpWiki (which is limited to fewer than 200k sentence pairs), we next experiment with several approaches for filtering the NMT data. We first consider filtering based on length, described in Section 7.5.5. We then consider filtering based on several quality measures designed to find more natural and higher-quality translations, described in Section 7.5.6. Finally, we consider several measures of diversity. By diversity we mean here a measure of the lexical and syntactic difference between the reference and its paraphrase. We describe these experiments in Section 7.5.7. We note that these filtering methods are not all mutually exclusive and could be combined, though in this paper we experiment with each individually and leave combination to future work.

7.5.5 *Length Filtering*

We first consider filtering candidate sentence pairs by length, i.e., the number of tokens in the translation. The tunable parameters are the upper and lower bounds of the translation lengths.

We experiment with a partition of length ranges, showing the results in Table 47. These results are averages across all language pairs and data sources of training data for each length range shown. We find it best to select NMT data where the translations have between 0 and 10 tokens, with performance dropping as sentence length increases. This is true for both the GRAN and WORD models. We do the same filtering for the SimpWiki data, though the trend is not nearly as strong. This may be because machine translation quality drops as sentence length increases. This trend appears even though the datasets with higher ranges have more tokens of training data, since only the number of training sentence pairs is kept constant across configurations.

We then tune the length range using our development data, considering the following length ranges: [0,10], [0,15], [0,20], [0,30], [0,100], [10,20], [10,30], [10,100], [15,25], [15,30], [15,100], [20,30], [20,100], [30,100]. We tune over ranges as well as language, data source, and stopping epoch, each time training on 24,000 sentence pairs. We report the average test results over all languages and datasets in Table 48. We compare to a baseline that draws a random set of data, showing that length-based filtering leads to gains of nearly half a point on average across our test sets. The tuned length ranges are short for both NMT and SimpWiki.

The distribution of lengths in the NMT and SimpWiki data is fairly similar. The 10 NMT datasets all have mean translation lengths between 22 and 28 tokens, with common crawl on the lower end and Europarl on the higher end. The data has fairly large standard deviations (11-25 tokens) indicating that there are some very long translations in the data. SimpWiki has a mean length of 24.2 and a standard deviation of 13.1 tokens.

7.5.6 *Quality Filtering*

We also consider filtering based on several measures of the "quality" of the back-translation:

- **Translation Cost**: We use the cost (negative log likelihood) of the translation from the NMT system, divided by the number of tokens in the translation.

		Length Range			
Data	Model	0-10	10-20	20-30	30-100
SimpWiki	GRAN	67.4	67.7	67.1	67.3
	Word	65.9	65.7	65.6	65.9
NMT	GRAN	66.6	66.5	66.0	64.8
	Word	65.7	65.6	65.3	65.0

Table 47: Test correlations for our models when trained on sentences with particular length ranges (averaged over languages and data sources for the NMT rows). Results are on STS datasets (Pearson's $r \times 100$).

	NMT		SimpWiki	
Filtering Method	GRAN	Word	GRAN	Word
None (Random)	66.9	65.5	67.2	65.8
Length	67.3	66.0	67.4	66.2
Tuned Len. Range	[0,10]	[0,10]	[0,10]	[0,15]

Table 48: Length filtering test results after tuning length ranges on development data (averaged over languages and data sources for the NMT rows). Results are on STS datasets (Pearson's $r \times 100$).

- **Language Model**: We train a separate language model for each language/data pair on 40,000 references that are separate from the 100,000 used for mining data. Due to the small data size, we train a 3-gram language model and use the KenLM toolkit (Heafield, 2011).
- **Reference/Translation Classification**: We train binary classifiers to predict whether a given sentence is a reference or translation (described in Section 7.5.6.1). We use the probability of being a reference as the score for filtering.

For translation cost, we tune the upper bound of the cost over the range $[0.2, 1]$ using increments of 0.1. For the language model, we tune an upper bound on the perplexity of the translations among the set $\{25, 50, 75, 100, 150, 200, \infty\}$. For the classifier, we tune the minimum probability of being a reference over the range $[0, 0.9]$ using increments of 0.1.

Filtering Method	GRAN	Word
None (Random)	66.9	65.5
Translation Cost	66.6	65.4
Language Model	66.7	65.5
Reference Classification	67.0	65.5

Table 49: Quality filtering test results after tuning quality hyperparameters on development data (averaged over languages and data sources for the NMT rows). Results are on STS datasets (Pearson's $r \times 100$).

Table 49 shows average test results over all languages and datasets after tuning hyperparameters on our development data for each. The translation cost and language model are not helpful for filtering, as random selection outperforms them. Both methods are outperformed by the reference classifier, which slightly outperforms random selection when using the stronger GRAN model. We now discuss further how we trained the reference classifier and the data characteristics that it reveals. We did not experiment with quality filtering for SimpWiki since it is human-written text.

7.5.6.1 *Reference/Translation Classification*

We experiment with predicting whether a given sentence is a reference or a back-translation. We train two kinds of binary classifiers, one using an LSTM and the other using word averaging, followed by a softmax layer. We select 40,000 reference/translation pairs for training and 5,000 for each of validation and testing. A single example is a sentence with label 1 if it is a reference translation and 0 if it is a translation.

In training, we consider the entire k-best list as examples of translations, selecting one translation to be the 0-labeled example. We either do this randomly or we score each sentence in the k-best list using our model and select the one with the highest probability of being a reference as the 0-labeled example. We tune this choice as well as an L_2 regularizer on the word embeddings (tuned over $\{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 0\}$). We use PARAGRAM-SL999 embeddings (Wieting et al., 2015) to initialize the word embeddings for both models. Models were trained by minimizing cross entropy for 10 epochs using Adam with learning rate 0.001. We performed this procedure separately for each of the 10 language/data pairs.

The results are shown in Table 50. While performance varies greatly across data sources, the LSTM always outperforms the word averaging model. For our translation-reference classification, we note that our results can be further improved. We also trained models on 90,000 examples, essentially doubling the amount of data, and the results improved by about 2% absolute on each dataset on both the validation and testing data.

ANALYZING REFERENCE CLASSIFICATION. We inspected the output of our reference classifier and noted a few qualitative trends which we then verified empirically. First, neural MT systems tend to use a smaller vocabulary and exhibit more restricted use of phrases. They correspondingly tend to show more repetition in terms of both words and longer n-grams. This hypothesis can be verified empirically in several ways. We do so by calculating the entropy of the unigrams and trigrams for both the references and the translations

Model	Lang.	Data	Test Acc.	+ Acc.	- Acc.
LSTM	CS	CC	72.2	72.2	72.3
		EP	72.3	64.3	80.3
		News	79.7	73.2	86.3
	FR	CC	80.7	82.1	79.3
		EP	79.3	75.2	83.4
		Giga	**93.1**	**92.3**	93.8
		News	84.2	81.2	87.3
	DE	CC	79.3	71.7	86.9
		EP	85.1	78.0	92.2
		News	89.8	82.3	**97.4**
WORD	CS	CC	71.2	68.9	73.5
		EP	69.1	63.0	75.1
		News	77.6	71.7	83.6
	FR	CC	78.8	80.4	77.2
		EP	78.9	75.5	82.3
		Giga	**92.5**	**91.5**	93.4
		News	82.8	81.1	84.5
	DE	CC	77.3	70.4	84.1
		EP	82.7	73.4	91.9
		News	87.6	80.0	**95.3**

Table 50: Results of reference/translation classification (accuracy$\times$100). The highest score in each column is in boldface. Final two columns show accuracies of positive (reference) and negative classes, respectively.

Lang.	Data	Ent. (uni)	Ent. (tri)	Rep. (uni)	Rep. (tri)
CS	CC	0.50	1.13	-7.57%	-5.58%
	EP	0.14	0.31	-0.88%	-0.11%
	News	0.16	0.31	-0.96%	-0.16%
FR	CC	0.97	1.40	-8.50%	-7.53%
	EP	0.51	0.69	-1.85%	-0.58%
	Giga	0.97	1.21	-5.30%	-7.74%
	News	0.67	0.75	-2.98%	-0.85%
DE	CC	0.29	0.57	-1.09%	-0.73%
	EP	0.32	0.53	-0.14%	-0.11%
	News	0.40	0.37	-1.02%	-0.24%
All		0.46	0.74	-2.80%	-2.26%

Table 51: Differences in entropy and repetition of unigrams/trigrams in references and translations. Negative values indicate translations have a higher value, so references show consistently higher entropies and lower repetition rates.

from our 150,000 reference-translation pairs.[2] We also calculate the repetition percentage of unigrams and trigrams in both the references and translations. This is defined as the percentage of words that are repetitions (i.e., have already appeared in the sentence). For unigrams, we only consider words consisting of at least 3 characters.

The results are shown in Table 51, in which we subtract the translation value from the reference value for each measure. The translated text has lower n-gram entropies and higher rates of repetition. This appears for all datasets, but is strongest for common crawl and French-English 10^9.

We also noticed that translations are less likely to use rare words, instead willing to use a larger sequence of short words to convey the same meaning. We found that translations were sometimes more vague and, unsurprisingly, were more likely to be ungrammatical.

We check whether our classifier is learning these patterns by computing the reference probabilities $P(R)$ of 100,000 randomly sampled translation-reference pairs from each dataset (the same used to train models). We then compute the correlation between our classification score and different metrics: the repetition rate of the sentence, the average inverse-document frequency (IDF) of the sentence,[3] and the length of the translation.

The results are shown in Table 52. Negative correlations with repetitions indicates that fewer repetitions lead to higher $P(R)$. A positive correlation with average IDF indicates that $P(R)$ rewards the use of rare words. Interestingly, negative correlation with length suggests

2 We randomly selected translations from the beam search.
3 Wikipedia was used to calculate the frequencies of the tokens. All tokens were lowercased.

Metric	Spearman's ρ
Unigram repetition rate	-35.1
Trigram repetition rate	-18.4
Average IDF	27.8
Length	-34.0

Table 52: Spearman's ρ between our reference classifier probability and various measures.

	Sentence	P(R)
R:	Room was comfortable and the staff at the front desk were very helpful.	1.0
T:	The staff were very nice and the room was very nice and the staff were very nice.	<0.01
R:	The enchantment of your wedding day, captured in images by Flore-Ael Surun.	0.98
T:	The wedding of the wedding, put into images by Flore-Ael A.	<0.01
R:	Mexico and Sweden are longstanding supporters of the CTBT.	1.0
T:	Mexico and Sweden have been supporters of CTBT for a long time now.	0.06
R:	We thought Mr Haider ′ s Austria was endangering our freedom.	1.0
T:	We thought that our freedom was put at risk by Austria by Mr Haider.	0.09

Table 53: Illustrative examples of references (R) and back-translations (T), along with probabilities from the reference classifier. See text for details.

that the classifier prefers more concise sentences.[4] We show examples of these phenomena in Table 53. The first two examples show the tendency of NMT to repeat words and phrases. The second two show how they tend to use sequences of common words ("put at risk") rather than rare words ("endangering").

	NMT		SimpWiki	
Filtering Method	GRAN	Word	GRAN	Word
Random	66.9	65.5	67.2	65.8
Unigram Overlap	66.6	66.1	67.8	67.4
Bigram Overlap	67.0	65.5	68.0	67.2
Trigram Overlap	66.9	65.4	67.8	66.6
BLEU Score	67.1	65.3	67.5	66.5

Table 54: Diversity filtering test results after tuning filtering hyperparameters on development data (averaged over languages and data sources for the NMT rows). Results are on STS datasets (Pearson's $r \times 100$).

7.5.7 *Diversity Filtering*

We consider several filtering criteria based on measures that encourage particular amounts of disparity between the reference and its back-translation:

- **n-gram Overlap**: Our n-gram overlap measures are calculated by counting n-grams of a given order in both the reference and translation, then dividing the number of shared n-grams by the total number of n-grams in the reference or translation, whichever has fewer. We use three n-gram overlap scores ($n \in \{1, 2, 3\}$).
- **BLEU Score**: We use a smoothed sentence-level BLEU variant from Nakov et al. (2012) that uses smoothing for all n-gram lengths and also smooths the brevity penalty.

For both methods, the tunable hyperparameters are the upper and lower bounds for the above scores. We tune over the cross product of lower bounds $\{0, 0.1, 0.2, 0.3\}$ and upper bounds $\{0.6, 0.7, 0.8, 0.9, 1.0\}$. Our intuition is that the best data will have some amount of n-gram overlap, but not too much. Too much n-gram overlap will lead to pairs that are not useful for learning.

The results are shown in Table 54, for both models and for both NMT and SimpWiki. We find that the diversity filtering methods lead to consistent improvements when training on SimpWiki. We believe this is because many of the sentence pairs in SimpWiki are near-duplicates and these filtering methods favor data with more differences.

Diversity filtering can also help when selecting NMT data, though the differences are smaller. We do note that unigram overlap is the strongest filtering strategy for WORD. When looking at the threshold tuning, the best lower bounds are often 0 or 0.1 and the best upper bounds are typically 0.6-0.7, indicating that sentence pairs with a high degree of word overlap are not useful for training. We also find that the GRAN benefits more from filtering based on higher-order n-gram overlap than WORD.

7.5.8 *Scaling Up*

Unlike the SimpWiki data, which is naturally limited and only available for English, we can scale our approach. Since we use data on which the NMT systems were trained and perform back-translation, we can easily produce large training sets of paraphrastic sentence pairs for many languages and data domains, limited only by the availability of bitext.

4 This is noteworthy because the average sentence length of translations and references is not significantly different.

Data	GRAN	WORD
PPDB	64.6	66.3
SimpWiki (100k/168k)	67.4	**67.7**
CC-CS (24k)	66.8	-
CC-DE (24k)	-	66.6
CC-CS (100k)	**68.5**	-
CC-DE (168k)	-	67.6

Table 55: Test results when using more training data. More data helps both WORD and GRAN, where both models get close to or surpass training on SimpWiki and comfortably surpass the PPDB baseline. The amount of training examples used is in parentheses.

To test this, we took the tuned filtering approaches and language/data pairs (according to our development dataset only), and trained them on more data. These were the CC-CS for GRAN and CC-DE for WORD models. We trained them with the same amount of sentence pairs from SimpWiki.[5] We also compare to PPDB XL, and since PPDB has fewer tokens per example, we use enough PPDB data so that it has at least as many tokens as the SimpWiki data used in the experiment.[6]

The results are shown in Table 55, where the NMT data surpasses SimpWiki, while both SimpWiki and the NMT data perform similarly for WORD. PPDB is soundly beaten by both data sources. Moreover, the CC-CS and CC-DE data have greatly improved their performance from training on just 24,000 examples from 66.77 and 66.61 respectively, providing more evidence that this approach does indeed scale.

7.6 CONCLUSION

We have shown how back-translating can be an effective paraphrase-generation technique, by using it to produce paraphrastic embeddings at least on par with human-written paraphrase datasets and significantly surpassing PPDB. We have also shown that filtering can improve the generated paraphrase corpus, and we explored a variety of filtering techniques. In doing so, we also identified characteristics that distinguish NMT output from human-written sentences. Future work will aim to improve this generation process, for instance by devising more sophisticated filtering techniques.

5 Since the CC-CS data was the smallest dataset used to train the CS NMT system (See Table 44) and limited, we only used 100,000 pairs for the GRAN experiment. For WORD, we used the full 167,689.
6 This was 800,011 and 1,341,188 pairs in the GRAN and WORD respectively.

8 PARANMT: A PARAPHRASE CORPUS

8.1 INTRODUCTION

While many approaches have been developed for generating or finding paraphrases (Barzilay and McKeown, 2001; Lin and Pantel, 2001; Dolan et al., 2004), there do not exist any freely-available datasets with millions of sentential paraphrase pairs. The closest such resource is the Paraphrase Database (PPDB; Ganitkevitch et al., 2013), which was created automatically from bilingual text by pivoting over the non-English language (Bannard and Callison-Burch, 2005). PPDB has been used to improve word embeddings (Faruqui et al., 2015; Mrkšić et al., 2016). However, PPDB is less useful for learning *sentence* embeddings (Wieting and Gimpel, 2017).

In this chapter, we describe the creation of a dataset containing more than 50 million sentential paraphrase pairs. We create it automatically by scaling up the approach shown in Chapter 7. We use neural machine translation (NMT) to translate the Czech side of a large Czech-English parallel corpus. We pair the English translations with the English references to form paraphrase pairs. We call this dataset ParaNMT-50M. It contains examples illustrating a broad range of paraphrase phenomena; we show examples in Section 12.2. ParaNMT-50Mhas the potential to be useful for many tasks, from linguistically controlled paraphrase generation, style transfer, and sentence simplification to core NLP problems like machine translation.

We show the utility of ParaNMT-50Mby using it to train paraphrastic sentence embeddings using the learning framework we showed in Chapter 4. We primarily evaluate our sentence embeddings on the SemEval semantic textual similarity (STS) competitions from 2012-2016. Since so many domains are covered in these datasets, they form a demanding evaluation for a general purpose sentence embedding model.

Our sentence embeddings learned from ParaNMT-50Moutperform all systems in every STS competition from 2012 to 2016. These tasks have drawn substantial participation; in 2016, for example, the competition attracted 43 teams and had 119 submissions. Most STS systems use curated lexical resources, the provided supervised training data with manually-annotated similarities, and joint modeling of the sentence pair. We use none of these, simply encoding each sentence independently using our models and computing cosine similarity between their embeddings.

We experiment with several compositional architectures and find them all to work well. We find benefit from making a simple change to learning ("mega-batching") to better leverage the large training set, namely, increasing the search space of negative examples. In the supplementary, we evaluate on general-purpose sentence embedding tasks used in past work (Kiros et al., 2015; Conneau et al., 2017), finding our embeddings to perform competitively.

Finally, in Section 8.7, we briefly report results showing how ParaNMT-50Mcan be used for paraphrase generation. A standard encoder-decoder model trained on ParaNMT-50Mcan generate paraphrases that show effects of "canonicalizing" the input sentence. In other work, fully described in Chapter 12, we used ParaNMT-50Mto generate paraphrases that have a specific syntactic structure (represented as the top two levels of a linearized parse tree).

We release the ParaNMT-50Mdataset, our trained sentence embeddings, and our code. ParaNMT-50Mis the largest collection of sentential paraphrases released to date. We hope it can motivate new research directions and be used to create powerful NLP models, while adding a robustness to existing ones by incorporating paraphrase knowledge. Our paraphrastic sentence embeddings are state-of-the-art by a significant margin, and we hope they can be useful for many applications both as a sentence representation function and as a general similarity metric.

8.2 related work

We discuss work in automatically building paraphrase corpora, learning general-purpose sentence embeddings, and using parallel text for learning embeddings and similarity functions.

paraphrase discovery and generation. Many methods have been developed for generating or finding paraphrases, including using multiple translations of the same source material (Barzilay and McKeown, 2001), using distributional similarity to find similar dependency paths (Lin and Pantel, 2001), using comparable articles from multiple news sources (Dolan et al., 2004; Dolan and Brockett, 2005; Quirk et al., 2004), aligning sentences between standard and Simple English Wikipedia (Coster and Kauchak, 2011), crowdsourcing (Xu et al., 2014, 2015b; Jiang et al., 2017), using diverse MT systems to translate a single source sentence (Suzuki et al., 2017), and using tweets with matching URLs (Lan et al., 2017a).

The most relevant prior work uses bilingual corpora. Bannard and Callison-Burch (2005) used methods from statistical machine translation to find lexical and phrasal paraphrases in parallel text. Ganitkevitch et al. (2013) scaled up these techniques to produce the Paraphrase Database (PPDB). Our goals are similar to those of PPDB, which has likewise been generated for many languages (Ganitkevitch and Callison-Burch, 2014) since it only needs parallel text. In particular, we follow the approach from Chapter 7, where we used NMT to translate the non-English side of parallel text to get English-English paraphrase pairs. We scale up the method to a larger dataset, produce state-of-the-art paraphrastic sentence embeddings, and release all of our resources.

SENTENCE EMBEDDINGS. Our learning and evaluation setting is the same as that of our recent work that seeks to learn paraphrastic sentence embeddings that can be used for downstream tasks (see Chapters 4 to 7. We trained models on noisy paraphrase pairs and evaluated them primarily on semantic textual similarity (STS) tasks. Prior work in learning general sentence embeddings has used autoencoders (Socher et al., 2011; Hill et al., 2016), encoder-decoder architectures (Kiros et al., 2015; Gan et al., 2017), and other sources of supervision and learning frameworks (Le and Mikolov, 2014; Pham et al., 2015; Arora et al., 2017; Pagliardini et al., 2017; Conneau et al., 2017).

PARALLEL TEXT FOR LEARNING EMBEDDINGS. Prior work has shown that parallel text, and resources built from parallel text like NMT systems and PPDB, can be used for learning embeddings for words and sentences. Several have used PPDB as a knowledge resource for training or improving embeddings (Faruqui et al., 2015; Wieting et al., 2015; Mrkšić et al., 2016). NMT architectures and training settings have been used to obtain better embeddings for words (Hill et al., 2014a,b) and words-in-context (McCann et al., 2017). Hill et al. (2016) evaluated the encoders of English-to-X NMT systems as sentence representations. Mallinson et al. (2017) adapted trained NMT models to produce sentence similarity scores in semantic evaluations.

Dataset	Avg. Length	Avg. IDF	Avg. Para. Score	Vocab. Entropy	Parse Entropy	Total Size
Common Crawl	$24.0_{\pm 34.7}$	$7.7_{\pm 1.1}$	$0.83_{\pm 0.16}$	7.2	3.5	0.16M
CzEng 1.6	$13.3_{\pm 19.3}$	$7.4_{\pm 1.2}$	$0.84_{\pm 0.16}$	6.8	4.1	51.4M
Europarl	$26.1_{\pm 15.4}$	$7.1_{\pm 0.6}$	$0.95_{\pm 0.05}$	6.4	3.0	0.65M
News Commentary	$25.2_{\pm 13.9}$	$7.5_{\pm 1.1}$	$0.92_{\pm 0.12}$	7.0	3.4	0.19M

Table 56: Statistics of 100K-samples of Czech-English parallel corpora; standard deviations are shown for averages.

Reference Translation	Machine Translation
so, what's half an hour?	half an hour won't kill you.
well, don't worry. i've taken out tons and tons of guys. lots of guys.	don't worry, i've done it to dozens of men.
it's gonna be ...... classic.	yeah, sure. it's gonna be great.
greetings, all!	hello everyone!
but she doesn't have much of a case.	but as far as the case goes, she doesn't have m
it was good in spite of the taste.	despite the flavor, it felt good.

Table 57: Example paraphrase pairs from PARANMT-50M, where each consists of an English reference translation and the machine translation of the Czech source sentence (not shown).

8.3 THE PARANMT-50MDATASET

To create our dataset, we used back-translation of bitext (Chapter 7). We used a Czech-English NMT system to translate Czech sentences from the training data into English. We paired the translations with the English references to form English-English paraphrase pairs.

We used the pretrained Czech-English model from the NMT system of Sennrich et al. (2017). Its training data includes four sources: Common Crawl, CzEng 1.6 (Bojar et al., 2016), Europarl, and News Commentary. We did not choose Czech due to any particular linguistic properties. In Chapter 7, we found little difference among Czech, German, and French as source languages for back-translation. There were much larger differences due to data domain, so we focus on the question of domain in this section. We leave the question of investigating properties of back-translation of different languages to future work.

8.3.1 *Choosing a Data Source*

To assess characteristics that yield useful data, we randomly sampled 100K English reference translations from each data source and computed statistics. Table 56 shows the average sentence length, the average inverse document frequency (IDF) where IDFs are computed using Wikipedia sentences, and the average paraphrase score for the two sentences. The paraphrase score is calculated by averaging PARAGRAM-PHRASE embeddings from Chapter 4 for the two sentences in each pair and then computing their cosine similarity. The table also shows the entropies of the vocabularies and constituent parses obtained using the Stanford Parser (Manning et al., 2014).[1]

Europarl exhibits the least diversity in terms of rare word usage, vocabulary entropy, and parse entropy. This is unsurprising given its formulaic and repetitive nature. CzEng has shorter sentences than the other corpora and more diverse sentence structures, as shown by its high parse entropy. In terms of vocabulary use, CzEng is not particularly more diverse than Common Crawl and News Commentary, though this could be due to the prevalence of named entities in the latter two.

In Section 8.5.3, we empirically compare these data sources as training data for sentence embeddings. The CzEng corpus yields the strongest performance when controlling for training data size. Since its sentences are short, we suspect this helps ensure high-quality back-translations. A large portion of it is movie subtitles which tend to use a wide vocabulary and have a diversity of sentence structures; however, other domains are included as well. It is also the largest corpus, containing over 51 million sentence pairs. In addition to providing a large number of training examples for downstream tasks, this means that the NMT system should be able to produce quality translations for this subset of its training data.

For all of these reasons, we chose the CzEng corpus to create ParaNMT-50M. When doing so, we used beam search with a beam size of 12 and selected the highest scoring translation from the beam. It took over 10,000 GPU hours to back-translate the CzEng corpus. We show illustrative examples in Table 57.

8.3.2 *Manual Evaluation*

We conducted a manual analysis of our dataset in order to quantify its noise level and assess how the noise can be ameliorated with filtering. Two native English speakers annotated a sample of 100 examples from each of five ranges of the Paraphrase Score.[2] We obtained annotations for both the strength of the paraphrase relationship and the fluency of the translations.

1 To mitigate sparsity in the parse entropy, we used only the top two levels of each parse tree.

2 Even though the similarity score lies in $[-1, 1]$, most observed scores were positive, so we chose the five ranges shown in Table 58.

Para. Score	#	Avg. Tri.	Paraphrase			Fluency		
Range	(M)	Overlap	1	2	3	1	2	3
(-0.1, 0.2]	4.0	$0.00_{\pm 0.0}$	92	6	2	1	5	94
(0.2, 0.4]	3.8	$0.02_{\pm 0.1}$	53	32	15	1	12	87
(0.4, 0.6]	6.9	$0.07_{\pm 0.1}$	22	45	33	2	9	89
(0.6, 0.8]	14.4	$0.17_{\pm 0.2}$	1	43	56	11	0	89
(0.8, 1.0]	18.0	$0.35_{\pm 0.2}$	1	13	86	3	0	97

Table 58: Manual evaluation of PARANMT-50M. 100-pair samples were drawn from five ranges of the automatic paraphrase score (first column). Paraphrase strength and fluency were judged on a 1-3 scale and counts of each rating are shown.

To annotate paraphrase strength, we adopted the annotation guidelines used by Agirre et al. (2012). The original guidelines specify six classes, which we reduced to three for simplicity. We combined the top two into one category, left the next, and combined the bottom three into the lowest category. Therefore, for a sentence pair to have a rating of 3, the sentences must have the same meaning, but some unimportant details can differ. To have a rating of 2, the sentences are roughly equivalent, with some important information missing or that differs slightly. For a rating of 1, the sentences are not equivalent, even if they share minor details.

For fluency of the back-translation, we use the following: A rating of 3 means it has no grammatical errors, 2 means it has one to two errors, and 1 means it has more than two grammatical errors or is not a natural English sentence.

Table 58 summarizes the annotations. For each score range, we report the number of pairs, the mean trigram overlap score, and the number of times each paraphrase/fluency label was present in the sample of 100 pairs. There is noise but it is largely confined to the bottom two ranges which together comprise only 16% of the entire dataset. In the highest paraphrase score range, 86% of the pairs possess a strong paraphrase relationship. The annotations suggest that PARANMT-50Mcontains approximately 30 million strong paraphrase pairs, and that the paraphrase score is a good indicator of quality. At the low ranges, we inspected the data and found there to be many errors in the sentence alignment in the original bitext. With regards to fluency, approximately 90% of the back-translations are fluent, even at the low end of the paraphrase score range. We do see an outlier at the second-highest range of the paraphrase score, but this may be due to the small number of annotated examples.

8.4 LEARNING SENTENCE EMBEDDINGS

To show the usefulness of the ParaNMT-50Mdataset, we will use it to train sentence embeddings. We adopt the learning framework from Chapter 4, which was developed to train sentence embeddings from pairs in PPDB. We first describe the compositional sentence embedding models we will experiment with, then discuss training and our modification ("mega-batching").

MODELS. We want to embed a word sequence s into a fixed-length vector. We denote the tth word in s as s_t, and we denote its word embedding by x_t. We focus on three model families, though we also experiment with combining them in various ways. The first, which we call Word, simply averages the embeddings x_t of all words in s. This model was found in Chapter 4 to perform strongly for semantic similarity tasks.

The second is similar to Word, but instead of word embeddings, we average character trigram embeddings (Huang et al., 2013). We call this Trigram. In Chapter 5, we found this to work well for sentence embeddings compared to other n-gram orders and to word averaging.

The third family includes long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997). We average the hidden states to produce the final sentence embedding. For regularization during training, we scramble words with a small probability (from Chapter 6). We also experiment with bidirectional LSTMs (BLSTM), averaging the forward and backward hidden states with no concatenation.[3]

TRAINING. The training data is a set S of paraphrase pairs $\langle s, s' \rangle$ and we minimize a margin-based loss $\ell(s, s') =$

$$\max(0, \delta - \cos(g(s), g(s')) + \cos(g(s), g(t)))$$

where g is the model (Word, Trigram, etc.), δ is the margin, and t is a "negative example" taken from a mini-batch during optimization. The intuition is that we want the two texts

3 Unlike Conneau et al. (2017), we found this to outperform max-pooling for both semantic similarity and general sentence embedding tasks.

to be more similar to each other than to their negative examples. To select t we choose the most similar sentence in some set. For simplicity we use the mini-batch for this set, i.e.,

$$t = \operatorname*{argmax}_{t':\langle t',\cdot\rangle \in S_b \setminus \{\langle s,s'\rangle\}} \cos(g(s), g(t'))$$

where $S_b \subseteq S$ is the current mini-batch.

MODIFICATION: MEGA-BATCHING. By using the mini-batch to select negative examples, we may be limiting the learning procedure. That is, if all potential negative examples in the mini-batch are highly dissimilar from s, the loss will be too easy to minimize. Stronger negative examples can be obtained by using larger mini-batches, but large mini-batches are sub-optimal for optimization.

Therefore, we propose a procedure we call "mega-batching." We aggregate M mini-batches to create one mega-batch and select negative examples from the mega-batch. Once each pair in the mega-batch has a negative example, the mega-batch is split back up into M mini-batches and training proceeds. We found that this provides more challenging negative examples during learning as shown in Section 8.5.5. Table 61 shows results for different values of M, showing consistently higher correlations with larger M values.

8.5 EXPERIMENTS

We now investigate how best to use our generated paraphrase data for training paraphrastic sentence embeddings.

8.5.1 *Evaluation*

We evaluate sentence embeddings using the SemEval semantic textual similarity (STS) tasks from 2012 to 2016 (Agirre et al., 2012, 2013, 2014, 2015, 2016) and the STS Benchmark (Cer et al., 2017). Given two sentences, the aim of the STS tasks is to predict their similarity on a 0-5 scale, where 0 indicates the sentences are on different topics and 5 means they are completely equivalent. As our test set, we report the average Pearson's r over each year of the STS tasks from 2012-2016. We use the small (250-example) English dataset from SemEval 2017 (Cer et al., 2017) as a development set, which we call STS2017 below.

Training Corpus	Word	Trigram	LSTMAvg
Common Crawl	80.9	80.2	79.1
CzEng 1.6	**83.6**	**81.5**	**82.5**
Europarl	78.9	78.0	80.4
News Commentary	80.2	78.2	80.5

Table 59: Pearson's $r \times 100$ on STS2017 when training on 100k pairs from each back-translated parallel corpus. CzEng works best for all models.

The supplementary material contains a description of a method to obtain a paraphrase lexicon from ParaNMT-50Mthat is on par with that provided by PPDB 2.0. We also evaluate our sentence embeddings on a range of additional tasks that have previously been used for evaluating sentence representations (Kiros et al., 2015).

8.5.2 *Experimental Setup*

For training sentence embeddings on ParaNMT-50M, we follow the experimental procedure from Chapter 4. We use paragram-sl999 embeddings from Chapter 3 to initialize the word embedding matrix for all models that use word embeddings. We fix the mini-batch size to 100 and the margin δ to 0.4. We train all models for 5 epochs. For optimization we use Adam (Kingma and Ba, 2014) with a learning rate of 0.001. For the LSTM and BLSTM, we fixed the scrambling rate to 0.3.[4]

8.5.3 *Dataset Comparison*

We first compare parallel data sources. We evaluate the quality of a data source by using its back-translations paired with its English references as training data for paraphrastic sentence embeddings. We compare the four data sources described in Section 12.2. We use 100K samples from each corpus and trained 3 different models on each: Word, Trigram, and LSTMAvg. Table 59 shows that CzEng provides the best training data for all models, so we used it to create ParaNMT-50Mand for all remaining experiments.

CzEng is diverse in terms of both vocabulary and sentence structure. It has significantly shorter sentences than the other corpora, and has much more training data, so its translations are expected to be better than those in the other corpora. In Chapter 7, we found that

4 As in Chapter 6, we found that scrambling significantly improves results, even with our much larger training set. But while we previously used a scrambling rate of 0.5, we found that a smaller rate of 0.3 worked better when training on ParaNMT-50M, presumably due to the larger training set.

Filtering Method	Model Avg.
Translation Score	83.2
Trigram Overlap	83.1
Paraphrase Score	**83.3**

Table 60: Pearson's $r \times 100$ on STS2017 for the best training fold across the average of WORD, TRIGRAM, and LSTMAVG models for each filtering method.

sentence length was the most important factor in filtering quality training data, presumably due to how NMT quality deteriorates with longer sentences. We suspect that better translations yield better data for training sentence embeddings.

8.5.4 *Data Filtering*

Since the PARANMT-50Mdataset is so large, it is computationally demanding to train sentence embeddings on it in its entirety. So, we filter the data to create a training set for sentence embeddings.

We experiment with three simple methods: (1) the length-normalized translation score from decoding, (2) trigram overlap (from Chapter 7), and (3) the paraphrase score from Section 12.2. Trigram overlap is calculated by counting trigrams in the reference and translation, then dividing the number of shared trigrams by the total number in the reference or translation, whichever has fewer.

We filtered the back-translated CzEng data using these three strategies. We ranked all 51M+ paraphrase pairs in the dataset by the filtering measure under consideration and then split the data into tenths (so the first tenth contains the bottom 10% under the filtering criterion, the second contains those in the bottom 10-20%, etc.).

We trained WORD, TRIGRAM, and LSTMAVG models for a single epoch on 1M examples sampled from each of the ten folds for each filtering criterion. We averaged the correlation on the STS2017 data across models for each fold. Table 60 shows the results of the filtering methods. Filtering based on the paraphrase score produces the best data for training sentence embeddings.

We randomly selected 5M examples from the top two scoring folds using paraphrase score filtering, ensuring that we only selected examples in which both sentences have a maximum length of 30 tokens.[5] These resulting 5M examples form the training data for the rest of our experiments. Note that many more than 5M pairs from the dataset are

5 In Chapter 7, we found that sentence length cutoffs were effective for filtering back-translated parallel text.

M	WORD	TRIGRAM	LSTMAVG
1	82.3	81.5	81.5
20	84.0	83.1	84.6
40	**84.1**	**83.4**	**85.0**

Table 61: Pearson's $r \times 100$ on STS2017 with different mega-batch sizes M.

original	sir, i'm just trying to protect.
negative examples:	
$M=1$	i mean, colonel...
$M=20$	i only ask that the baby be safe.
$M=40$	just trying to survive. on instinct.
original	i'm looking at him, you know?
$M=1$	they know that i've been looking for her.
$M=20$	i'm keeping him.
$M=40$	i looked at him with wonder.
original	i'il let it go a couple of rounds.
$M=1$	sometimes the ball doesn't go down.
$M=20$	i'll take two.
$M=40$	i want you to sit out a couple of rounds, all right?

Table 62: Negative examples for various mega-batch sizes M with the BLSTM model.

useful, as suggested by our human evaluations in Section 8.3.2. We have experimented with doubling the training data when training our best sentence similarity model and found the correlation increased by more than half a percentage point on average across all datasets.

8.5.5 *Effect of Mega-Batching*

Table 61 shows the impact of varying the mega-batch size M when training for 5 epochs on our 5M-example training set. For all models, larger mega-batches improve performance. There is a smaller gain when moving from 20 to 40, but all models show clear gains over $M = 1$.

Table 62 shows negative examples with different mega-batch sizes M. We use the BLSTM model and show the negative examples (nearest neighbors from the mega-batch excluding the current training example) for three sentences. Using larger mega-batches improves performance, presumably by producing more compelling negative examples for the learning procedure. This is likely more important when training on sentences than prior work on learning from text snippets (Wieting et al., 2015, 2016b; Pham et al., 2015).

	Training Data	Model	Dim.	2012	2013	2014	2015	2016
Our Work	ParaNMT	Word	300	66.2	61.8	76.2	79.3	77.5
		Trigram	300	67.2	60.3	76.1	79.7	**78.3**
		LSTMAvg	300	67.0	62.3	76.3	78.5	76.0
		LSTMAvg	900	**68.0**	60.4	76.3	78.8	75.9
		BLSTM	900	67.4	60.2	76.1	79.5	76.5
		Word + Trigram (addition)	300	67.3	**62.8**	**77.5**	80.1	78.2
		Word + Trigram + LSTMAvg (addition)	300	67.1	**62.8**	76.8	79.2	77.0
		Word, Trigram (concatenation)	600	67.8	62.7	77.4	**80.3**	78.1
		Word, Trigram, LSTMAvg (concatenation)	900	67.7	**62.8**	76.9	79.8	76.8
	SimpWiki	Word, Trigram (concatenation)	600	61.8	58.4	74.4	77.0	74.0
STS Competitions		1^{st} Place System	-	64.8	62.0	74.3	79.0	77.7
		2^{nd} Place System	-	63.4	59.1	74.2	78.0	75.7
		3^{rd} Place System	-	64.1	58.3	74.3	77.8	75.7
Related Work		InferSent (AllSNLI) (Conneau et al., 2017)	4096	58.6	51.5	67.8	68.3	67.2
		InferSent (SNLI) (Conneau et al., 2017)	4096	57.1	50.4	66.2	65.2	63.5
		FastSent (Hill et al., 2016)	100	-	-	63	-	-
		DictRep (Hill et al., 2016)	500	-	-	67	-	-
		SkipThought (Kiros et al., 2015)	4800	-	-	29	-	-
		CPHRASE (Pham et al., 2015)	-	-	-	65	-	-
		CBOW (from Hill et al., 2016)	500	-	-	64	-	-
		BLEU (Papineni et al., 2002)	-	39.2	29.5	42.8	49.8	47.4
		METEOR (Denkowski and Lavie, 2014)	-	53.4	47.6	63.7	68.8	61.8

Table 63: Pearson's $r \times 100$ on the STS tasks of our models and those from related work. We compare to the top performing systems from each SemEval STS competition. Note that we are reporting the mean correlations over domains for each year rather than weighted means as used in the competitions. Our best performing overall model (Word, Trigram) is in bold.

8.5.6 *Model Comparison*

Table 63 shows results on the 2012-2016 STS tasks and Table 64 shows results on the STS Benchmark.[6] Our best models outperform all STS competition systems and all related work of which we are aware on the 2012-2016 STS datasets. Note that the large improvement over BLEU and METEOR suggests that our embeddings could be useful for evaluating machine translation output.

Overall, our individual models (Word, Trigram, LSTMAvg) perform similarly. Using 300 dimensions appears to be sufficient; increasing dimensionality does not necessarily improve correlation. When examining particular STS tasks, we found that our individual models showed marked differences on certain tasks. Table 65 shows the mean absolute difference in Pearson's r over all 25 datasets. The Trigram model shows the largest differences from

6 Baseline results are from `http://ixa2.si.ehu.es/stswiki/index.php/STSbenchmark`, except for the unsupervised InferSent result which we computed.

	Dim.	Corr.
Our Work (Unsupervised)		
WORD	300	79.2
TRIGRAM	300	79.1
LSTMAVG	300	78.4
WORD + TRIGRAM (addition)	300	79.9
WORD + TRIGRAM + LSTMAVG (addition)	300	79.6
Word, Trigram (concatenation)	600	79.9
WORD, TRIGRAM, LSTMAVG (concatenation)	900	79.2
Related Work (Unsupervised)		
InferSent (AllSNLI) (Conneau et al., 2017)	4096	70.6
C-PHRASE (Pham et al., 2015)		63.9
GloVe (Pennington et al., 2014)	300	40.6
word2vec (Mikolov et al., 2013b)	300	56.5
sent2vec (Pagliardini et al., 2017)	700	75.5
Related Work (Supervised)		
Dep. Tree LSTM (Tai et al., 2015)		71.2
Const. Tree LSTM (Tai et al., 2015)		71.9
CNN (Shao, 2017)		78.4

Table 64: Results on STS Benchmark test set.

Models	Mean Pearson Abs. Diff.
WORD / TRIGRAM	2.75
WORD / LSTMAVG	2.17
TRIGRAM / LSTMAVG	2.89

Table 65: The means (over all 25 STS competition datasets) of the absolute differences in Pearson's r between each pair of models.

Target Syntax	Paraphrase
original	with the help of captain picard, the borg will be prepared for everything.
`(SBARQ(ADVP)(,)(S)(,)(SQ))`	now, the borg will be prepared by picard, will it?
`(S(NP)(ADVP)(VP))`	the borg here will be prepared for everything.
original	you seem to be an excellent burglar when the time comes.
`(S(SBAR)(,)(NP)(VP))`	when the time comes, you'll be a great thief.
`(S(“)(UCP)(”)(NP)(VP))`	“you seem to be a great burglar, when the time comes.” you said.

Table 66: Syntactically controlled paraphrases generated by the SCPN trained on PARANMT-50M.

the other two, both of which use word embeddings. This suggests that TRIGRAM may be able to complement the other two by providing information about words that are unknown to models that rely on word embeddings.

We experiment with two ways of combining models. The first is to define additive architectures that form the embedding for a sentence by adding the embeddings computed by two (or more) individual models. All parameters are trained jointly just like when we train individual models; that is, we do not first train two simple models and add their embeddings. The second way is to define concatenative architectures that form a sentence embedding by concatenating the embeddings computed by individual models, and again to train all parameters jointly.

In Table 63 and Table 64, these combinations show consistent improvement over the individual models as well as the larger LSTMAvg and BLSTM. Concatenating Word and Trigram results in the best performance on average across STS tasks, outperforming the best supervised systems from each year. We have released the pretrained model for these "Word, Trigram" embeddings. In addition to providing a strong baseline for future STS tasks, these embeddings offer the advantages of being extremely efficient to compute and being robust to unknown words.

We show the usefulness of ParaNMT by also reporting the results of training the "Word, Trigram" model on SimpWiki, a dataset of aligned sentences from Simple English and standard English Wikipedia (Coster and Kauchak, 2011). We have shown SimpWiki to be useful for training sentence embeddings in Chapter 6. However, Table 63 shows that training on ParaNMT leads to gains in correlation of 3 to 6 points compared to SimpWiki.

8.5.7 *General-Purpose Sentence Embedding Evaluations*

We evaluate our sentence embeddings on a range of tasks that have previously been used for evaluating sentence representations (Kiros et al., 2015). These include sentiment analysis (MR, Pang and Lee, 2005; CR, Hu and Liu, 2004; SST, Socher et al., 2013), subjectivity classification (SUBJ; Pang and Lee, 2004), opinion polarity (MPQA; Wiebe et al., 2005), question classification (TREC; Li and Roth, 2002), paraphrase detection (MRPC; Dolan et al., 2004), semantic relatedness (SICK-R; Marelli et al., 2014), and textual entailment (SICK-E). We use the SentEval package from Conneau et al. (2017) to train models on our fixed sentence embeddings for each task.[7]

Table 67 shows results on the general sentence embedding tasks. Each of our individual models produces 300-dimensional sentence embeddings, which is far fewer than the several thousands (often 2400-4800) of dimensions used in most prior work. While using higher dimensionality does not improve correlation on the STS tasks, it does help on the general sentence embedding tasks. Using higher dimensionality leads to more trainable parameters in the subsequent classifiers, increasing their ability to linearly separate the data. For a

7 github.com/facebookresearch/SentEval

Model	Dim.	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	SICK-R	SICK-E
Unsupervised (Unordered Sentences)										
Unigram-TFIDF (Hill et al., 2016)		73.7	79.2	90.3	82.4	-	85.0	73.6/81.7	-	-
SDAE (Hill et al., 2016)	2400	74.6	78.0	90.8	86.9	-	78.4	73.7/80.7	-	-
Unsupervised (Ordered Sentences)										
FastSent (Hill et al., 2016)	100	70.8	78.4	88.7	80.6	-	76.8	72.2/80.3	-	-
FastSent+AE (Hill et al., 2016)		71.8	76.7	88.8	81.5	-	80.4	71.2/79.1	-	-
SkipThought (Kiros et al., 2015)	4800	76.5	80.1	93.6	87.1	82.0	92.2	73.0/82.0	85.8	82.3
Unsupervised (Structured Resources)										
DictRep (Hill et al., 2016)	500	76.7	78.7	90.7	87.2	-	81.0	68.4/76.8	-	-
NMT En-to-Fr (Hill et al., 2016)	2400	64.7	70.1	84.9	81.5	-	82.8	-		
BYTE mLSTM (Radford et al., 2017)	4096	**86.9**	**91.4**	94.6	88.5	-	-	75.0/82.8	79.2	-
Individual Models (Our Work)										
Word	300	75.8	80.5	89.2	87.1	80.0	80.1	68.6/80.9	83.6	80.6
Trigram	300	68.8	75.5	83.6	82.3	73.6	73.0	71.4/82.0	79.3	78.0
LSTMAvg	300	73.8	78.4	88.5	86.5	80.6	76.8	73.6/82.3	83.9	81.9
LSTMAvg	900	75.8	81.7	90.5	87.4	81.6	84.4	74.7/83.0	86.0	83.0
BLSTM	900	75.6	82.4	90.6	87.7	81.3	87.4	75.0/82.9	85.8	84.4
Mixed Models (Our Work)										
Word + Trigram (addition)	300	74.8	78.8	88.5	87.4	78.7	79.0	71.4/81.4	83.2	80.6
Word + Trigram + LSTMAvg (addition)	300	75.0	80.7	88.6	86.6	77.9	78.6	72.7/80.8	83.6	81.8
Word, Trigram (concatenation)	600	75.8	80.5	89.9	87.8	79.7	82.4	70.7/81.7	84.6	82.0
Word, Trigram, LSTMAvg (concatenation)	900	77.6	81.4	91.4	88.2	82.0	85.4	74.0/81.5	85.4	83.8
BLSTM (Avg., concatenation)	4096	77.5	82.6	91.0	89.3	82.8	86.8	75.8/82.6	85.9	83.8
BLSTM (Max, concatenation)	4096	76.6	83.4	90.9	88.5	82.0	87.2	76.6/83.5	85.3	82.5
Supervised (Transfer)										
InferSent (SST) (Conneau et al., 2017)	4096	-	83.7	90.2	89.5	-	86.0	72.7/80.9	86.3	83.1
InferSent (SNLI) (Conneau et al., 2017)	4096	79.9	84.6	92.1	89.8	83.3	88.7	75.1/82.3	**88.5**	**86.3**
InferSent (AllNLI) (Conneau et al., 2017)	4096	81.1	86.3	92.4	90.2	84.6	88.2	76.2/83.1	88.4	**86.3**
Supervised (Direct)										
Naive Bayes - SVM		79.4	81.8	93.2	86.3	83.1	-	-	-	-
AdaSent (Zhao et al., 2015)		83.1	86.3	**95.5**	**93.3**	-	92.4	-	-	-
BLSTM-2DCNN (Zhou et al., 2016)		82.3	-	94.0	-	**89.5**	**96.1**	-	-	-
TF-KLD (Ji and Eisenstein, 2013)		-	-	-	-	-	-	**80.4/85.9**	-	-
Illinois-LH (Lai and Hockenmaier, 2014)		-	-	-	-	-	-	-	-	84.5
Dependency Tree-LSTM (Tai et al., 2015)		-	-	-	-	-	-	-	86.8	-

Table 67: General-purpose sentence embedding tasks, divided into categories based on resource requirements.

further discussion on the effect of dimensionality and issues with the reported performance of some of these models, see (Wieting and Kiela, 2019).

To enlarge the dimensionality, we concatenate the forward and backward states prior to averaging. This is similar to Conneau et al. (2017), though they used max pooling. We experimented with both averaging ("BLSTM (Avg., concatenation)") and max pooling ("BLSTM (Max, concatenation)") using recurrent networks with 2048-dimensional hidden states, so concatenating them yields a 4096-dimension embedding. These high-dimensional models outperform SkipThought (Kiros et al., 2015) on all tasks except SUBJ and TREC. Nonetheless, the InferSent (Conneau et al., 2017) embeddings trained on AllNLI still outperform our embeddings on nearly all of these general-purpose tasks.

We also note that on five tasks (SUBJ, MPQA, SST, TREC, and MRPC), all sentence embedding methods are outperformed by supervised baselines. These baselines use the same amount of supervision as the general sentence embedding methods; the latter actually use far more data overall than the supervised baselines. This suggests that the pretrained sentence representations are not capturing the features learned by the models engineered for those tasks.

We take a closer look of how our embeddings compare to InferSent (Conneau et al., 2017). InferSent is a supervised model trained on a large textual entailment dataset (the SNLI and MultiNLI corpora (Bowman et al., 2015; Williams et al., 2017), which consist of nearly 1 million human-labeled examples).

While InferSent has strong performance across all downstream tasks, our model obtains better results on semantic similarity tasks. It consistently reach correlations approximately 10 points higher than those of InferSent.

Regarding the general-purpose tasks, we note that some result trends appear to be influenced by the domain of the data. InferSent is trained on a dataset of mostly captions, especially the model trained on just SNLI. Therefore, the datasets for the SICK relatedness and entailment evaluations are similar in domain to the training data of InferSent. Further, the training task of natural language inference is aligned to the SICK entailment task. Our results on MRPC and entailment are significantly better than SkipThought, and on a paraphrase task that does not consist of caption data (MRPC), our embeddings are competitive with InferSent. To quantify these domain effects, we performed additional experiments that are described in Section 8.5.8.

There are many ways to train sentence embeddings, each with its own strengths. InferSent, our models, and the BYTE mLSTM of Radford et al. (2017) each excel in particular classes of downstream tasks. Ours are specialized for semantic similarity. BYTE mLSTM is trained on review data and therefore is best at the MR and CR tasks. Since the InferSent models are trained using entailment supervision and on caption data, they excel on the SICK tasks. Future work will be needed to combine multiple supervision signals to generate embeddings that perform well across all tasks.

8.5.8 *Effect of Training Domain on InferSent*

We performed additional experiments to investigate the impact of training domain on downstream tasks. We first compare the performance of our "Word, Trigram (concate-

Data	AllNLI	SNLI
Overall mean diff.	10.5	12.5
MSRvid (2012) diff.	5.2	4.6
Images (2014) diff.	6.4	4.8
Images (2015) diff.	3.6	3.0

Table 68: Difference in correlation (Pearson's $r \times 100$) between "WORD, TRIGRAM" and InferSent models trained on two different datasets: AllNLI and SNLI. The first row is the mean difference across all 25 datasets, then the following rows show differences on three individual datasets that are comprised of captions. The InferSent models are much closer to our model on the caption datasets than overall.

Model	All	Cap.	No Cap.
Unsupervised			
InferSent (AllNLI)	70.6	83.0	56.6
InferSent (SNLI)	67.3	83.4	51.7
WORD, TRIGRAM	79.9	87.1	71.7
Supervised			
InferSent (AllNLI)	75.9	85.4	64.8
InferSent (SNLI)	75.9	86.4	63.1

Table 69: STS Benchmark results (Pearson's $r \times 100$) comparing our WORD, TRIGRAM model to InferSent trained on AllNLI and SNLI. We report results using all of the data (All), only the caption portion of the data (Cap.), and all of the data except for the captions (No Cap.).

nation)" model to the InferSent SNLI and AllNLI models on all STS tasks from 2012-2016. We then compare the overall mean with that of the three caption STS datasets within the collection. The results are shown in Table 68. The InferSent models are much closer to our WORD, TRIGRAM model on the caption datasets than overall, and InferSent trained on SNLI shows the largest difference between its overall performance and its performance on caption data.

We also compare the performance of these models on the STS Benchmark under several conditions (Table 69). Unsupervised results were obtained by simply using cosine similarity of the pretrained embeddings on the test set with no training or tuning. Supervised results were obtained by training and tuning using the training and development data of the STS Benchmark.

We first compare unsupervised results on the entire test set, the subset consisting of captions (3,250 of the 8,628 examples in the test set), and the remainder. We include analogous results in the supervised setting, where we filter the respective training and development sets in addition to the test sets. Compared to our model, InferSent shows a much larger gap between captions and non-captions, providing evidence of a bias. Note that this bias is smaller for the model trained on AllNLI, as its training data includes other domains.

laughed	PPDB	giggled, smiled, funny, used, grew, bust, ri, did
	ParaNMT-50M	chortled, guffawed, pealed, laughin, laughingstock, cackled, chuckled, snickered, mirthless, chuckling, jeered, laughs, laughing, taunted, burst, cackling, scoffed,...
respectful	PPDB	respect, respected, courteous, disrespectful, friendly, respecting, respectable, humble, environmentally-friendly, child-friendly, dignified, respects, compliant, sensitive,...
	ParaNMT-50M	reverent, deferential, revered, respectfully, awed, respect, respected, respects, respectable, politely, considerate, treat, civil, reverence, polite, keeping, behave, proper, dignified,...

Table 70: Example lexical paraphrases from PPDB ranked using the PPDB 2.0 scoring function and from the paraphrase lexicon we induced from ParaNMT-50Mranked using adjusted PMI.

8.6 paraphrase lexicon

While ParaNMT-50Mconsists of sentence pairs, we demonstrate how a paraphrase lexicon can be extracted from it. One simple approach is to extract and rank word pairs $\langle u, v \rangle$ using the **cross-sentence pointwise mutual information (PMI)**:

$$\text{PMI}_{\text{cross}}(u, v) = \log \frac{\#(u, v)\#(\cdot, \cdot)}{\#(u)\#(v)}$$

where joint counts $\#(u, v)$ are incremented when u appears in a sentence and v appears in its paraphrase. The marginal counts (e.g., $\#(u)$) are computed based on single-sentence counts, as in ordinary PMI. This works reasonably well but is not able to differentiate words that frequently occur in paraphrase pairs from words that simply occur frequently together in the same sentence. For example, "Hong" and "Kong" have high cross-sentence PMI. We can improve the score by subtracting the ordinary PMI that computes joint counts based on single-sentence co-occurrences. We call the result the **adjusted PMI**:

$$\text{PMI}_{\text{adj}}(u, v) = \text{PMI}_{\text{cross}}(u, v) - \text{PMI}(u, v)$$

Before computing these PMIs from ParaNMT-50M, we removed sentence pairs with a paraphrase score less than 0.35 and where either sentence is longer than 30 tokens. When computing the ordinary PMI with single-sentence context, we actually compute separate versions of this PMI score for translations and references in each ParaNMT-50Mpair, then we average them together. We did this because the two sentences in each pair have highly correlated information, so computing PMI on each half of the data would correspond to capturing natural corpus statistics in a standard application of PMI.

Dataset	Score	$\rho \times 100$
PPDB L	PPDB 2.0	37.97
PPDB XL	PPDB 2.0	52.32
PPDB XXL	PPDB 2.0	60.44
PPDB XXXL	PPDB 2.0	61.47
ParaNMT-50M	cross-sentence PMI	52.12
ParaNMT-50M	adjusted PMI	**61.59**

Table 71: Spearman's $\rho \times 100$ on SimLex-999 for scored paraphrase lexicons.

original	overall, i that it's a decent buy, and am happy that i own it.
paraphrase	it's a good buy, and i'm happy to own it.
original	oh, that's a handsome women, that is.
paraphrase	that's a beautiful woman.

Table 72: Examples from our paraphrase generation model that show the ability to canonicalize text and correct grammatical errors.

Table 71 shows an evaluation of the resulting score functions on the SimLex-999 word similarity dataset (Hill et al., 2015). As a baseline, we use the lexical portion of PPDB 2.0 (Pavlick et al., 2015), evaluating its ranking score as a similarity score and assigning a similarity of 0 to unseen word pairs.[8] Our adjusted PMI computed from ParaNMT-50Mis on par with the best PPDB lexicon.

Table 70 shows examples from PPDB and our paraphrase lexicon computed from ParaNMT-50M. Paraphrases from PPDB are ordered by the PPDB 2.0 scoring function. Paraphrases from our lexicon are ordered using our adjusted PMI scoring function; we only show paraphrases that appeared at least 10 times in ParaNMT-50M.

8.7 PARAPHRASE GENERATION

In addition to powering state-of-the-art paraphrastic sentence embeddings, our dataset is useful for paraphrase generation. We briefly describe two efforts in paraphrase generation here.

We have found that training an encoder-decoder model on ParaNMT-50Mcan produce a paraphrase generation model that canonicalizes text. For this experiment, we used a bidirectional LSTM encoder and a two-layer LSTM decoder with soft attention over the encoded states (Bahdanau et al., 2015). The attention computation consists of a bilinear

8 If both orderings for a SimLex word pair appear in PPDB, we average their PPDB 2.0 scores. If multiple lexical entries are found with different POS tags, we take the first instance.

product with a learned parameter matrix. Table 72 shows examples of output generated by this model, showing how the model is able to standardize the text and correct grammatical errors. This model would be interesting to evaluate for automatic grammar correction as it does so without any direct supervision. Future work could also use this canonicalization to improve performance of models by standardizing inputs and removing noise from data.

ParaNMT-50Mhas also been used for syntactically-controlled paraphrase generation; this work is described in detail by Iyyer et al. (2018). A syntactically controlled paraphrase network (SCPN) is trained to generate a paraphrase of a sentence whose constituent structure follows a provided parse template. A parse template contains the top two levels of a linearized parse tree. Table 57 shows example outputs using the SCPN. The paraphrases mostly preserve the semantics of the input sentences while changing their syntax to fit the target syntactic templates. The SCPN was used for augmenting training data and finding adversarial examples.

We believe that ParaNMT-50Mand future datasets like it can be used to generate rich paraphrases that improve the performance and robustness of models on a multitude of NLP tasks.

8.8 DISCUSSION

One way to view ParaNMT-50Mis as a way to represent the learned translation model in a monolingual generated dataset. This raises the question of whether we could learn an effective sentence embedding model from the original parallel text used to train the NMT system, rather than requiring the intermediate step of generating a paraphrase training set.

However, while Hill et al. (2016) and Mallinson et al. (2017) used trained NMT models to produce sentence similarity scores, their correlations are considerably lower than ours (by 10% to 35% absolute in terms of Pearson). It appears that NMT encoders form representations that do not necessarily encode the semantics of the sentence in a way conducive to STS evaluations. They must instead create representations suitable for a decoder to generate a translation. These two goals of representing sentential semantics and producing a translation, while likely correlated, evidently have some significant differences.

Our use of an intermediate dataset leads to the best results, but this may be due to our efforts in optimizing learning for this setting (Wieting et al., 2016b; Wieting and Gimpel, 2017). Future work will be needed to develop learning frameworks that can leverage parallel text directly to reach the same or improved correlations on STS tasks.

8.9 CONCLUSION

We described the creation of ParaNMT-50M, a dataset of more than 50M English sentential paraphrase pairs. We showed how to use ParaNMT-50Mto train paraphrastic sentence embeddings that outperform supervised systems on STS tasks, as well as how it can be used for generating paraphrases for purposes of data augmentation, robustness, and even grammar correction.

The key advantage of our approach is that it only requires parallel text. There are hundreds of millions of parallel sentence pairs, and more are being generated continually. Our procedure is immediately applicable to the wide range of languages for which we have parallel text.

We release ParaNMT-50M, our code, and pretrained sentence embeddings, which also exhibit strong performance as general-purpose representations for a multitude of tasks. We hope that ParaNMT-50M, along with our embeddings, can impart a notion of meaning equivalence to improve NLP systems for a variety of tasks. We are actively investigating ways to apply these two new resources to downstream applications, including machine translation, question answering, and additional paraphrase generation tasks.

Part III

MULTILINGUAL PARAPHRASTIC EMBEDDINGS

9

LEARNING SIMPLE AND EFFECTIVE BILINGUAL REPRESENTATIONS

9.1 INTRODUCTION

Measuring sentence similarity is a core task in semantics (Cer et al., 2017), and prior work has achieved strong results by training similarity models on datasets of paraphrase pairs (Dolan et al., 2004). However, such datasets are not produced naturally at scale and therefore must be created either through costly manual annotation or by leveraging natural annotation in specific domains, like Simple English Wikipedia (Coster and Kauchak, 2011) or Twitter (Lan et al., 2017b).

One of the most promising approaches for inducing paraphrase datasets is via manipulation of large bilingual corpora. Examples include bilingual pivoting over phrases (Callison-Burch et al., 2006; Ganitkevitch et al., 2013), and automatic translation of one side of the bitext (Wieting et al., 2017; Wieting and Gimpel, 2018; Hu et al., 2019) (Chapter chapters 7 and 8). However, this is costly – in Chapter 8 we reported that our large-scale database of sentential paraphrases required 10,000 GPU hours to generate.

In this paper, we propose a method that trains highly performant sentence embeddings (Pham et al., 2015; Hill et al., 2016; Pagliardini et al., 2017; McCann et al., 2017; Conneau et al., 2017) directly on bitext, obviating these intermediate steps and avoiding the noise and error propagation from automatic dataset preparation methods. This approach eases data collection, since bitext occurs naturally more often than paraphrase data and, further, has the additional benefit of creating cross-lingual representations that are useful for tasks such as mining or filtering parallel data and cross-lingual retrieval.

Most previous work for cross-lingual representations has focused on models based on encoders from neural machine translation (Espana-Bonet et al., 2017; Schwenk and Douze, 2017; Schwenk, 2018) or deep architectures using a contrastive loss (Grégoire and Langlais, 2018; Guo et al., 2018; Chidambaram et al., 2018). However, the paraphrastic sentence embedding literature has observed that simple models such as pooling word embeddings generalize significantly better than complex architectures as we show in Chapter 4. Here, we find a similar effect in the bilingual setting. We propose a simple model that not only produces state-of-the-art monolingual and bilingual sentence representations, but also encode sentences hundreds of times faster – an important factor when applying these representations for mining or filtering large amounts of bitext. Our approach forms the simplest method to date that is able to achieve state-of-the-art results on multiple monolingual and cross-lingual semantic textual similarity (STS) and parallel corpora mining tasks.[1]

Lastly, since bitext is available for so many language pairs, we analyze how the choice of language pair affects the performance of English paraphrastic representations, finding that using related languages yields the best results.

9.2 LEARNING SENTENCE EMBEDDINGS

9.2.1 *Models*

We first describe our objective function and then describe our encoder, in addition to several baseline encoders. The methodology proposed here borrows much from past work (Wieting and Gimpel, 2018; Guo et al., 2018; Grégoire and Langlais, 2018; Singla et al., 2018), but this specific setting has not been explored and, as we show in our experiments, is surprisingly effective.

TRAINING. The training data consists of a sequence of parallel sentence pairs (s_i, t_i) in source and target languages respectively. For each sentence pair, we randomly choose a *negative* target sentence t'_i during training that is not a translation of s_i. Our objective is to have source and target sentences be more similar than source and negative target examples by a margin δ:

$$\min_{\theta_{\text{src}}, \theta_{\text{tgt}}} \sum_i \Big[\delta - f_\theta(s_i, t_i) + f_\theta(s_i, t'_i)\Big]_+.$$

The similarity function is defined as:

$$f_\theta(s, t) = \cos\Big(g(s; \theta_{\text{src}}), g(t; \theta_{\text{tgt}})\Big)$$

1 In fact, we show that for monolingual similarity, we can devise random encoders that outperform some of this work.

where g is the sentence encoder with parameters for each language $\theta = (\theta_{\text{src}}, \theta_{\text{tgt}})$. To select t'_i we choose the most similar sentence in some set according to the current model parameters, i.e., the one with the highest cosine similarity.

NEGATIVE SAMPLING. The described objective can also be applied to monolingual paraphrase data, which we explore in our experiments. The choice of negative examples differs whether we are using a monolingual parallel corpus or a bilingual parallel corpus. In the monolingual case, we select from all examples in the batch except the current pair. However, in the bilingual case, negative examples are only selected from the sentences in the batch from the opposing language. To select difficult negative examples that aid training, we use the *mega-batching* procedure from Chapter 8, which aggregates M mini-batches to create one mega-batch and selects negative examples therefrom. Once each pair in the mega-batch has a negative example, the mega-batch is split back up into M mini-batches for training.

ENCODERS. Our primary sentence encoder simply averages the embeddings of subword units generated by `sentencepiece` (Kudo and Richardson, 2018); we refer to it as SP. This means that the sentence piece embeddings themselves are the only learned parameters of this model. As baselines we explore averaging character trigrams (TRIGRAM) from Chapter 5 and words (WORD). SP provides a compromise between averaging words and character trigrams, combining the more distinct semantic units of words with the coverage of character trigrams.

We also use a bidirectional long short-term memory LSTM encoder (Hochreiter and Schmidhuber, 1997), with LSTM parameters fully shared between languages , as well as BLSTM-SP, which uses sentence pieces instead of words as the input tokens. For all encoders, when the vocabularies of the source and target languages overlap, the corresponding encoder embedding parameters are shared. As a result, language pairs with more lexical overlap share more parameters.

We utilize several regularization methods from Chapter 6 including dropout (Srivastava et al., 2014) and shuffling the words in the sentence when training BLSTM-SP. Additionally, we find that annealing the mega-batch size by slowly increasing it during training improved performance by a significant margin for all models, but especially for BLSTM-SP.

Model	en-en	en-cs(1M)	en-cs(2M)
BLSTM-SP (20k)	66.5	66.4	66.2
SP (20k)	69.7	**70.0**	**71.0**
Word	66.7	65.2	66.8
Trigram	**70.0**	**70.0**	70.6

Table 73: Comparison between training on 1 million examples from a backtranslated English-English corpus (en-en) and the original bitext corpus (en-cs) sampling 1 million and 2 million sentence pairs (the latter equalizes the amount of English text with the en-en setting). Performance is the average Pearson's r over the 2012-2016 STS datasets.

9.3 EXPERIMENTS

Our experiments are split into two groups. First, we compare training on parallel data to training on back-translated parallel data. We evaluate these models on the 2012-2016 SemEval Semantic Textual Similarity (STS) shared tasks (Agirre et al., 2012, 2013, 2014, 2015, 2016), which predict the degree to which sentences have the same meaning as measured by human judges. The evaluation metric is Pearson's r with the gold labels. We use the small STS English-English dataset from Cer et al. (2017) for model selection. Second, we compare our best model, SP, on two semantic cross-lingual tasks: the 2017 SemEval STS task (Cer et al., 2017) which consists of monolingual and cross-lingual datasets and the 2018 Building and Using Parallel Corpora (BUCC) shared bitext mining task (Zweigenbaum et al., 2018).

9.3.1 *Hyperparameters and Optimization*

Unless otherwise specified, we fix the hyperparameters in our model to the following: mega-batch size to 60, margin δ to 0.4, annealing rate to 150,[2] dropout to 0.3, shuffling rate for BLSTM-SP to 0.3, and the size of the `sentencepiece` vocabulary to 20,000. For Word and Trigram, we limited the vocabulary to the 200,000 most frequent types in the training data. We optimize our models using Adam (Kingma and Ba, 2014) with a learning rate of 0.001 and trained the models for 10 epochs.

9.3.2 *Back-Translated Text vs. Parallel Text*

We first compare sentence encoders and sentence embedding quality between models trained on backtranslated text and those trained on bitext directly. As our bitext, we use

2 Annealing rate is the number of minibatches that are processed before the megabatch size is increased by 1.

the Czeng1.6 English-Czech parallel corpus (Bojar et al., 2016). We compare it to training on ParaNMT from Chapter 8, a corpus of 50 million paraphrases obtained from automatically translating the Czech side of Czeng1.6 into English. We sample 1 million examples from ParaNMT and Czeng1.6 and evaluate on all 25 datasets from the STS tasks from 2012-2016. Since the models see two full English sentences for every example when training on ParaNMT, but only one when training on bitext, we also experiment with sampling twice the amount of bitext data to keep fixed the number of English training sentences.

Results in Table 73 show two observations. First, models trained on `en-en`, in contrast to those trained on `en-cs`, have higher correlation for all encoders except SP. However, when the same number of English sentences is used, models trained on bitext have greater than or equal performance across all encoders. Second, SP has the best overall performance in the `en-cs` setting. It also has fewer parameters and is faster to train than BLSTM-SP and TRIGRAM. Further, it is faster at encoding new sentences at test time.

9.3.3 *Monolingual and Cross-Lingual Similarity*

We evaluate on the cross-lingual STS tasks from SemEval 2017. This evaluation contains Arabic-Arabic, Arabic-English, Spanish-Spanish, Spanish-English, and Turkish-English STS datsets. These datasets were created by translating one or both pairs of an English STS pair into Arabic (`ar`), Spanish (`es`), or Turkish (`tr`).[3]

BASELINES. We compare to several models from prior work (Guo et al., 2018; Chidambaram et al., 2018). A fair comparison to other models is difficult due to different training setups. Therefore, we perform a variety of experiments at different scales to demonstrate that even with much less data, our method has the best performance.[4] In the case of Schwenk (2018), we replicate their setting in order to do a fair comparison. [5]

As another baseline, we analyze the performance of averaging randomly initialized embeddings. We experiment with SP having `sentencepiece` vocabulary sizes of 20,000 and 40,000 tokens as well as TRIGRAM with a maximum vocabulary size of 200,000. The embed-

3 Note that for experiments with 1M OS examples, we trained for 20 epochs.

4 We do not directly compare to recent work in learning contextualized word embeddings (Peters et al., 2018; Devlin et al., 2018). While these have been very successful in many NLP tasks, they do not perform well on STS tasks without fine tuning.

5 Two follow-up papers (Artetxe and Schwenk, 2018a,b) use essentially the same underlying model, but we compare to Schwenk (2018) because it was the only one of these papers where the model has been made available when this paper was written.

dings have 300 dimensions and are initialized from a normal distribution with mean 0 and variance 1.

Model	Data	N	Dim.	ar-ar	ar-en	es-es	es-en	en-en	tr-en
Random TRIGRAM	OS	1M	300	67.9	1.8	77.3	2.8	73.7	19.4
Random SP (20k)	OS	1M	300	61.9	17.5	68.8	6.5	67.0	23.1
Random SP (40k)	OS	1M	300	58.3	16.1	68.2	10.4	66.6	22.2
SP (20k)	OS	1M	300	75.6	74.7	85.4	76.4	**84.5**	77.2
TRIGRAM	OS	1M	300	75.6	**75.2**	84.1	73.2	83.5	74.8
SP (80k)	OS	10M	1024	**76.2**	75.0	**86.2**	**78.3**	**84.5**	**77.5**
SP (20k)	EP	2M	300	-	-	78.6	54.9	79.1	-
SP (20k)	EP	2M	1024	-	-	81.0	56.4	80.4	-
Schwenk (2018)	EP	18M	1024	-	-	64.4	40.8	66.0	-
Espana-Bonet et al. (2017)	MIX	32.8M	2048	59	44	78	49	76	-
Chidambaram et al. (2018)	MIX	470M/500M	512	-	-	64.2	58.7	-	-
2017 STS 1st Place	-	-	-	**75.4**	**74.9**	**85.6**	**83.0**	**85.5**	**77.1**
2017 STS 2nd Place	-	-	-	**75.4**	71.3	85.0	81.3	85.4	74.2
2017 STS 3rd Place	-	-	-	74.6	70.0	84.9	79.1	85.4	73.6

Table 74: Comparison of our models with those in the literature and random encoder baselines. Performance is measured in Pearson's r (%). N refers to the number of examples in the training data. OS stands for OpenSubtitles, EP for Europarl, and MIX for a variety of domains.

RESULTS. The results are shown in Table 74. We make several observations. The first is that the 1024 dimension SP model trained on 2016 OpenSubtitles Corpus[6] (Lison and Tiedemann, 2016) outperforms prior work on 4 of the 6 STS datasets. This result outperforms the baselines from the literature as well, all of which use deep architectures.[7] Our SP model trained on Europarl[8] (EP) also surpasses the model from Schwenk (2018) which is trained on the same corpus. Since that model is based on many-to-many translation, Schwenk (2018) trains on nine (related) languages in Europarl. We only train on the splits of interest (`en-es` for STS and `en-de`/`en-fr` for the BUCC tasks) in our experiments.

Secondly, we find that SP outperforms TRIGRAM overall. This seems to be especially true when the languages have more `sentencepiece` tokens in common.

Lastly, we find that random encoders, especially random TRIGRAM, perform strongly in the monolingual setting. In fact, the random encoders are competitive or outperform all three models from the literature in these cases. For cross-lingual similarity, however,

6 http://opus.nlpl.eu/OpenSubtitles.php
7 Including a 3-layer transformer trained on a constructed parallel corpus (Chidambaram et al., 2018), a bidirectional gated recurrent unit (GRU) network trained on a collection of parallel corpora using `en-es`, `en-ar`, and `ar-es` bitext (Espana-Bonet et al., 2017), and a 3 layer bidirectional LSTM trained on 9 languages in Europarl (Schwenk, 2018).
8 http://opus.nlpl.eu/Europarl.php

Model	en-de	en-fr
Schwenk (2018)	76.1	74.9
SP (20k)	77.0	76.3
SP (40k)	**77.5**	**76.8**

Table 75: F1 scores for bitext mining on BUCC.

random encoders lag behind because they are essentially measuring the lexical overlap in the two sentences and there is little lexical overlap in the cross-lingual setting, especially for distantly related languages like Arabic and English.

9.3.4 *Mining Bitext*

Lastly, we evaluate on the BUCC shared task on mining bitext. This task consists of finding the gold *aligned* parallel sentences given two large corpora in two distinct languages. Typically, only about 2.5% of the sentences are aligned. Following Schwenk (2018), we train our models on Europarl and evaluate on the publicly available BUCC data.

Results in Table 75 on the French and German mining tasks demonstrate the proposed model outperforms Schwenk (2018), although the gap is substantially smaller than on the STS tasks. The reason for this is likely the domain mismatch between the STS data (image captions) and the training data (Europarl). We suspect that the deep NMT encoders of Schwenk (2018) overfit to the domain more than the simpler SP model, and the BUCC task uses news data which is closer to Europarl than image captions.

9.4 ANALYSIS

We next conduct experiments on encoding speed and analyze the effect of language choice.

9.4.1 *Encoding Speed*

Model	Dim	Sentences/Sec.
Schwenk (2018)	1024	2,601
Chidambaram et al. (2018)	512	3,049
SP (20k)	300	855,571
SP (20k)	1024	683,204

Table 76: A comparison of encoding times for our model compared to two models from prior work.

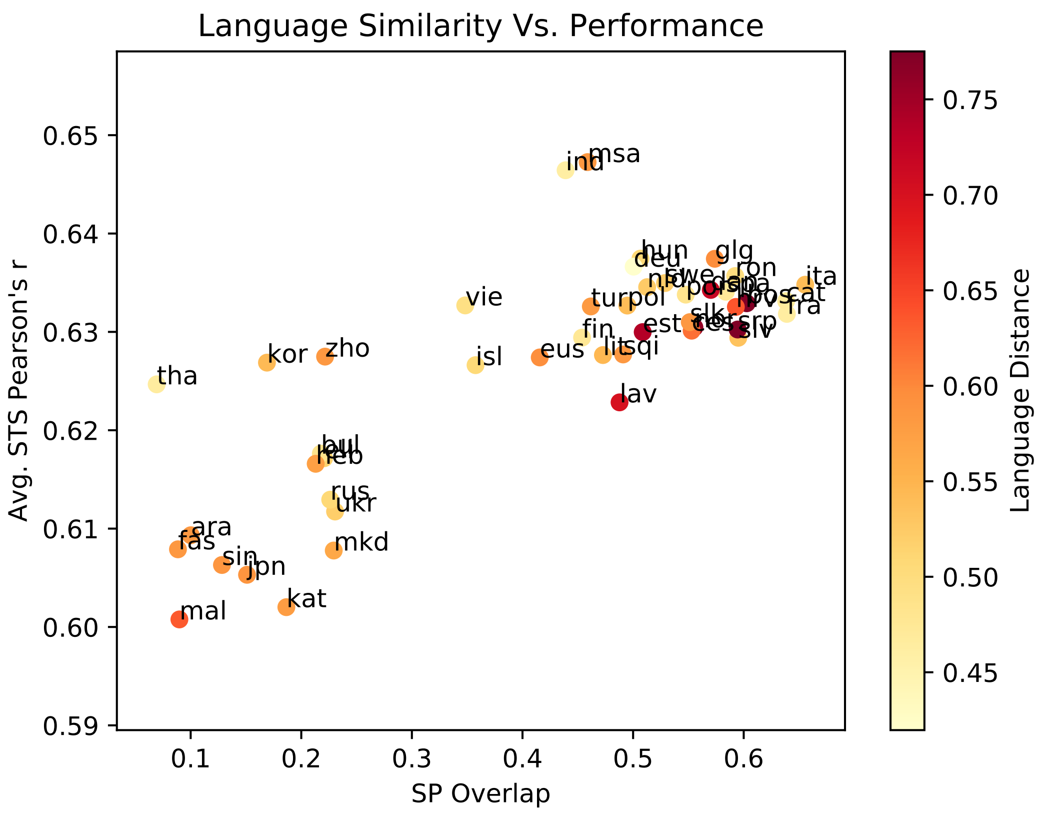


Figure 3: Plot of average performance on the 2012-2016 STS tasks compared to SP overlap and language distance as defined by Littell et al. (2017).

In addition to outperforming more complex models (Schwenk, 2018; Chidambaram et al., 2018), the simple SP models are much faster at encoding sentences. Since implementations to encode sentences are publicly available for several baselines, we are able to test their encoding speed and compare to SP. To do so, we randomly select 128,000 English sentences from the English-Spanish Europarl corpus. We then encode these sentences in batches of 128 on an Nvidia Quadro GP100 GPU. The number of sentences encoded per second is shown in Table 76, showing that SP is hundreds of times faster.

9.4.2 *Does Language Choice Matter?*

We next investigate the impact of the non-English language in the bitext when training English paraphrastic sentence embeddings. We took all 46 languages with at least 100k parallel sentence pairs in the 2016 OpenSubtitles Corpus (Lison and Tiedemann, 2016) and made a plot of their average STS performance on the 2012-2016 English datasets compared

Model	SP Ovl.	Lang. Distance
All Lang.	71.5	-22.8
Lang. (SP Ovl. $\leqslant$ 0.3)	23.6	-63.8
Lang. (SP Ovl. $>$ 0.3)	18.5	-34.2

Table 77: Spearman's $\rho \times 100$ between average performance on the 2012-2016 STS tasks compared to SP overlap (SP Ovl.) and language distance as defined by Littell et al. (2017). We included correlations for all languages as well as those with low and high SP overlap with English.

to their SP overlap[9] and language distance.[10] We segmented the languages separately and trained the models for 10 epochs using the 2017 `en-en` task for model selection.

The plot, shown in Figure 3, shows that `sentencepiece` (SP) overlap is highly correlated with STS score. There are also two clusters in the plot, languages that have a similar alphabet to English and those that do not. In each cluster we find that performance is negatively correlated with language distance. Therefore, languages similar to English yield better performance. The Spearman's correlations (multiplied by 100) for all languages and these two clusters are shown in Table 77. When choosing a language to pair up with English for learning paraphrastic embeddings, ideally there will be a lot of SP overlap. However, beyond or below a certain threshold (approximately 0.3 judging by the plot), the linguistic distance to English is more predictive of performance. Of the factors in URIEL, syntactic distance was the feature most correlated with STS performance in the two clusters with correlations of -56.1 and -29.0 for the low and high overlap clusters respectively. This indicates that languages with similar syntax to English helped performance. One hypothesis to explain this relationship is that translation quality is higher for related languages, especially if the languages have the same syntax, resulting in a cleaner training signal.

We also hypothesize that having high SP overlap is correlated with improved performance because the English SP embeddings are being updated more frequently during training. To investigate the effect, we again learned segmentations separately for both languages then prefixed all tokens in the non-English text with a marker to ensure that there would be no shared parameters between the two languages. Results showed that SP overlap was still correlated (correlation of 24.9) and language distance was still negatively correlated with performance albeit significantly less so at -10.1. Of all the linguistic features, again the syntactic distance was the highest correlated at -37.5.

9 We define SP overlap as the percentage of SPs in the English corpus that also appear in the non-English corpus.
10 We used the feature distance in URIEL (Littell et al., 2017) which accounts for a number of factors when calculating distance like phylogeny, geography, syntax, and phonology.

9.5 CONCLUSION

We have shown that using automatic dataset preparation methods such as pivoting or back-translation are not needed to create higher performing sentence embeddings. Moreover by using the bitext directly, our approach also produces strong paraphrastic cross-lingual representations as a byproduct. Our approach is much faster than comparable methods and yields stronger performance on cross-lingual and monolingual semantic similarity and cross-lingual bitext mining tasks.

10 DEEP GENERATIVE MODELS FOR BILINGUAL PARAPHRASTIC SENTENCE REPRESENTATIONS

10.1 INTRODUCTION

Learning useful representations of language has been a source of recent success in natural language processing (NLP). Much work has been done on learning representations for words (Mikolov et al., 2013b; Pennington et al., 2014) and sentences (Kiros et al., 2015; Conneau et al., 2017). More recently, deep neural architectures have been used to learn contextualized word embeddings (Peters et al., 2018; Devlin et al., 2018) which have enabled state-of-the-art results on many tasks. We focus on learning semantic *sentence* embeddings in this paper, which play an important role in many downstream applications. Since they do not require any labelled data for fine-tuning, sentence embeddings are useful for a variety of problems right out of the box. These include Semantic Textual Similarity (STS; Agirre et al. (2012)), mining bitext (Zweigenbaum et al., 2018), and paraphrase identification (Dolan et al., 2004). Semantic similarity measures also have downstream uses such as fine-tuning machine translation systems as discussed in 13.

There are three main ingredients when designing a sentence embedding model: the architecture, the training data, and the objective function. Many architectures including LSTMs (Hill et al., 2016; Conneau et al., 2017; Schwenk and Douze, 2017; Subramanian et al., 2018), Transformers (Cer et al., 2018; Reimers and Gurevych, 2019), and averaging models (Wieting et al., 2016b; Arora et al., 2017) (Chapter 4) have found success for learning sentence embeddings. The choice of training data and objective are intimately intertwined, and there are a wide variety of options including next-sentence prediction (Kiros et al., 2015), machine translation (Espana-Bonet et al., 2017; Schwenk and Douze, 2017; Schwenk, 2018; Artetxe and Schwenk, 2018b), natural language inference (NLI) (Conneau et al., 2017), and multi-task objectives which include some of the previously mentioned objectives (Cer et al., 2018) as well as additional tasks like constituency parsing (Subramanian et al., 2018).

or bilingual data (Chapter 9). The inclusion of latent variables in these models has also been explored (Chen et al., 2019).

Intuitively, bilingual data in particular is promising because it potentially offers a useful signal for learning the underlying semantics of sentences. Within a translation pair, properties shared by both sentences are more likely semantic, while those that are divergent are more likely stylistic or language-specific. While previous work learning from bilingual data perhaps takes advantage of this fact *implicitly*, the focus of this paper is modelling this intuition *explicitly*, and to the best of our knowledge, this has not not been explored in prior work. Specifically, we propose a deep generative model that is encouraged to perform *source separation* on parallel sentences, isolating what they have in common in a latent *semantic embedding* and explaining what is left over with *language-specific latent vectors*. At test time, we use inference networks (Kingma and Welling, 2013) for approximating the model's posterior on the semantic and source-separated latent variables to encode monolingual sentences. Finally, since our model and training objective are generative, our approach does not require knowledge of the distance metrics to be used during evaluation,[1] and it has the additional property of being able to generate text.

In experiments, we evaluate our probabilistic source-separation approach on a standard suite of STS evaluations. We demonstrate that the proposed approach is effective, most notably allowing the learning of high-capacity deep transformer architectures (Vaswani et al., 2017) while still generalizing to new domains, significantly outperforming a variety of state-of-the-art baselines. Further, we conduct a thorough analysis by identifying subsets of the STS evaluation where simple word overlap is not able to accurately assess semantic similarity. On these most difficult instances, we find that our approach yields the largest gains, indicating that our system is modeling interactions between words to good effect. We also find that our model better handles cross-lingual semantic similarity than multilingual translation baseline approaches, indicating that stripping away language-specific information allows for better comparisons between sentences from different languages.

Finally, we analyze our model to uncover what information was captured by the source separation into the semantic and language-specific variables and the relationship between this encoded information and language distance to English. We find that the language-specific variables tend to explain more superficial or language-specific properties such as overall sentence length, amount and location of punctuation, and the gender of articles (if gender is present in the language), but semantic and syntactic information is more concentrated in the shared semantic variables, matching our intuition. Language distance has an effect as well, where languages that share common structures with English put more in-

1 In other words, we don't assume cosine similarity as a metric, though it does work well in our experiments.

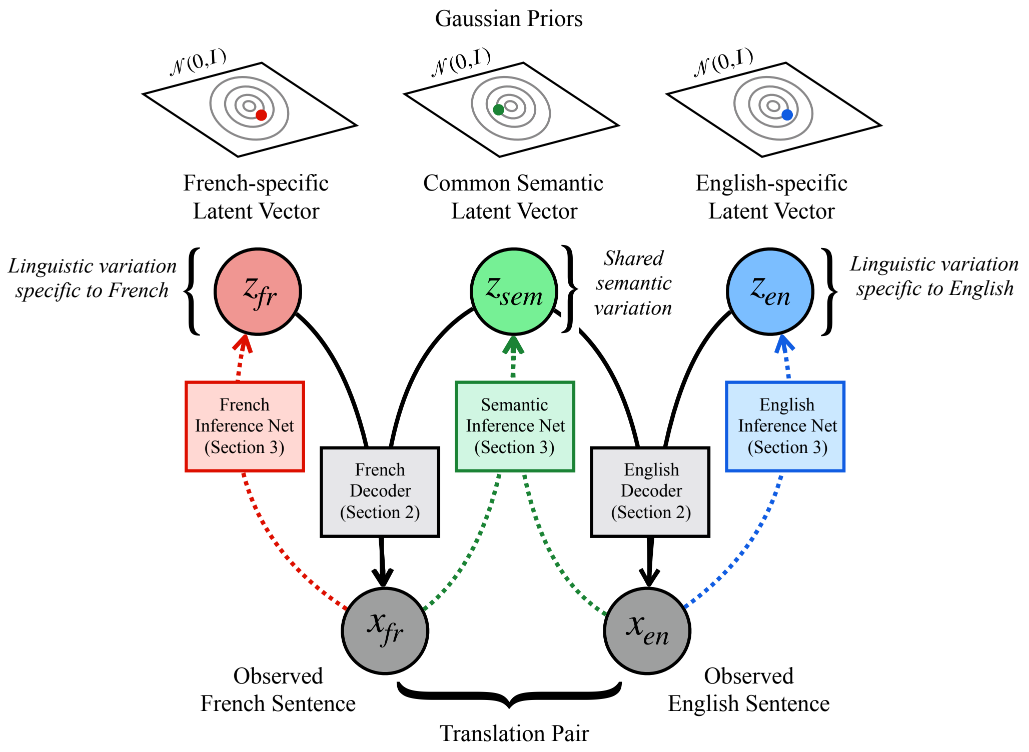

Gaussian Priors
$\mathcal{N}(0,I)$
$\mathcal{N}(0,I)$
$\mathcal{N}(0,I)$
French-specific Latent Vector
Common Semantic Latent Vector
English-specific Latent Vector
Linguistic variation specific to French
z_{fr}
z_{sem}
Shared semantic variation
z_{en}
Linguistic variation specific to English
French Inference Net (Section 3)
Semantic Inference Net (Section 3)
English Inference Net (Section 3)
French Decoder (Section 2)
English Decoder (Section 2)
x_{fr}
x_{en}
Observed French Sentence
Observed English Sentence
Translation Pair

Figure 4: The generative process of our model. Latent variables modeling the linguistic variation in French and English, z_{fr} and z_{en}, as well as a latent variable modeling the common semantics, z_{sem}, are drawn from a multivariate Gaussian prior. The observed text in each language is then conditioned on its language-specific variable and z_{sem}.

formation into the semantic variables, while more distant languages put more information into the language-specific variables. Lastly, we show outputs generated from our model that exhibit its ability to do a type of *style transfer*.

10.2 MODEL

10.2.1 *Model*

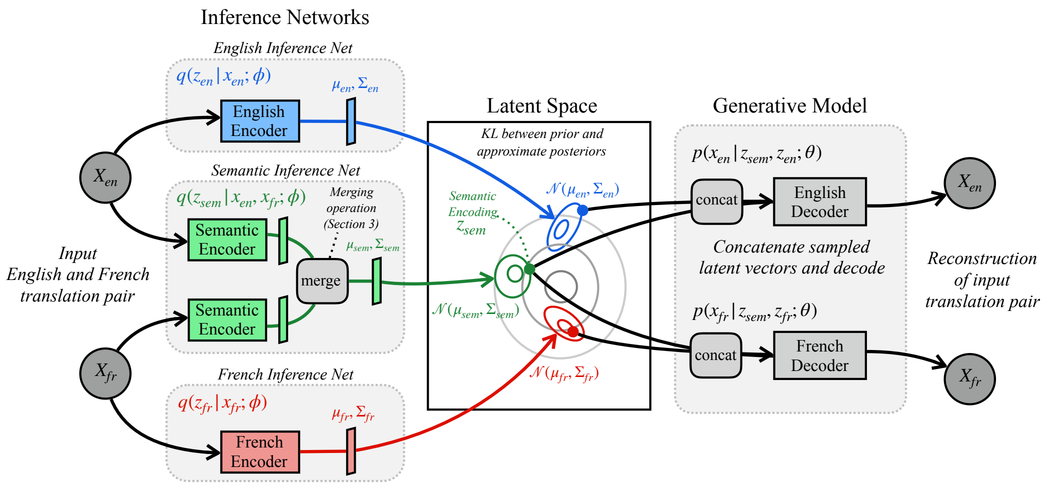
Our proposed training objective leverages a generative model of parallel text in two languages (e.g. English (en) and French (fr)) that form a pair consisting of an English sentence x_{en} and a French sentence x_{fr}. Importantly, this generative process utilizes three underlying latent vectors: language-specific variation variables (language variables) z_{fr} and z_{en} respectively for each side of the translation, as well as a shared semantic variation variable (semantic variable) z_{sem}. In this section we will first describe the generative model for the text and latent variables. In the following section we will describe the inference procedure of z_{sem} given an input sentence, which corresponds to our core task of obtaining sentence embeddings useful for downstream tasks such as semantic similarity.

Further, by encouraging the model to perform this source separation, the learned semantic encoders will more crisply represent the underlying semantics, increasing performance on downstream semantic tasks.

The generative process of our model, the Bilingual Generative Transformer (BGT), is depicted in Figure 4 and its computation graph is shown in Figure 5. First, we sample latent variables $\langle z_{fr}, z_{en}, z_{sem} \rangle$, where $z_i \in \mathbb{R}^k$, from a multivariate Gaussian prior $\mathcal{N}(0, I_k)$. These variables are then fed into a decoder that samples sentences; x_{en} is sampled conditioned on z_{sem} and z_{en}, while x_{fr} is sampled conditioned on z_{sem} and z_{fr}. Because sentences in both languages will use z_{sem} in generation, we expect that in a well-trained model this variable will encode semantic, syntactic, or stylistic information shared across both sentences, while z_{fr} and z_{en} will handle any language-specific peculiarities or specific stylistic decisions that are less central to the sentence meaning and thus do not translate across sentences. In the following section, we further discuss how this is explicitly encouraged by the learning process.

DECODER ARCHITECTURE. Many latent variable models for text use LSTMs (Hochreiter and Schmidhuber, 1997) as their decoders (Yang et al., 2017; Ziegler and Rush, 2019; Ma et al., 2019). However, state-of-the-art models in neural machine translation have seen increased performance and speed using deep Transformer architectures. We also found in our experiments (see Appendix A.3 for details) that Transformers led to increased performance in our setting, so they are used in our main model.

We use two decoders in our model, one for modelling $p(x_{fr}|z_{sem}, z_{fr}; \theta)$ and one for modeling $p(x_{en}|z_{sem}, z_{en}; \theta)$. These decoders are depicted on the right side of Figure 5. Each decoder takes in two latent variables, a language variable and a semantic variable. These variables are concatenated together prior to being used by the decoder for reconstruction. We explore four ways of using this latent vector: (1) Concatenate it to the word embeddings (Word) (2) Use it as the initial hidden state (Hidden, LSTM only) (3) Use it as you would the *attention context vector* in the traditional sequence-to-sequence framework (Attention) and (4) Concatenate it to the hidden state immediately prior to computing the logits (Logit). Unlike Attention, there is no additional feedforward layer in this setting. We experimented with these four approaches, as well as combinations thereof, and report this analysis in Appendix A.1. From these experiments, we see that the closer the sentence embedding is to the softmax, the better the performance on downstream tasks evaluating its semantic content. We hypothesise that this is due to better gradient propagation because the sentence



Inference Networks
English Inference Net
$q(z_{en}|x_{en};\phi)$
μ_{en}, Σ_{en}
English Encoder
Semantic Inference Net
$q(z_{sem}|x_{en}, x_{fr};\phi)$
Merging operation (Section 3)
μ_{sem}, Σ_{sem}
Semantic Encoder
merge
Semantic Encoder
French Inference Net
$q(z_{fr}|x_{fr};\phi)$
μ_{fr}, Σ_{fr}
French Encoder
X_{en}
X_{fr}
Input English and French translation pair
Latent Space
KL between prior and approximate posteriors
Semantic Encoding, z_{sem}
$\mathcal{N}(\mu_{en}, \Sigma_{en})$
$\mathcal{N}(\mu_{sem}, \Sigma_{sem})$
$\mathcal{N}(\mu_{fr}, \Sigma_{fr})$
Generative Model
$p(x_{en}|z_{sem}, z_{en};\theta)$
concat
English Decoder
Concatenate sampled latent vectors and decode
$p(x_{fr}|z_{sem}, z_{fr};\theta)$
concat
French Decoder
X_{en}
X_{fr}
Reconstruction of input translation pair

Figure 5: The computation graph for the variational lower bound used during training. The English and French text are fed into their respective inference networks and the semantic inference network to ultimately produce the language variables z_{fr} and z_{en} and semantic variable z_{sem}. Each language-specific variable is then concatenated to z_{sem} and used by the decoder to reconstruct the input sentence pair.

embedding is now closer to the error signal. Since Attention and Logit performed best, we use these in our Transformer experiments.

10.3 LEARNING AND INFERENCE

Our model is trained on a training set X of parallel text consisting of N examples, $X = \{\langle x^1_{en}, x^1_{fr}\rangle, \ldots, \langle x^N_{en}, x^N_{fr}\rangle\}$, and Z is our collection of latent variables $Z = (\langle z^1_{en}, z^1_{fr}, z^1_{sem}\rangle, \ldots, \langle z^N_{en}, z^N_{fr}, z^N_{sem}\rangle)$. We wish to maximize the likelihood of the parameters of the two decoders θ with respect to the observed X, marginalizing over the latent variables Z.

$$p(X;\theta) = \int_Z p(X, Z;\theta)dZ$$

Unfortunately, this integral is intractable due to the complex relationship between X and Z. However, related latent variable models like variational autoencoders (VAEs (Kingma and Welling, 2013)) learn by optimizing a variational lower bound on the log marginal likelihood. This surrogate objective is called the evidence lower bound (ELBO) and introduces a variational approximation, q to the true posterior of the model p. The q distribution is parameterized by a neural network with parameters ϕ. ELBO can be written for our model as follows:

$$\begin{aligned}\text{ELBO} =& \mathbb{E}_{q(Z|X;\phi)}[\log p(X|Z;\theta)] - \\ & \text{KL}(q(Z|X;\phi)||p(Z;\theta))\end{aligned}$$

This lower bound on the marginal can be optimized by gradient ascent by using the reparameterization trick (Kingma and Welling, 2013). This trick allows for the expectation under q to be approximated through sampling in a way that preserves backpropagation.

We make several independence assumptions for $q(z_{sem}, z_{en}, z_{fr}|x_{en}, x_{fr}; \phi)$. Specifically, to match our goal of source separation, we factor q as $q(z_{sem}, z_{en}, z_{fr}|x_{en}, x_{fr}; \phi) = q(z_{sem}|x_{en}, x_{fr}; \phi)q(z_{en}|x_{en})q(z_{fr}|x_{fr}; \phi)$, with ϕ being the parameters of the encoders that make up the inference networks, defined in the next paragraph.

Lastly, we note that the KL term in our ELBO equation encourages explaining variation that is shared by translations with the shared semantic variable and explaining language-specific variation with the corresponding language-specific variables. Information shared by the two sentences will result in a lower KL loss if it is encoded in the shared variable, otherwise that information will be replicated and the overall cost of encoding will increase.

ENCODER ARCHITECTURE. We use three inference networks as shown on the left side of Figure 5: an English inference network to produce the English language variable, a French inference network to produce the French language variable, and a semantic inference network to produce the semantic variable. Just as in the decoder architecture, we use a Transformer for the encoders.

The semantic inference network is a bilingual encoder that encodes each language. For each translation pair, we alternate which of the two parallel sentences is fed into the semantic encoder within a batch. Since the semantic encoder is meant to capture language agnostic semantic information, its outputs for a translation pair should be similar regardless of the language of the input sentence. We note that other operations are possible for combining the views each parallel sentence offers. For instance, we could feed both sentences into the semantic encoder and pool their representations. However, in practice we find that alternating works well and leave further study of this to future work.

10.4 EXPERIMENTS

10.4.1 *Baseline Models*

We experiment with twelve baseline models, covering both the most effective approaches for learning sentence embeddings from the literature and ablations of our own BGT model. These baselines can be split into three groups as detailed below.

MODELS FROM THE LITERATURE (TRAINED ON DIFFERENT DATA) We compare to well known sentence embedding models Infersent (Conneau et al., 2017), GenSen (Subramanian et al., 2018), the Universal Sentence Encoder (USE) (Cer et al., 2018), as well as BERT (Devlin et al., 2018).[2] We used the pretrained BERT model in two ways to create a sentence embedding. The first way is to concatenate the hidden states for the CLS token in the last four layers. The second way is to concatenate the hidden states of all word tokens in the last four layers and mean pool these representations. Both methods result in a 4096 dimension embedding. Finally, we compare to the newly released model, Sentence-Bert (Reimers and Gurevych, 2019). This model is similar to Infersent (Conneau et al., 2017) in that it is trained on natural language inference data, SNLI (Bowman et al., 2015). However, instead of using pretrained word embeddings, they fine-tune BERT in a way to induce sentence embeddings.[3]

MODELS FROM THE LITERATURE (TRAINED ON OUR DATA) These models are amenable to being trained in the exact same setting as our own models as they only require parallel text. These include the sentence piece averaging model, SP, from Chapter 9, which is among the best of the averaging models (i.e. compared to averaging only words or character n-grams) as well the LSTM model, LSTMAVG, from Chapter 6. These models use a contrastive loss with a margin. Following their settings, we fix the margin to 0.4 and tune the number of batches to pool for selecting negative examples from $\{40, 60, 80, 100\}$. For both models, we set the dimension of the embeddings to 1024. For LSTMAVG, we train a single layer bidirectional LSTM with hidden states of 512 dimensions. To create the sentence

2 Note that in all experiments using BERT, including Sentence-BERT, the large, uncased version is used.

3 Most work evaluating accuracy on STS tasks has averaged the Pearson's r over each individual dataset for each year of the STS competition. However, Reimers and Gurevych (2019) computed Spearman's ρ over concatenated datasets for each year of the STS competition. To be consistent with previous work, we re-ran their model and calculated results using the standard method, and thus our results are not the same as those reported Reimers and Gurevych (2019).

embedding, the forward and backward hidden states are concatenated and mean-pooled. Following Chapter 6, we shuffle the inputs with probability p, tuning p from $\{0.3, 0.5\}$.

We also implicitly compare to previous machine translation approaches like (Espana-Bonet et al., 2017; Schwenk and Douze, 2017; Artetxe and Schwenk, 2018b) in Appendix A.1 where we explore different variations of training LSTM sequence-to-sequence models. We find that our translation baselines reported in the tables below (both LSTM and Transformer) outperform the architectures from these works due to using the Attention and Logit methods mentioned in Section 10.2.1 , demonstrating that our baselines represent, or even over-represent, the state-of-the-art for machine translation approaches.

BGT ABLATIONS Lastly, we compare to ablations of our model to better understand the benefits of language-specific variables, benefits of the KL loss term, and how much we gain from the more conventional translation baselines.

- ENG. TRANS.: Translation from `en` to `fr`.
- MULTILING. TRANS.: Translation from both `en` to `fr` and `fr` to `en` where the encoding parameters are shared but each language has its own decoder.
- VAR. MULTILING. TRANS.: A model similar to MULTILING. TRANS., but it includes a prior over the embedding space and therefore a KL loss term. This model differs from BGT since it does not have any language-specific variables.
- BGT W/O PRIOR: Follows the same architecture as BGT, but without the priors and KL loss term.

10.4.2 *Experimental Settings*

The training data for our models is a mixture of OpenSubtitles 2018[4] `en-fr` data and `en-fr` Gigaword[5] data. To create our dataset, we combined the complete corpora of each dataset and then randomly selected 1,000,000 sentence pairs to be used for training with 10,000 used for validation. We use `sentencepiece` (Kudo and Richardson, 2018) with a vocabulary size of 20,000 to segment the sentences, and we chose sentence pairs whose sentences are between 5 and 100 tokens each.

In designing the model architectures for the encoders and decoders, we experimented with Transformers and LSTMs. Due to better performance, we use a 5 layer Transformer

4 `http://opus.nlpl.eu/OpenSubtitles.php`
5 `https://www.statmt.org/wmt10/training-giga-fren.tar`

for each of the encoders and a single layer decoder for each of the decoders. This design decision was empirically motivated as we found using a larger decoder was slower and worsened performance, but conversely, adding more encoder layers improved performance. More discussion of these trade-offs along with ablations and comparisons to LSTMs are included in Appendix A.3.

For all of our models, we set the dimension of the embeddings and hidden states for the encoders and decoders to 1024. Since we experiment with two different architectures,[6] we follow two different optimization strategies. For training models with Transformers, we use Adam (Kingma and Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-8}$. We use the same learning rate schedule as (Vaswani et al., 2017), i.e., the learning rate increases linearly for 4,000 steps to 5×10^{-4}, after which it is decayed proportionally to the inverse square root of the number of steps. For training the LSTM models, we use Adam with a fixed learning rate of 0.001. We train our models for 20 epochs.

For models incorporating a translation loss, we used label smoothed cross entropy (Szegedy et al., 2016; Pereyra et al., 2017) with $\epsilon = 0.1$. For BGT and MULTILING. TRANS., we anneal the KL term so that it increased linearly for 2^{16} updates, which robustly gave good results in preliminary experiments. We also found that in training BGT, combining its loss with the MULTILING. TRANS. objective during training of both models increased performance, and so this loss was summed with the BGT loss in all of our experiments. We note that this doesn't affect our claim of BGT being a generative model, as this loss is only used in a multi-task objective at training time, and we calculate the generation probabilities according to standard BGT at test time.

Lastly, in Appendix A.2, we illustrate that it is crucial to train the Transformers with large batch sizes. Without this, the model can learn the goal task (such as translation) with reasonable accuracy, but the learned semantic embeddings are of poor quality until batch sizes approximately reach 25,000 tokens. Therefore, we use a maximum batch size of 50,000 tokens in our ENG. TRANS., MULTILING. TRANS., and BGT W/O PRIOR, experiments and 25,000 tokens in our VAR. MULTILING. TRANS. and BGT experiments.

10.4.3 *Evaluation*

Our primary evaluation are the 2012-2016 SemEval Semantic Textual Similarity (STS) shared tasks (Agirre et al., 2012, 2013, 2014, 2015, 2016), where the goal is to accurately

6 We use LSTMs in our ablations.

Data	Sentence 1	Sentence 2	Gold Score
Hard+	Other ways are needed.	It is necessary to find other means.	4.5
Hard-	How long can you keep chocolate in the freezer?	How long can I keep bread dough in the refrigerator?	1.0
Negation	It's not a good idea.	It's a good idea to do both.	1.0

Table 78: Examples from our *Hard STS* dataset and our negation split. The sentence pair in the first row has dissimilar structure and vocabulary yet a high gold score. The second sentence pair has similar structure and vocabulary and a low gold score. The last sentence pair contains negation, where there is a *not* in Sentence 1 that causes otherwise similar sentences to have low semantic similarity.

predict the degree to which two sentences have the same meaning as measured by human judges. The evaluation metric is Pearson's r with the gold labels.

Secondly, we evaluate on *Hard STS*, where we combine and filter the STS datasets in order to make a more difficult evaluation. We hypothesize that these datasets contain many examples where their gold scores are easy to predict by either having similar structure and word choice and a high score or dissimilar structure and word choice and a low score. Therefore, we split the data using symmetric word error rate (SWER),[7] finding sentence pairs with low SWER and low gold scores as well as sentence pairs with high SWER and high gold scores. This results in two datasets, Hard+ which have SWERs in the bottom 20% of all STS pairs and whose gold label is between 0 and 1,[8] and Hard- where the SWERs are in the top 20% of the gold scores are between 4 and 5. We also evaluate on a split where negation was likely present in the example.[9] Examples are shown in Table 78.

Lastly, we evaluate on STS in `es` and `ar` as well as cross-lingual evaluations for `en-es`, `en-ar`, and `en-tr`. We use the datasets from SemEval 2017 (Cer et al., 2017). For this setting, we train Multiling. Trans. and BGT on 1 million examples from `en-es`, `en-ar`, and `en-tr` OpenSubtitles 2018 data.

10.4.4 *Results*

The results on the STS and *Hard STS* are shown in Table 79.[10] From the results, we see that BGT has the highest overall performance. It does especially well compared to prior work on the two *Hard STS* datasets.

7 We define symmetric word error rate for sentences s_1 and s_2 as $\frac{1}{2}\mathrm{WER}(s_1, s_2) + \frac{1}{2}\mathrm{WER}(s_2, s_2)$, since word error rate (WER) is an asymmetric measure.

8 STS scores are between 0 and 5.

9 We selected examples for the negation split where one sentence contained *not* or *'t* and the other did not.

10 We obtained values for STS 2012-2016 from prior works using SentEval (Conneau and Kiela, 2018). Note that we include all datasets for the 2013 competition, including SMT, which is not included in SentEval.

Model	Semantic Textual Similarity (STS)								
	2012	2013	2014	2015	2016	**Avg.**	Hard+	Hard-	**Avg.**
BERT (CLS)	33.2	29.6	34.3	45.1	48.4	38.1	7.8	12.5	10.2
BERT (Mean)	48.8	46.5	54.0	59.2	63.4	54.4	3.1	24.1	13.6
Infersent	61.1	51.4	68.1	70.9	70.7	64.4	4.2	29.6	16.9
GenSen	60.7	50.8	64.1	73.3	66.0	63.0	**24.2**	6.3	15.3
USE	61.4	59.0	70.6	74.3	73.9	67.8	16.4	28.1	22.3
Sentence-BERT	66.9	**63.2**	74.2	77.3	72.8	70.9	23.9	3.6	13.8
SP	68.4	60.3	75.1	78.7	76.8	71.9	19.1	29.8	24.5
LSTMAVG	67.9	56.4	74.5	78.2	75.9	70.6	18.5	23.2	20.9
ENG. TRANS.	66.5	60.7	72.9	78.1	78.3	71.3	18.0	47.2	32.6
MULTILING. TRANS.	67.1	61.0	73.3	78.0	77.8	71.4	20.0	**48.2**	34.1
VAR. MULTILING. TRANS.	68.3	61.3	74.5	79.0	78.5	72.3	24.1	46.8	**35.5**
BGT W/O PRIOR	67.6	59.8	74.1	78.4	77.9	71.6	17.9	45.5	31.7
BGT	**68.9**	62.2	**75.9**	**79.4**	**79.3**	**73.1**	22.5	46.6	34.6

Table 79: Results of our models and models from prior work. The first six rows are pretrained models from the literature, the next two rows are strong baselines trained on the same data as our models, and the last 5 rows include model ablations and BGT, our final model. We show results, measured in Pearson's $r \times 100$, for each year of the STS tasks 2012-2016 and our two *Hard STS* datasets.

We show further difficult splits in Table 80, including a negation split, beyond those used in *Hard STS* and compare the top two performing models in the STS task from Table 79. We also show easier splits in the bottom of the table.

From these results, we see that both positive examples that have little shared vocabulary and structure and negative examples with significant shared vocabulary and structure benefit significantly from using a deeper architecture. Similarly, examples where negation occurs also benefit from our deeper model. These examples are difficult because more than just the identity of the words is needed to determine the relationship of the two sentences, and this is something that SP is not equipped for since it is unable to model word order. The bottom two rows show *easier* examples where positive examples have high overlap and low SWER and vice versa for negative examples. Both models perform similarly on this data, with the BGT model having a small edge consistent with the overall gap between these two models.

Lastly, in Table 81, we show the results of STS evaluations in `es` and `ar` and cross-lingual evaluations for `en-es`, `en-ar`, and `en-tr`. From these results, we see that BGT has the best performance across all datasets, however the performance is significantly stronger than the MULTILING. TRANS. and BGT W/O PRIOR baselines in the cross-lingual setting. Since VAR.

Data Split	n	BGT	SP
All	13,023	**75.3**	74.1
Negation	705	**73.1**	68.7
Bottom 20% SWER, label $\in [0, 2]$	404	**63.6**	54.9
Bottom 10% SWER, label $\in [0, 1]$	72	**47.1**	22.5
Top 20% SWER, label $\in [3, 5]$	937	**20.0**	14.4
Top 10% SWER, label $\in [4, 5]$	159	**18.1**	10.8
Top 20% WER, label $\in [0, 2]$	1380	**51.5**	49.9
Bottom 20% WER, label $\in [3, 5]$	2079	**43.0**	42.2

Table 80: Performance, measured in Pearson's $r \times 100$, for different data splits of the STS data. The first row shows performance across all unique examples, the next row shows the negation split, and the last four rows show difficult examples filtered symmetric word error rate (SWER). The last two rows show relatively easy examples according to SWER.

Model	es-es	ar-ar	en-es	en-ar	en-tr
Multiling. Trans.	83.4	72.6	64.1	37.6	59.1
Var. Multiling. Trans.	81.7	72.8	72.6	73.4	74.8
BGT w/o Prior	84.5	73.2	68.0	66.5	70.9
BGT	**85.7**	**74.9**	**75.6**	**73.5**	**74.9**

Table 81: Performance measured in Pearson's $r \times 100$, on the SemEval 2017 STS task on the es-es, ar-ar, en-es, en-ar, and en-tr datasets.

Multiling. Trans. also has significantly better performance on these tasks, most of this gain seems to be due to the prior have a regularizing effect. However, BGT outperforms Var. Multiling. Trans. overall, and we hypothesize that the gap in performance between these two models is due to BGT being able to strip away the language-specific information in the representations with its language-specific variables, allowing for the semantics of the sentences to be more directly compared.

10.5 analysis

We next analyze our BGT model by examining what elements of syntax and semantics the language and semantic variables capture relative both to each-other and to the sentence embeddings from the Multiling. Trans. models. We also analyze how the choice of language and its lexical and syntactic distance from English affects the semantic and syntactic information captured by the semantic and language-specific encoders. Finally, we also show that our model is capable of sentence generation in a type of *style transfer*, demonstrating its capabilities as a generative model.

10.5.1 *STS*

We first show that the language variables are capturing little semantic information by evaluating the learned English language-specific variable from our BGT model on our suite of semantic tasks. The results in Table 82 show that these encoders perform closer to a random encoder than the semantic encoder from BGT. This is consistent with what we would expect to see if they are capturing extraneous language-specific information.

Model	Semantic Textual Similarity (STS)						
	2012	2013	2014	2015	2016	Hard+	Hard-
Random Encoder	51.4	34.6	52.7	52.3	49.7	4.8	17.9
English Language Encoder	44.4	41.7	53.8	62.4	61.7	15.3	26.5
Semantic Encoder	**68.9**	**62.2**	**75.9**	**79.4**	**79.3**	**22.5**	**46.6**

Table 82: STS performance on the 2012-2016 datasets and our *STS Hard* datasets for a randomly initialized Transformer, the trained English language-specific encoder from BGT, and the trained semantic encoder from BGT. Performance is measured in Pearson's $r \times 100$.

10.5.2 *Probing*

Lang.	Model	STS	S. Num.	O. Num.	Depth	Top Con.	Word	Len.	P. Num.	P. First	Gend.
fr	Multiling. Trans.	71.2	78.0	76.5	28.2	65.9	**80.2**	74.0	56.9	88.3	53.0
	Semantic Encoder	**72.4**	**84.6**	**80.9**	**29.7**	**70.5**	77.4	73.0	60.7	87.9	52.6
	en Language Encoder	56.8	75.2	72.0	28.0	63.6	65.4	**80.2**	**65.3**	**92.2**	-
	fr Language Encoder	-	-	-	-	-	-	-	-	-	**56.5**
es	Multiling. Trans.	70.5	84.5	82.1	29.7	68.5	**79.2**	77.7	63.4	90.1	54.3
	Semantic Encoder	**72.1**	**85.7**	**83.6**	**32.5**	**71.0**	77.3	76.7	63.1	89.9	52.6
	en Language Encoder	55.8	75.7	73.7	29.1	63.9	63.3	**80.2**	**64.2**	**92.7**	-
	es Language Encoder	-	-	-	-	-	-	-	-	-	**54.7**
ar	Multiling. Trans.	70.2	77.6	74.5	28.1	67.0	**77.5**	72.3	57.5	89.0	-
	Semantic Encoder	**70.8**	**81.9**	**80.8**	**32.1**	**71.7**	71.9	73.3	61.8	88.5	-
	en Language Encoder	58.9	76.2	73.1	28.4	60.7	71.2	**79.8**	**63.4**	**92.4**	-
tr	Multiling. Trans.	70.7	78.5	74.9	28.1	60.2	**78.4**	72.1	54.8	87.3	-
	Semantic Encoder	**72.3**	**81.7**	**80.2**	**30.6**	**66.0**	75.2	72.4	59.3	86.7	-
	en Language Encoder	57.8	77.3	74.4	28.3	63.1	67.1	**79.7**	**67.0**	**92.5**	-
ja	Multiling. Trans.	71.0	66.4	64.6	25.4	54.1	**76.0**	67.6	53.8	87.8	-
	Semantic Encoder	**71.9**	68.0	66.8	27.5	58.9	70.1	68.7	52.9	86.6	-
	en Language Encoder	60.6	**77.6**	**76.4**	**28.0**	**64.6**	70.0	**80.4**	**62.8**	**92.0**	-

Table 83: Average STS performance for the 2012-2016 datasets, measured in Pearson's $r \times 100$, followed by probing results on predicting number of subjects, number of objects, constituent tree depth, top constituent, word content, length, number of punctuation marks, the first punctuation mark, and whether the articles in the sentence are the correct gender. All probing results are measured in accuracy $\times 100$.

We probe our BGT semantic and language-specific encoders, along with our MULTILING. TRANS. encoders as a baseline, to compare and contrast what aspects of syntax and semantics they are learning relative to each other across five languages with various degrees of similarity with English. All models are trained on the OpenSubtitles 2018 corpus. We use the datasets from (Conneau et al., 2018a) for semantic tasks like number of subjects and number of objects, and syntactic tasks like tree depth, and top constituent. Additionally, we include predicting the word content and sentence length. We also add our own tasks to validate our intuitions about punctuation and language-specific information. In the first of these, *punctuation number*, we train a classifier to predict the number of punctuation marks[11] in a sentence. To make the task more challenging, we limit each label to have at most 20,000 examples split among training, validation, and testing data.[12] In the second task, *punctuation first*, we train a classifier to predict the identity of the first punctuation mark in the sentence. In our last task, *gender*, we detect examples where the gender of the articles in the sentence is incorrect in French of Spanish. To create an incorrect example, we switch articles from {le, la, un, une} for French and {el, la, los, las} for Spanish, with their (indefinite or definite for French and singular or plural for Spanish) counterpart with the opposite gender. This dataset was balanced so random chances gives 50% on the testing data. All tasks use 100,000 examples for training and 10,000 examples for validation and testing. The results of these experiments are shown in Table 83.

These results show that the source separation is effective - stylistic and language-specific information like length, punctuation and language-specific gender information are more concentrated in the language variables, while word content, semantic and syntactic information are more concentrated in the semantic encoder. The choice of language is also seen to be influential on what these encoders are capturing. When the languages are closely related to English, like in French and Spanish, the performance difference between the semantic and English language encoder is larger for word content, subject number, object number than for more distantly related languages like Arabic and Turkish. In fact, word content performance is directly tied to how well the alphabets of the two languages overlap. This relationship matches our intuition, because lexical information will be cheaper to encode in the semantic variable when it is shared between the languages. Similarly for the tasks of length, punctuation first, and punctuation number, the gap in performance between the two encoders also grows as the languages become more distant from English. Lastly,

11 Punctuation were taken from the set { ' ! " # $ % & \' ( ) ∗ + , − . / : ; < = > ? @ [ ] ^ ' { | } ~ . }.

12 The labels are from 1 punctuation mark up to 10 marks with an additional label consolidating 11 or more marks.

the gap on STS performance between the two encoders shrinks as the languages become more distant, which again is what we would expect, as the language-specific encoders are forced to capture more information.

Japanese is an interesting case in these experiments, where the English language-specific encoder outperforms the semantic encoder on the semantic and syntactic probing tasks. Japanese is a very distant language to English both in its writing system and in its sentence structure (it is an SOV language, where English is an SVO language). However, despite these difference, the semantic encoder strongly outperforms the English language-specific encoder, suggesting that the underlying meaning of the sentence is much better captured by the semantic encoder.

10.5.3 *Generation and Style Transfer*

Source	you know what i've seen?
Style	he said, "since when is going fishing" had anything to do with fish?"
Output	he said, "what is going to do with me since i saw you?"
Source	guys, that was the tech unit.
Style	is well, "capicci" ...
Output	is that what, "technician"?
Source	the pay is no good, but it's money.
Style	do we know cause of death?
Output	do we have any money?
Source	we're always doing stupid things.
Style	all right listen, i like being exactly where i am,
Output	all right, i like being stupid, but i am always here.

Table 84: *Style transfer* generations from our learned BGT model. *Source* refers to the sentence fed into the semantic encoder, *Style* refers to the sentence fed into the English language-specific encoder, and *Output* refers to the text generated by our model.

In this section, we qualitatively demonstrate the ability of our model to generate sentences. We focus on a *style-transfer* task where we have original seed sentences from which we calculate our semantic vector z_{sem} and language specific vector z_{en}. Specifically, we feed in a *Source* sentence into the semantic encoder to obtain z_{sem}, and another *Style* sentence into the English language-specific encoder to obtain z_{en}. We then generate a new sentence using these two latent variables. This can be seen as a type of style transfer where we expect the model to generate a sentence that has the semantics of the *Source* sentence and the style of the *Style* sentence. We use our `en-fr` BGT model from Table 83 and show some examples in Table 84. All input sentences are from held-out `en-fr` OpenSubtitles data. From these examples, we see further evidence of the role of the semantic and language-specific encoders, where most of the semantics (e.g. topical word such as *seen* and *tech* in the *Source* sentence) are reflected in the output, but length and structure are more strongly influenced by the language-specific encoder.

10.6 conclusion

We propose Bilingual Generative Transformers, a model that uses parallel data to learn to perform *source separation* of common semantic information between two languages from language-specific information. We show that the model is able to accomplish this source separation through probing tasks and text generation in a style-transfer setting. We find that our model bests all baselines on semantic similarity tasks, with the largest gains coming from a new challenge we propose as *Hard STS*, designed to foil methods approximating semantic similarity as word overlap. We also find our model to be especially effective on cross-lingual semantic similarity, due to its stripping away of language-specific information allowing for the underlying semantics to be more directly compared. In future work, we will explore generalizing this approach to the multilingual setting.

11 THE MULTILINGUAL GENERATIVE TRANSFORMER

11.1 INTRODUCTION

The BGT model presented in Chapter 10 separates language-specific information from interlingua semantic information

Within a translation pair, properties shared by both sentences are more likely semantic, while those that are divergent are more likely stylistic or language-specific.

In experiments, we evaluate the MGT on a suite of semantic evaluations. These include semantic similarity, both cross-lingual and monolingual, bitext mining including zero-shot bitext mining, and transfer learning using the XNLI natural language inference dataset (Conneau et al., 2018b). MGT significantly exceeds the performance of our strong translation baseline on all of these tasks. Moreover, our model exceeds or is competitive with LASER (Artetxe and Schwenk, 2018b), a multilingual sentence embeddings trained on an order of magnitude more data for a much longer period of time. We cannot replicate the exact setting of LASER since complete code for the model is not released and training on that large amount of data over 80 GPU-days is too costly.

Finally, we analyze our model to uncover what information was captured by the source separation into the semantic and language-specific variables. We find that the language-specific variables tend to explain more superficial or language-specific properties such as overall sentence length and amount and location of punctuation. We also find that semantic and syntactic information is more concentrated in the shared semantic variables, even more so than in our translation baseline, matching our intuition.

11.2 MODEL

11.2.1 *Model*

Our proposed training objective leverages a generative model of parallel text in multiple languages. Ideally, we would have N-way parallel text, but in practice we often have par-

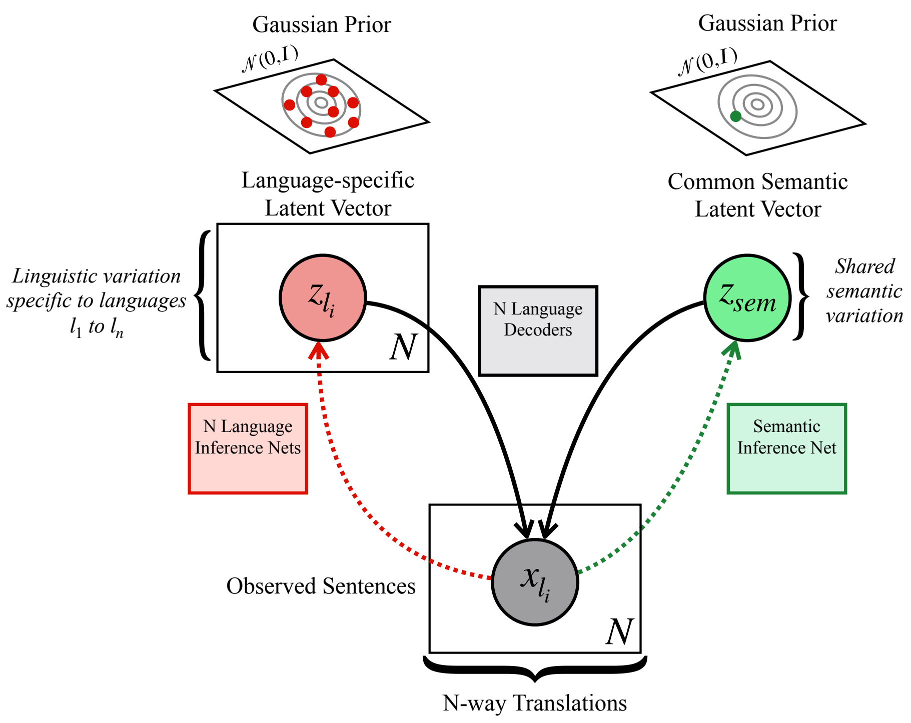


Figure 6: The generative process of our model. Latent variables modeling the linguistic variation for each language i, z_{l_i}, as well as a latent variable modeling the common semantics, z_{sem}, are drawn from a multivariate Gaussian prior. The observed text in each language is then conditioned on its language-specific variable and z_{sem}.

allel corpora for a set of language pairs. However, we can approximate an N-way parallel corpora by sampling translation pairs from this set of parallel corpora for each mini-batch.

Like the BGT in Chapter 10, our generative process utilizes two types of underlying latent vectors: language-specific variation variables (language variables) z_{l_i} for each language and a shared semantic variation variable (semantic variable) z_{sem}. In this section we will first describe the generative model for the text and latent variables. In the following section we will describe the inference procedure of z_{sem} given an input sentence, which corresponds to our core task of obtaining sentence embeddings useful for downstream tasks.

The generative process of MGT is depicted in Figure 6 and its computation graph is shown in Figure 7. First, we sample latent variables for two languages, l_i and l_j, and a latent semantic variable: $\langle z_{l_i}, z_{l_j}, z_{sem} \rangle$, where $z_k \in \mathbb{R}^n$, from a multivariate Gaussian prior $\mathcal{N}(0, I_n)$. These variables are then fed into a decoder that samples sentences; x_{l_i} is sampled conditioned on z_{sem} and z_{l_i}, while x_{l_j} is sampled conditioned on z_{sem} and z_{l_j}. Because sentences in both languages will use z_{sem} in generation, we expect that in a well-trained model this variable will encode semantic, syntactic, or stylistic information shared across both sentences, while z_{l_i} and z_{l_j} will handle any language-specific peculiarities or specific stylistic decisions that are less central to the sentence meaning and thus do not

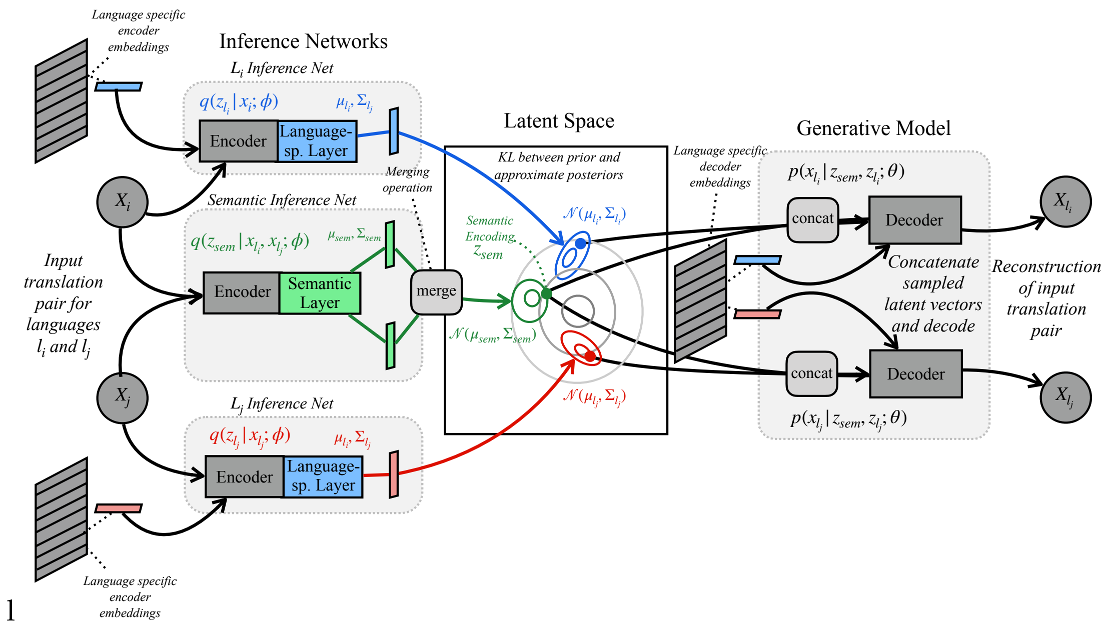

Language specific encoder embeddings
Inference Networks
L_i Inference Net
$q(z_{l_i}|x_i;\phi)$
μ_{l_i}, Σ_{l_i}
Encoder
Language-sp. Layer
X_i
Semantic Inference Net
$q(z_{sem}|x_{l_i}, x_{l_j};\phi)$
μ_{sem}, Σ_{sem}
Encoder
Semantic Layer
Merging operation
merge
Input translation pair for languages l_i and l_j
X_j
L_j Inference Net
$q(z_{l_j}|x_j;\phi)$
μ_{l_j}, Σ_{l_j}
Encoder
Language-sp. Layer
Language specific encoder embeddings
1
Latent Space
KL between prior and approximate posteriors
Semantic Encoding z_{sem}
$\mathcal{N}(\mu_{l_i}, \Sigma_{l_i})$
$\mathcal{N}(\mu_{sem}, \Sigma_{sem})$
$\mathcal{N}(\mu_{l_j}, \Sigma_{l_j})$
Language specific decoder embeddings
Generative Model
$p(x_{l_i}|z_{sem}, z_{l_i};\theta)$
concat
Decoder
Concatenate sampled latent vectors and decode
concat
Decoder
$p(x_{l_j}|z_{sem}, z_{l_j};\theta)$
X_{l_i}
X_{l_j}
Reconstruction of input translation pair

Figure 7: The computation graph for the variational lower bound used during training. Parallel text from languages l_i and l_j are fed into their respective inference networks and the semantic inference network to ultimately produce the language variables z_{l_i} and z_{l_j} and semantic variable z_{sem}. Each language-specific variable is then concatenated to z_{sem} and used by the decoder to reconstruct the input sentence pair.

translate across sentences. In the following section, we further discuss how this is explicitly encouraged by the learning process.

DECODER ARCHITECTURE. In contrast to our work on BGT 10, we use a single Transformer decoder to model $p(x_{l_i}|z_{sem}, z_{l_i};\theta)$ for all languages i. The decoder is depicted on the right side of Figure 7. The decoder takes in two latent variables, a language variable and a semantic variable. These variables are concatenated together to form a single embedding prior to being used by the decoder for reconstruction. Following our work on BGT, we concatenate this embeddings to the hidden state immediately prior to computing the logits as we found this yielded the best performance.[1] Since we are using a single decoder, we encode the desired output language by using a language-specific embedding as the first input to the decoder as depicted in the right side of Figure 7 to encourage the decoder outputs to follow the desired language.

11.3 LEARNING AND INFERENCE

Our model is trained on a training set X of parallel text consisting of N_ij parallel examples of languages i and j for a set of language pairs, $X = \{\langle x^1_{l_i}, x^1_{l_j}\rangle, \ldots, \langle x^N_{l_i}, x^N_{l_j}\rangle\}$, and Z is our collection of latent variables $Z = (\langle z^1_{l_i}, z^1_{l_j}, z^1_{sem}\rangle, \ldots, \langle z^N_{l_i}, z^N_{l_j}, z^N_{sem}\rangle)$. Just as we did for the BGT, we wish to maximize the likelihood of the parameters of the decoders θ with respect to the observed X, marginalizing over the latent variables Z. However, computing the marginal is intractable, and so therefore we optimize a surrogate objective called the evidence lower bound (ELBO). ELBO introduces a variational approximation, q to the true posterior of the model p. The q distribution is parameterized by a neural network with parameters ϕ. ELBO can be written for our model as follows:

$$\begin{aligned}\text{ELBO} =& \mathbb{E}_{q(Z|X;\phi)}[\log p(X|Z;\theta)] - \\ & \text{KL}(q(Z|X;\phi)||p(Z;\theta))\end{aligned}$$

We optimize ELBO by gradient ascent using the reparameterization trick (Kingma and Welling, 2013). This trick allows for the expectation under q to be approximated through sampling in a way that preserves backpropagation.

Similar to the BGT, we make several independence assumptions for $q(z_{sem}, z_{l_i}, z_{l_j}|x_{l_i}, x_{l_j}; \phi)$. Specifically, to match our goal of source separation, we factor q as $q(z_{sem}, z_{l_i}, z_{l_j}|x_{l_i}, x_{l_j}; \phi) = q(z_{sem}|x_{l_i}, x_{l_j}; \phi)q(z_{l_i}|x_{l_i})q(z_{l_j}|x_{l_j}; \phi)$, with ϕ being the parameters of the encoder that make up the inference networks, defined in the next paragraph.

Lastly, as mentioned in Chapter 10, the KL term in our ELBO equation encourages explaining variation that is shared by translations with the shared semantic variable and explaining language-specific variation with the corresponding language-specific variables. Information shared by the two sentences will result in a lower KL loss if it is encoded in the shared variable, otherwise that information will be replicated and the overall cost of encoding will increase.

ENCODER ARCHITECTURE. In contrast to the BGT, we use a single Transformer encoder as shown on the left side of Figure 5, however we modify the architecture and use

1 See Appendix A.1.

embeddings to have this single encoder play the role of $N + 1$ inference networks, one each for each of the N languages in our training data and another for our semantic network. To distinguish between the encoder acting as a language inference network for language l_i and as a semantic inference network, we append an embedding to the input sequence indicating the desired network. For a language-specific inference network, we append a `lang` embedding along with an embedding for l_i, the language of interest. For the semantic inference network, we append a `sem` embedding. We also switch out the top layer of the encoder depending if we are using a language-specific or semantic inference network. We assume that the language specific and semantic information can be mostly shared in the lower layers of the network, but switching out the top layer helps guide the appropriate information into the resulting sentence embedding.

For each translation pair, we alternate which of the two parallel sentences is fed into the semantic inference network within a batch. We sample the pairs so that each language will be fed into the inference network the same amount of times.[2] Since the semantic inference network is meant to capture language agnostic semantic information, its outputs for a translation pair should be similar regardless of the language of the input sentence. We note that other operations are possible for combining the views each parallel sentence offers. For instance, we could feed both sentences into the semantic encoder and pool their representations. However, in practice we find that alternating works well and leave further study of this to future work.

11.4 EXPERIMENTS

11.4.1 *Experimental Settings*

DATA We perform two experiments, one using four languages for ablations Arabic (`ar`, English `en`, Spanish `es`, and Turkish`tr`), and another more large-scale experiment using eight languages: (`ar`, `en`, `es`, `tr`, French `fr`, German `de`, Russian `ru`, and Chinese `zh`). In both cases, we use the data used by LASER for the appropriate languages with the exception

2 Since often most parallel data is aligned to a high resource language like English, we then decrease the probability of selecting English to be in line with the other languages. We set the probability of selecting English to be $\frac{1}{N+1}$ and the probability of selecting non-English to be $\frac{N}{N+1}$ for each translation pair.

of the Tatoeba corpus, since we use it for evaluation.[3][4]. Therefore, for all languages we use examples from the Open Subtitles 2018[5] (Lison and Tiedemann, 2016). We use the Tanzil corpus [6] (Tiedemann, 2012) for Arabic, Spanish, Russian, Turkish, and Chinese. We additionally use Europarl[7] for German, French, and Spanish. Lastly, we additionally use the MultiUN corpus[8] for Arabic, Russian, and Chinese.

BASELINE MODELS We compare to LASER (Artetxe and Schwenk, 2018b), a massively multilingual model trained on hundreds of millions of translations covering more than 100 languages. We note however, that this model was trained on 16 V100 GPUs for 5 days and trained on a superset of the data we used that is about an order of magnitude larger times larger. Therefore a fair comparison of the models between models is not possible, however we compare to both LASER and a similar translation ablation to give context to our results.

We experiment with 3 ablations of our model listed below:

- MULTILING. TRANS.: Translation from both en to X and X to en where the weight of the loss terms is $\frac{N}{N_1}$ for language X and $\frac{1}{N_1}$ for en so that all languages are reflected equally.
- VAR. MULTILING. TRANS.: A model similar to MULTILING. TRANS., but it includes a prior over the embedding space and therefore a KL loss term. This model differs from MGT since it does not have any language-specific variables.
- MGT W/O PRIOR: Follows the same architecture as MGT, but without the priors and KL loss term – it just contains the language variables.

HYPERPARAMETERS AND OPTIMIZATION We set the dimension of the embeddings and hidden states for the encoders and decoders to 1024. The feedforward layer for the Transformers have dimension 2048 and we use 16 attention heads. For training models with Transformers, we use Adam (Kingma and Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-8}$. We use the same learning rate schedule as (Vaswani et al., 2017), i.e., the learning rate increases linearly for 4,000 steps to 5×10^{-4}, after which it is decayed proportionally to the inverse square root of the number of steps. We train our models for 10 epochs.

3 LASER was trained on a portion of Tatoeba that was outside the evaluation set of the bitext mining task introduced in (Artetxe and Schwenk, 2018b)
4 We also note that LASER followed a fundamentally different training paradigm than we did where they used X-en and x-es data for language X. We only use X-en, so we doubled the data used for OpenSubtitles to try to more closely approximate the data used. However, for some corpora, this cannot be done like Europarl.
5 http://opus.nlpl.eu/OpenSubtitles.php
6 http://opus.nlpl.eu/Tanzil.php.
7 http://opus.nlpl.eu/Europarl.php
8 http://opus.nlpl.eu/MultiUN.php

For models incorporating a translation loss, we used label smoothed cross entropy (Szegedy et al., 2016; Pereyra et al., 2017) with $\epsilon = 0.1$. For MGT and VAR. MULTILING. TRANS., we anneal the KL term so that it increased linearly for the first 8 of 10 epochs. We also found that in training MGT, as with BGT, combining its loss with the MULTILING. TRANS. objective during training of both models increased performance and led to faster training, and so this loss was summed with the MGT loss in all of our experiments. We note that this doesn't affect our claim of MGT being a generative model, as this loss is only used in a multi-task objective at training time, and we calculate the generation probabilities according to standard MGT at test time.

11.4.2 *Evaluation*

We evaluate no a host of semantic language tasks detailed below:

- English STS: Our primary evaluation are the 2012-2016 SemEval Semantic Textual Similarity (STS) shared tasks (Agirre et al., 2012, 2013, 2014, 2015, 2016), where the goal is to accurately predict the degree to which two sentences have the same meaning as measured by human judges. The evaluation metric is Pearson's r with the gold labels.
- Cross-Lingual STS: We evaluate on cross-lingual STS for `en-es`, `en-ar`, and `en-tr` using the datasets from SemEval 2017 (Cer et al., 2017). We average the Pearson's r for all three datasets.
- non-English STS: We evaluate on the `es-es` and `ar-ar` non-English STS datasets from SemEval 2017 (Cer et al., 2017). We average the Pearson's r for these two datasets.
- Tatoeba: This is a bitext mining task introduced in (Artetxe and Schwenk, 2018b) based on the Tatoeba corpus. [9]. These datasets are small, about 1000 examples, and cosine similarity is used to re-align the bitext. We evaluate on the Arabic, Spanish, and Turkish languages and average performance (accuracy) for both the `en->xx` and `xx->en` directions.
- Zero-Shot Tatoeba: We evaluate on low resource languages similar to Spanish and Turkish (Galician and Azerbaijani respectively). We call this zero-shot since data from these languages was not included during training.
- XNLI: We evaluate on the transfer learning task XNLI from (Conneau et al., 2018b). In this task, a classifier is trained on the Stanford Natural Language Inference dataset (SNLI) (Bowman et al., 2015), which is in English, and then tested on translations of the

[9] https://tatoeba.org/eng/

test set in multiple languages. We evaluate on English as well as the average performance on Arabic, Spanish, and Turkish. Performance is measured on accurately the correct label for each sentence pair.

- BUCC: We evaluate on the BUCC bitext mining task from (Zweigenbaum et al., 2018) for `en-de`, `en-fr`, and `en-ru`. Performance is measured using F1.

11.4.3 *Results*

Model	En. STS	CL STS	non-En. STS	Tat.	ZS Tat.	En. XNLI	Tr. XNLI
LASER	64.9	65.8	74.5	**95.8**	**80.8**	**73.9**	**71.3**
MULTILING. TRANS.	71.4	68.7	77.7	88.2	37.7	69.9	55.0
VAR. MULTILING. TRANS.	72.8	72.3	79.9	91.1	41.1	69.0	54.7
MGT W/O PRIOR	70.5	69.6	75.9	90.2	36.5	69.4	56.9
MGT	**73.5**	**75.8**	**80.5**	95.4	53.8	70.8	63.7

Table 85: Results of LASER, our model, and ablations of our model, including a translation baseline. We show results, measured in Pearson's $r \times 100$, for each year of the STS tasks 2012-2016, the cross-lingual and non-English semantic textual similarity on the SemEval 2017 task. Tatoeba bitext mining and zero-shot bitext mining, English XNLI and XNLI for `ar`, `es`, and `tr` all measured in accuracy $\times 100$.

Model	`en-fr`	`en-de`	`en-ru`
LASER	87.60	91.24	90.03
MULTILING. TRANS.	74.15	71.97	70.28
MGT	86.95	90.30	89.00

Table 86: Results of LASER, MGT, and our translation baseline on the BUCC task. Performance is measured in F1 $\times 100$.

The results of our 4 language experiments, including ablations, are shown in Table 85 and BUCC results for our 8 language model are shown in Table 86. The results show that across tasks, the MGT has the strongest performance over its ablations, far exceeding the MULTILING. TRANS. baseline. This is especially true for cross-lingual tasks like cross-lingual STS and bitext mining. The ablations also show that all components, the KL term and the reconstruction term, contribute significantly to performance. Comparisons to LASER are difficult due to the difference in training time, training data size, and the number of languages, but it seems that our model has better performance on semantic similarity tasks and similar performance on bitext mining. However, for the XNLI transfer tasks, our performance lags behind.

11.5 CONCLUSION

We have demonstrated how the BGT model can be extended to the multilingual setting with our MGT model. The results are much stronger than our translation baseline for a variety of semantic multilingual tasks including semantic similarity, bitext mining, and transfer learning. Through ablations and analysis, we also show that all components of our model are necessary for optimal performance and that the MGT is also able to separate language-specific from semantic information despite using only a single encoder and decoder. Future work will include scaling up this model to cover many of the world's languages.

Part IV

APPLICATIONS OF PARAPHRASTIC SENTENCE REPRESENTATIONS

In this section, we discuss applications of our work in representation learning. This section encompasses work that has appeared in two papers: (Iyyer et al., 2018; Wieting et al., 2019a).

In Chapter 12, we apply our ParaNMT-50Mcorpus and sentence embedding models towards learning controllable paraphrase generation. Specifically we focus on controlling the syntax of the generated sentences. We find that we can learn a model where by just supplying a *parse template*, i.e. the top production of a constituent parse, we can generate a sentence with that syntax. We show that when these syntactic paraphrases are added to training, models become more robust to adversarial examples. We use our paraphrastic sentence embeddings to filter the generated the paraphrases, removing those that semantically diverge too much from the source sentence.

Then in Chapter 13, we use our *paraphrastic* representations, along with a proposed length penalty, for fine-tuning neural machine translation systems using minimum risk training. The conventional approach for fine-tuning is to use BLEU (Papineni et al., 2002), since BLEU is commonly used for evaluation. However, we find that using an embedding model to evaluate similarity allows the range of possible scores to be continuous and, as a result, introduces fine-grained distinctions between similar translations. This allows for partial credit, reduces the penalties on semantically correct but lexically different translations. and provides more informative gradients during the optimization process. The result is better performance on both human evaluations and BLEU score, along with faster convergence during training. This is the first work on fine-tuning neural machine translation models with a semantic similarity reward based on embeddings, and we see this as becoming a trend in the future.

12

CONTROLLED PARAPHRASE GENERATION

12.1 INTRODUCTION

Natural language processing datasets often suffer from a dearth of linguistic variation, which can hurt the generalization of models trained on them. Recent work has shown it is possible to easily "break" many learned models by evaluating them on *adversarial examples* (Goodfellow et al., 2015), which are generated by manually introducing lexical, pragmatic, and syntactic variation not seen in the training set (Ettinger et al., 2017). Robustness to such adversarial examples can potentially be improved by augmenting the training data, as shown by prior work that introduces rule-based lexical substitutions (Jia and Liang, 2017; Liang et al., 2017). However, more complex transformations, such as generating syntactically adversarial examples, remain an open challenge, as input semantics must be preserved in the face of potentially substantial structural modifications. In this chapter, we introduce a new approach for learning to do *syntactically controlled paraphrase generation*: given a sentence and a target syntactic form (e.g., a constituency parse), a system must produce a paraphrase of the sentence whose syntax conforms to the target.

General purpose syntactically controlled paraphrase generation is a challenging task. Approaches that rely on handcrafted rules and grammars, such as the question generation system of McKeown (1983), support only a limited number of syntactic targets. We introduce the first learning approach for this problem, building on the generality of neural encoder-decoder models to support a wide range of transformations. In doing so, we face two new challenges: (1) obtaining a large amount of paraphrase pairs for training, and (2) defining syntactic transformations with which to label these pairs.

Since no large-scale dataset of sentential paraphrases exists publicly, we follow our work from Chapter 8 and use our millions of automatically generated paraphrase pairs using neural *backtranslation*. Backtranslation naturally injects linguistic variation between the original sentence and its backtranslated counterpart. By running the process at a very large scale and testing for the specific variations we want to produce, we can gather ample input-output

★ Authors contributed equally.

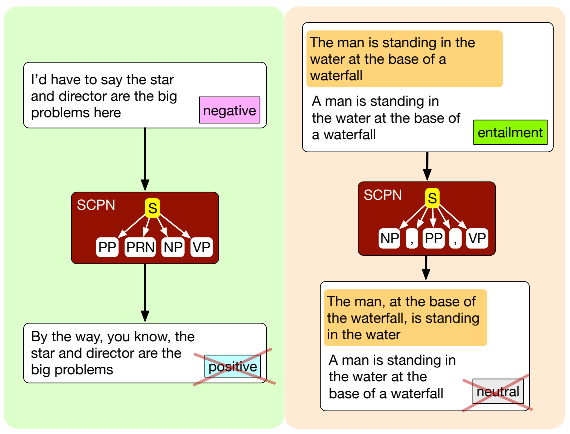

I'd have to say the star and director are the big problems here
negative
SCPN
S
PP
PRN
NP
VP
By the way, you know, the star and director are the big problems
~~positive~~
The man is standing in the water at the base of a waterfall
A man is standing in the water at the base of a waterfall
entailment
SCPN
S
NP
,
PP
,
VP
The man, at the base of the waterfall, is standing in the water
A man is standing in the water at the base of a waterfall
~~neutral~~

Figure 8: Adversarial examples for sentiment analysis (left) and textual entailment (right) generated by our syntactically controlled paraphrase network (SCPN) according to provided parse templates. In both cases, a pretrained classifier correctly predicts the label of the original sentence but not the corresponding paraphrase.

pairs for a wide range of phenomena. Our focus is on syntactic transformations, which we define using templates derived from linearized constituency parses (§12.2). Given such parallel data, we can easily train an encoder-decoder model that takes a sentence and target syntactic template as input, and produces the desired paraphrase.[1]

A combination of automated and human evaluations show that the generated paraphrases almost always follow their target specifications, while paraphrase quality does not significantly deteriorate compared to vanilla neural backtranslation (§12.4). Our model, the syntactically controlled paraphrase network (SCPN), is capable of generating adversarial examples for sentiment analysis and textual entailment datasets that significantly impact the performance of pretrained models (Figure 8). We also show that augmenting training sets with such examples improves robustness without harming accuracy on the original test sets (§12.5). Together these results not only establish the first general purpose syntactically controlled paraphrase approach, but also suggest that this general paradigm could be used for controlling many other aspects of the target text.

12.2 COLLECTING LABELED PARAPHRASE PAIRS

In this section, we describe a general purpose process for gathering and labeling training data for controlled paraphrase generation.

1 Code, labeled data, and pretrained models available at `https://github.com/miyyer/scpn`.

12.2.1 *Paraphrase data via backtranslation*

Inducing paraphrases from bilingual data has long been an effective method to overcome data limitations. In particular, bilingual pivoting (Bannard and Callison-Burch, 2005) finds quality paraphrases by pivoting through a different language. Mallinson et al. (2017) show that neural machine translation (NMT) systems outperform phrase-based MT on several paraphrase evaluation metrics.

In this paper, we use the ParaNMT-50M corpus from Chapter 8. This corpus consists of over 50 million paraphrases obtained by backtranslating the Czech side of the CzEng (Bojar et al., 2016) parallel corpus. The pretrained Czech-English model used for translation came from the Nematus NMT system (Sennrich et al., 2017). The training data of this system includes four sources: Common Crawl, CzEng 1.6, Europarl, and News Commentary. The CzEng corpus is the largest of these four and was found to have significantly more syntactic diversity than the other data sources (see Chapter 8).[2]

12.2.2 *Automatically labeling paraphrases with syntactic transformations*

We need labeled transformations in addition to paraphrase pairs to train a controlled paraphrase model. Manually annotating each of the millions of paraphrase pairs is clearly infeasible. Our key insight is that target transformations can be detected (with some noise) simply by parsing these pairs.[3]

Specifically, we parse the backtranslated paraphrases using the Stanford parser (Manning et al., 2014),[4] which yields a pair of constituency parses $\langle p_1, p_2 \rangle$ for each sentence pair $\langle s_1, s_2 \rangle$, where s_1 is the reference English sentence in the CzEng corpus and s_2 is its backtranslated counterpart. For syntactically controlled paraphrasing, we assume s_1 and p_2 are inputs, and the model is trained to produce s_2. To overcome learned biases of the NMT system, we also include reversed pairs $\langle s_2, s_1 \rangle$ during training.

12.2.2.1 *Syntactic templates*

To provide syntactic control, we linearize the bracketed parse structure without leaf nodes (i.e., tokens). For example, the corresponding linearized parse tree for the sentence *"She*

2 Syntactic diversity was measured by the entropy of the top two levels of parse trees in the corpora.

3 Similar automated filtering could be used to produce data for many other transformations, such as tense changes, point-of-view shifts, and even stylometric pattern differences (Feng et al., 2012). This is an interesting area for future work.

4 Because of the large dataset size, we use the faster but less accurate shift-reduce parser written by John Bauer.

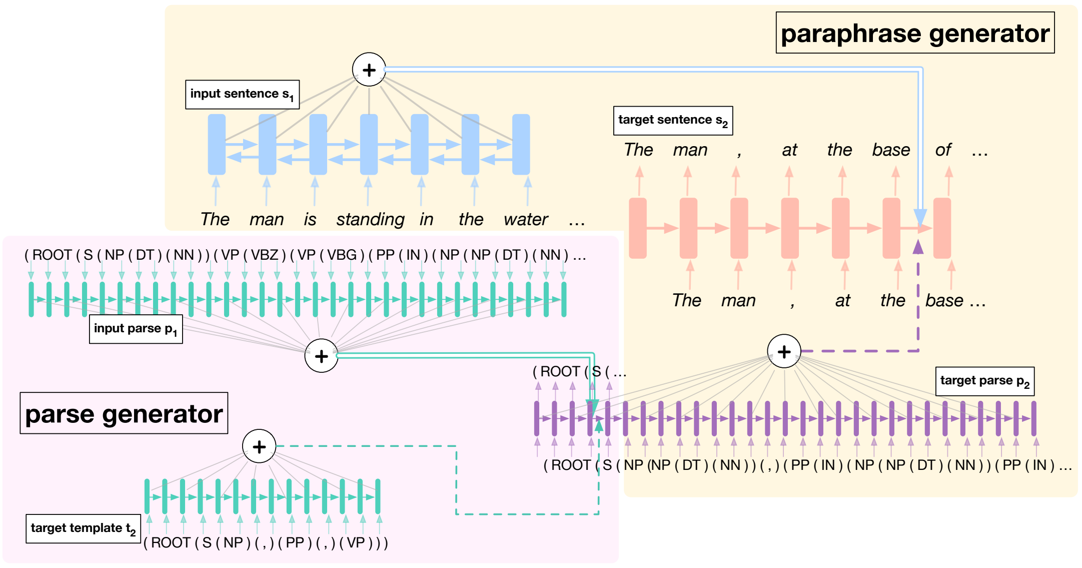

paraphrase generator
input sentence s_1
The man is standing in the water …
target sentence s_2
The man , at the base of …
The man , at the base …
(ROOT (S (NP (DT) (NN)) (VP (VBZ) (VP (VBG) (PP (IN) (NP (NP (DT) (NN) …
input parse p_1
parse generator
(ROOT (S (…
target parse p_2
(ROOT (S (NP (NP (DT) (NN)) (,) (PP (IN) (NP (NP (DT) (NN)) (PP (IN) …
target template t_2
(ROOT (S (NP) (,) (PP) (,) (VP)))

Figure 9: SCPN implements parse generation from templates as well as paraphrase generation from full parses as encoder-decoder architectures (attention depicted with dotted lines, copy mechanism with double stroked lines). While both components are trained separately, at test-time they form a pipelined approach to produce a controlled paraphrase from an input sentence s_1, its corresponding parse p_1, and a target template t_2.

drove home." is `(S(NP(PRP))(VP(VBD)(NP(NN)))(.))`. A system that requires a complete linearized target parse at test-time is unwieldy; how do we go about choosing the target parse? To simplify test-time usage, we relax the target syntactic form to a parse *template*, which we define as the top two levels of the linearized parse tree (the level immediately below the root along with the root); the prior example's template is $\texttt{S} \rightarrow \texttt{NP VP}$. In the next section, we design models such that users can feed in either parse templates or full parses depending on their desired level of control.

12.3 SYNTACTICALLY CONTROLLED PARAPHRASE NETWORKS

The SCPN encoder-decoder architecture is built from standard neural modules, as we describe in this section.

12.3.1 *Neural controlled paraphrase generation*

Given a sentential paraphrase pair $\langle s_1, s_2 \rangle$ and a corresponding target syntax tree p_2 for s_2, we encode s_1 using a bidirectional LSTM (Hochreiter and Schmidhuber, 1997), and our decoder is a two-layer LSTM augmented with soft attention over the encoded states (Bahdanau et al., 2015) as well as a copy mechanism (See et al., 2017). Following existing work in NMT (Sennrich et al., 2016c), we preprocess s_1 and s_2 into subword units using byte pair encoding, and we perform decoding using beam search. For all attention computations, we use a bilinear product with a learned parameter matrix $\mathbf{W}$: given vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, we score them by $\boldsymbol{u}^\top \mathbf{W} \boldsymbol{v}$.

We incorporate the target syntax p_2 into the generation process by modifying the inputs to the decoder. In particular, a standard decoder LSTM receives two inputs at every time step: (1) the embedding $\boldsymbol{w}_{t-1}$ of the ground-truth previous word in s_2, and (2) an attention-weighted average $\boldsymbol{a}_t$ of the encoder's hidden states. We additionally provide a representation $\boldsymbol{z}_t$ of the target p_2, so at every time step the decoder computes

$$\boldsymbol{h}_t = \text{LSTM}([\boldsymbol{w}_{t-1}; \boldsymbol{a}_t; \boldsymbol{z}_t]).$$

Since we preserve bracketed parse structure, our linearized parses can have hundreds of tokens. Forcing all of the relevant information contained by the parse tree into a single fixed representation (i.e., the last hidden state of an LSTM) is difficult with such large sequences. Intuitively, we want the decoder to focus on portions of the target parse tree that correspond with the current time step. As such, we encode p_2 using a (unidirectional) LSTM and compute $\boldsymbol{z}_t$ with an attention-weighted average of the LSTM's encoded states at every time step. This attention mechanism is conditioned on the decoder's previous hidden state $\boldsymbol{h}_{t-1}$.

12.3.2 *From parse templates to full parses*

As mentioned in Section 12.2.2.1, user-friendly systems should be able to accept high-level parse templates as input rather than full parses. Preliminary experiments show that SCPN struggles to maintain the semantics of the input sentence when we replace the full target parse with templates, and frequently generates short, formulaic sentences. The paraphrase generation model seems to rely heavily on the full syntactic parse to determine output length and clausal ordering, making it difficult to see how to modify the SCPN architecture for template-only target specification.

Instead, we train another model with exactly the same architecture as SCPN to generate complete parses from parse templates. This allows us to do the prediction in two steps: first predict the full syntactic tree and then use that tree to produce the paraphrase. Concretely, for the first step, assume t_2 is the parse template formed from the top two levels of the target parse p_2. The input to this *parse generator* is the input parse p_1 and t_2, and it is trained to produce p_2. We train the parse generator separately from SCPN (i.e., no joint optimization) for efficiency purposes. At test time, a user only has to specify an input sentence and target template; the template is fed through the parse generator, and its predicted target parse is in turn sent to SCPN for paraphrase generation (see Figure 9).

12.3.3 *Template selection and post-processing*

By switching from full parses to templates, we have reduced but not completely removed the burden of coming up with a target syntactic form. Certain templates may be not be appropriate for particular input sentences (e.g., turning a long sentence with multiple clauses into a noun phrase). However, others may be too similar to the input syntax, resulting in very little change. Since template selection is not a major focus of this paper, we use a relatively simple procedure, selecting the twenty most frequent templates in ParaNMT-50M.[5]

Since we cannot generate a valid paraphrase for every template, we postprocess to remove nonsensical outputs. In particular, we filter generated paraphrases using n-gram overlap and paraphrastic similarity, the latter of which is computed using the pretrained WORD,TRIAVG sentence embedding model from Chapter 8.[6] These paraphrastic sentence embeddings significantly outperform prior work due to the ParaNMT-50M data.

12.4 INTRINSIC EXPERIMENTS

Before using SCPN to generate adversarial examples on downstream datasets, we need to make sure that its output paraphrases are valid and grammatical and that its outputs follow the specified target syntax. In this section, we compare SCPN to a neural backtranslation baseline (NMT-BT) on the development set of our ParaNMT-50M split using both human and automated experiments. NMT-BT is the same pretrained Czech-English model used to create ParaNMT-50M; however, here we use it to generate in both directions (i.e., English-Czech and Czech-English).

5 However, we do provide some qualitative examples of rare and medium-frequency templates in Table 89.
6 After qualitatively analyzing the impact of different filtering choices, we set minimum n-gram overlap to 0.5 and minimum paraphrastic similarity to 0.7.

Model	2	1	0
SCPN w/ full parses	63.7	14.0	22.3
SCPN w/ templates	62.3	19.3	18.3
NMT-BT	65.0	17.3	17.7

Table 87: A crowdsourced paraphrase evaluation on a three-point scale (**0** = no paraphrase, **1** = ungrammatical paraphrase, **2** = grammatical paraphrase) shows both that NMT-BT and SCPN produce mostly grammatical paraphrases. Feeding parse templates to SCPN instead of full parses does not impact its quality.

12.4.1 *Paraphrase quality & grammaticality*

To measure paraphrase quality and grammaticality, we perform a crowdsourced experiment in which workers are asked to rate a paraphrase pair $\langle s, g \rangle$ on the three-point scale of Kok and Brockett (2010), where s is the source sentence and g is the generated sentence. A **0** on this scale indicates no paraphrase relationship, while **1** means that g is an ungrammatical paraphrase of s and **2** means that g is a grammatical paraphrase of s. We select 100 paraphrase pairs from the development set of our ParaNMT-50M split (after the postprocessing steps detailed in Section 12.3.3) and have three workers rate each pair.[7] To focus the evaluation on the effect of syntactic manipulation on quality, we only select sentences whose top-level parse templates differ (i.e., $t_s \neq t_g$), ensuring that the output of both systems varies syntactically from the source sentences.

The results (Table 87) show that the uncontrolled NMT-BT model's outputs are comparable in quality and grammaticality to those of SCPN; neither system has a significant edge. More interestingly, we observe no quality drop when feeding templates to SCPN (via the parse generator as described in Section 12.3.2) instead of complete parse trees, which suggests that the parse generator is doing a good job of generating plausible parse trees; thus, for all of the adversarial evaluations that follow, we only use the templated variant of SCPN.

12.4.2 *Do the paraphrases follow the target specification?*

We next determine how often SCPN's generated paraphrases conform to the target syntax: if g is a generated paraphrase and p_g is its parse, how often does p_g match the ground-truth target parse p_2? We evaluate on our development set using *exact template match*: g is deemed a syntactic match to s_2 only if the top two levels of its parse p_g matches those of

7 We use the Crowdflower platform for our experiments.

Model	Parse Acc.
SCPN w/ gold parse	64.5
SCPN w/ generated parse	51.6
Parse generator	99.9

Table 88: The majority of paraphrases generated by SCPN conform to the target syntax, but the level of syntactic control decreases when using generated target parses instead of gold parses. Accuracy is measured by exact template match (i.e., how often do the top two levels of the parses match).

p_2. We evaluate two SCPN configurations, where one is given the full target parse p_2 and the other is given the result of running our parse generator on the target template t_2. As a sanity check, we also evaluate our parse generator using the same metric.

The results (Table 88) show that SCPN does indeed achieve syntactic control over the majority of its inputs. Our parse generator produces full parses that almost always match the target template; however, paraphrases generated using these parses are less syntactically accurate.[8] A qualitative inspection of the generated parses reveals that they can differ from the ground-truth target parse in terms of ordering or existence of lower-level constituents (Table 92); we theorize that these differences may throw off SCPN's decoder.

The NMT-BT system produces paraphrases that tend to be syntactically very similar to the input sentences: 28.7% of these paraphrases have the same template as that of the input sentence s_1, while only 11.1% have the same template as the ground-truth target s_2. Even though we train SCPN on data generated by NMT backtranslation, we avoid this issue by incorporating syntax into our learning process.

12.5 ADVERSARIAL EXAMPLE GENERATION

The intrinsic evaluations show that SCPN produces paraphrases of comparable quality to the uncontrolled NMT-BT system while also adhering to the specified target specifications. Next, we examine the utility of controlled paraphrases for adversarial example generation. To formalize the problem, assume a pretrained model for some downstream task produces prediction y_x given test-time instance x. An *adversarial* example x' can be formed by making label-preserving modifications to x such that $y_x \neq y_{x'}$. Our results demonstrate that controlled paraphrase generation with appropriate template selection produces far more valid adversarial examples than backtranslation on sentiment analysis and entailment tasks.

8 With that said, exact match is a harsh metric; these paraphrases are more accurate than the table suggests, as often they differ by only a single constituent.

template	paraphrase
original	with the help of captain picard , the borg will be prepared for everything .
`(SBARQ(ADVP)(,)(S)(,)(SQ))`	now , the borg will be prepared by picard , will it ?
`(S(NP)(ADVP)(VP))`	the borg here will be prepared for everything .
`(S(S)(,)(CC)(S) (:)(FRAG))`	with the help of captain picard , the borg will be prepared , and the borg will be prepared for everything ... for everything .
`(FRAG(INTJ)(,)(S)(,)(NP))`	oh , come on captain picard , the borg line for everything .
original	you seem to be an excellent burglar when the time comes .
`(S(SBAR)(,)(NP)(VP))`	when the time comes , you 'll be a great thief .
`(S(“)(UCP)(”)(NP)(VP))`	“ you seem to be a great burglar , when the time comes . ” you said .
`(SQ(MD)(SBARQ))`	can i get a good burglar when the time comes ?
`(S(NP)(IN)(NP)(NP)(VP)`	look at the time the thief comes .

Table 89: Syntactically controlled paraphrases generated by SCPN for two examples from the ParaNMT-50M development set. For each input sentence, we show the outputs of four different templates; the fourth template is a failure case (highlighted in green) exhibiting semantic divergence and/or ungrammaticality, which occurs when the target template is unsuited for the input.

12.5.1 *Experimental setup*

We evaluate our syntactically adversarial paraphrases on the Stanford Sentiment Treebank (Socher et al., 2013, SST) and SICK entailment detection (Marelli et al., 2014). While both are relatively small datasets, we select them because they offer different experimental conditions: SST contains complicated sentences with high syntactic variance, while SICK almost exclusively consists of short, simple sentences. As a baseline, we compare the ten most probable beams from NMT-BT to controlled paraphrases generated by SCPN using ten templates randomly sampled from the template set described in Section 12.3.3.[9] We also need pretrained models for which to generate adversarial examples; we use the bidirectional LSTM baseline for both SST and SICK outlined in Tai et al. (2015) since it is a relatively simple architecture that has proven to work well for a variety of problems.[10] Since the SICK task involves characterizing the relationship between two sentences, for simplicity we only generate adversarial examples for the first sentence and keep the second sentence fixed to the ground truth.

9 We also experimented with the diverse beam search modification proposed by Li et al. (2016b) for NMT-BT but found that it dramatically warped the semantics of many beams; crowdsourced workers rated 49% of its outputs as **0** on the three-point scale.

10 We initialize both models using pretrained GloVe embeddings (Pennington et al., 2014) and set the LSTM hidden dimensionality to 300.

			No augmentation		**With augmentation**	
Model	Task	Validity	Test Acc	Dev Broken	Test Acc	Dev Broken
SCPN	SST	77.1	83.1	41.8	83.0	31.4
NMT-BT	SST	68.1	83.1	20.2	82.3	20.0
SCPN	SICK	77.7	82.1	33.8	82.7	19.8
NMT-BT	SICK	81.0	82.1	20.4	82.0	11.2

Table 90: SCPN generates more legitimate adversarial examples than NMT-BT, shown by the results of a crowdsourced validity experiment and the percentage of held-out examples that are broken through paraphrasing. Furthermore, we show that by augmenting the training dataset with syntactically-diverse paraphrases, we can improve the robustness of downstream models to syntactic adversaries (see "Dev Broken" before and after augmentation) without harming accuracy on the original test set.

12.5.2 *Breaking pretrained models*

For each dataset, we generate paraphrases for held-out examples and then run a pretrained model over them.[11] We consider a development example x *broken* if the original prediction y_x is correct, but the prediction $y_{x'}$ for at least one paraphrase x' is incorrect. For SST, we evaluate on the binary sentiment classification task and ignore all phrase-level labels (because our paraphrase models are trained on only sentences). Table 90 shows that for both datasets, SCPN breaks many more examples than NMT-BT. Moreover, as shown in Table 91, NMT-BT's paraphrases differ from the original example mainly by lexical substitutions, while SCPN often produces dramatically different syntactic structures.

12.5.3 *Are the adversarial examples valid?*

We have shown that we can break pretrained models with controlled paraphrases, but are these paraphrases actually valid adversarial examples? After all, it is possible that the syntactic modifications cause informative clauses or words (e.g., negations) to go missing. To measure the validity of our adversarial examples, we turn again to crowdsourced experiments. We ask workers to choose the appropriate label for a given sentence or sentence pair (e.g., positive or negative for SST), and then we compare the worker's judgment to the original development example's label. For both models, we randomly select 100 adversarial examples and have three workers annotate each one. The results (Table 90) show that on the more complex SST data, a higher percentage of SCPN's paraphrases are valid adversar-

11 Since the SICK development dataset is tiny, we additionally generate adversarial examples on its test set.

ial examples than those of NMT-BT, which is especially encouraging given our model also generates significantly *more* adversarial examples.

12.5.4 *Increasing robustness to adversarial examples*

If we additionally augment the *training* data of both tasks with controlled paraphrases, we can increase a downstream model's robustness to adversarial examples in the development set. To quantify this effect, we generate controlled paraphrases for the training sets of SST and SICK using the same templates as in the previous experiments. Then, we include these paraphrases as additional training examples and retrain our biLSTM task models.[12] As shown by Table 90, training on SCPN's paraphrases significantly improves robustness to syntactic adversaries without affecting accuracy on the original test sets. One important caveat is that this experiment only shows robustness to the set of templates used by SCPN; in real-world applications, careful template selection based on the downstream task, along with using a larger set of templates, is likely to increase robustness to less constrained syntactic adversaries. Augmentation with NMT-BT's paraphrases increases robustness on SICK, but on SST, it degrades test accuracy without any significant gain in robustness; this is likely due to its lack of syntactic variation compared to SCPN.

12.6 QUALITATIVE ANALYSIS

In the previous section, we quantitatively evaluated the SCPN's ability to produce valid paraphrases and adversarial examples. Here, we take a look at actual sentences generated by the model. In addition to analyzing SCPN's strengths and weaknesses compared to NMT-BT, we examine the differences between paraphrases generated by various configurations of the model to determine the impact of each major design decision (e.g., templates instead of full parses).

SYNTACTIC MANIPULATION: Table 89 demonstrates SCPN's ability to perform syntactic manipulation, showing paraphrases for two sentences generated using different templates. Many of the examples exhibit complex transformations while preserving both the input semantics and grammaticality, even when the target syntax is very different from that

12 We did not experiment with more complex augmentation methods (e.g., downweighting the contribution of paraphrased training examples to the loss).

template	original	paraphrase
(S(ADVP)(NP)(VP))	moody , heartbreaking , and filmed in a natural , unforced style that makes its characters seem entirely convincing even when its script is not .	so he 's filmed in a natural , unforced style that makes his characters seem convincing when his script is not .
(S(PP)(,)(NP)(VP))	there is no pleasure in watching a child suffer .	in watching the child suffer , there is no pleasure .
(S(S)(,)(CC)(S))	the characters are interesting and often very creatively constructed from figure to backstory .	the characters are interesting , and they are often built from memory to backstory .
	every nanosecond of the the new guy reminds you that you could be doing something else far more pleasurable .	each nanosecond from the new guy reminds you that you could do something else much more enjoyable .
	harris commands the screen , using his frailty to suggest the ravages of a life of corruption and ruthlessness .	harris commands the screen , using his weakness to suggest the ravages of life of corruption and recklessness .

Table 91: Adversarial sentiment examples generated by SCPN (top) and NMT-BT (bottom). The predictions of a pretrained model on the original sentences are correct (red is negative, blue is positive), while the predictions on the paraphrases are incorrect. The syntactically controlled paraphrases of SCPN feature more syntactic modification and less lexical substitution than NMT-BT's backtranslated outputs.

of the source (e.g., when converting a declarative to question). However, the failure cases demonstrate that not every template results in a valid paraphrase, as nonsensical outputs are sometimes generated when trying to squeeze the input semantics into an unsuitable target form.

ADVERSARIAL EXAMPLES: Table 91 shows that SCPN and NMT-BT differ fundamentally in the type of adversaries they generate. While SCPN mostly avoids lexical substitution in favor of making syntactic changes, NMT-BT does the opposite. These examples reinforce the results of the experiment in Section 12.4.2, which demonstrates NMT-BT's tendency to stick to the input syntax. While SCPN is able to break more validation examples than NMT-BT, it is alarming that even simple lexical substitution can break such a high percentage of both datasets we tested.

Ebrahimi et al. (2017) observe a similar phenomenon with HotFlip, their gradient-based substitution method for generating adversarial examples. While NMT-BT does not receive signal from the downstream task like HotFlip, it also does not require external constraints to maintain grammaticality and limit semantic divergence. As future work, it would be interesting to provide this downstream signal to both NMT-BT and SCPN; for the latter, perhaps this signal could guide the template selection process, which is currently fixed to a small, finite set.

TEMPLATES VS. GOLD PARSES: Why does the level of syntactic control decrease when we feed SCPN parses generated from templates instead of gold parses (Table 88)? The first two examples in Table 92 demonstrate issues with the templated approach. In the first example, the template is not expressive enough for the parse generator to produce slots for the highlighted clause. A potential way to combat this type of issue is to dynamically define templates based on factors such as the length of the input sentence. In the second example, a parsing error results in an inaccurate template which in turn causes SCPN to generate a semantically-divergent paraphrase. The final two examples show instances where the templated model performs equally as well as the model with gold parses, displaying the capabilities of our parse generator.

REMOVING SYNTACTIC CONTROL: To examine the differences between syntactically controlled and uncontrolled paraphrase generation systems, we train an SCPN without including z_t, the attention-weighted average of the encoded parse, in the decoder input. This uncontrolled configuration produces outputs that are very similar to its inputs, often identical syntactically with minor lexical substitution. Concretely, the uncontrolled SCPN produces a paraphrase with the same template as its input 38.6% of the time, compared to NMT-BT's 28.7% (Section 12.4.2).[13]

12.7 RELATED WORK

Paraphrase generation (Androutsopoulos and Malakasiotis, 2010; Madnani and Dorr, 2010) has been tackled using many different methods, including those based on hand-crafted rules (McKeown, 1983), synonym substitution (Bolshakov and Gelbukh, 2004), machine translation (Quirk et al., 2004), and, most recently, deep learning (Prakash et al., 2016; Mallinson et al., 2017; Dong et al., 2017). Our syntactically controlled setting also relates to controlled language generation tasks in which one desires to generate or rewrite a sentence with particular characteristics. We review related work in both paraphrase generation and controlled language generation below.

13 A configuration without the copy mechanism copies input syntax even more, with a 47.7% exact template match.

template	`(S(CC)(S)(,)(NP)(ADVP)(VP))`
original	damian encouraged me , criticized , he ... he always made me go a little deeper .
scpn parse	but damian , he supported me , he told me , he always made me go a little deeper .
scpn template	but damian supported me , he always made me go a little deeper .
template	`(S(S)(,)(NP)(VP))`
original	zacharias did n't deserve to die , grishanov thought , and he was aware of the huge irony of his situation
scpn parse	zacharias did not deserve to die , grishanov told himself , realizing the greatest irony of all .
scpn template	zacharias did not deserve to die , he was aware of the great irony of his situation .
template	`S(S)(,)(S))`
original	give me some water , my lips are dry , and i shall try to tell you .
scpn parse	give me some water , i have just a dry mouth .
scpn template	give me some water , my lips are dry .
template	`(S(NP)(,)(ADVP)(,)(VP))`
original	in the meantime , the house is weakened , and all its old alliances and deals are thrown into doubt .
scpn parse	the house , meanwhile , is weakening , which will be all of its old alliances and business .
scpn template	the house , meanwhile , is weakened , and its old alliances and deals are thrown into doubt .

Table 92: Examples from ParaNMT-50M comparing the output of two SCPN configurations, one with gold target parses (SCPN parse) and one with parses generated from templates (SCPN template), where templates are the top two levels of the gold parses. The first two examples demonstrate issues with missing information caused by inexpressive templates and parsing errors, respectively. The remaining examples, in which both configurations produce syntactically similar paraphrases, showcase the ability of the parse generator to produce viable full parses.

12.7.1 *Data-driven paraphrase generation*

Madnani and Dorr (2010) review data-driven methods for paraphrase generation, noting two primary families: template-based and translation-based. The first family includes approaches that use hand-crafted rules (McKeown, 1983), thesaurus-based substitution (Bolshakov and Gelbukh, 2004; Zhang and LeCun, 2015), lattice matching (Barzilay and Lee, 2003), and template-based "shake & bake" paraphrasing (Carl et al., 2005). These methods often yield grammatical outputs but they can be limited in diversity.

The second family includes methods that rewrite the input using methods based on parallel text (Bannard and Callison-Burch, 2005), machine translation (Quirk et al., 2004; Napoles et al., 2016; Suzuki et al., 2017), or related statistical techniques (Zhao et al., 2009). Of particular relevance to our work are methods that incorporate syntax to improve fluency of paraphrase output. Callison-Burch (2008) constrains paraphrases to be the same syntactic type as the input, though he was focused on phrase-level, not sentential, paraphrasing. Pang et al. (2003) learn finite-state automata from translation pairs that generate syntactic paraphrases, though this requires multiple translations into the same language and cannot be used to generate paraphrases outside this dataset. Shen et al. (2006) extend this to deeper syntactic analysis. All of these approaches use syntax to improve grammaticality, which is handled by our decoder language model.

Recent efforts involve neural methods. Iyyer et al. (2014) generate paraphrases with dependency tree recursive autoencoders by randomly selecting parse trees at test time. Li et al. (2017) generate paraphrases using deep reinforcement learning. Gupta et al. (2017) use variational autoencoders to generate multiple paraphrases. These methods differ from our approach in that none offer fine-grained control over the syntactic form of the paraphrase.

12.7.2 *Controlled language generation*

There is growing interest in generating language with the ability to influence the topic, style, or other properties of the output.

Most related to our methods are those based on syntactic transformations, like the tree-to-tree sentence simplification method of Woodsend and Lapata (2011) based on quasi-synchronous grammar (Smith and Eisner, 2006b). Our method is more general since we do not require a grammar and there are only soft constraints. Perhaps the closest to the proposed method is the conditioned recurrent language model of Ficler and Goldberg (2017), which produces language with user-selected properties such as sentence length and formality but is incapable of generating paraphrases.

For machine translation output, Niu et al. (2017) control the level of formality while Sennrich et al. (2016a) control the level of politeness. For dialogue, Li et al. (2016a) affect the output using speaker identity, while Wang et al. (2017) develop models to influence topic and style of the output. Shen et al. (2017) perform style transfer on non-parallel texts, while Guu et al. (2017) generate novel sentences from prototypes; again, these methods are not necessarily seeking to generate meaning-preserving paraphrases, merely transformed sentences that have an altered style.

12.8 CONCLUSION

We propose SCPN, an encoder-decoder model for syntactically controlled paraphrase generation, and show that it is an effective way of generating adversarial examples. Using a parser, we label syntactic variation in large backtranslated data, which provides training data for SCPN. The model exhibits far less lexical variation than existing uncontrolled paraphrase generation systems, instead preferring purely syntactic modifications. It is capable of generating adversarial examples that fool pretrained NLP models. Furthermore, by training on such examples, we increase the robustness of these models to syntactic variation.

13

NEURAL MACHINE TRANSLATION

13.1 INTRODUCTION

In neural machine translation (NMT) and other natural language generation tasks, it is common practice to improve likelihood-trained models by further tuning their parameters to explicitly maximize an automatic metric of system accuracy – for example, BLEU (Papineni et al., 2002) or METEOR (Denkowski and Lavie, 2014). Directly optimizing accuracy metrics involves backpropagating through discrete decoding decisions, and thus is typically accomplished with structured prediction techniques like reinforcement learning (Ranzato et al., 2016), minimum risk training (Shen et al., 2016), and other specialized methods (Wiseman and Rush, 2016). Generally, these methods work by repeatedly generating a translation under the current parameters (via decoding, sampling, or loss-augmented decoding), comparing the generated translation to the reference, receiving some reward based on their similarity, and finally updating model parameters to increase future rewards.

In the vast majority of work, discriminative training has focused on optimizing BLEU (or its sentence-factored approximation). This is not surprising given that BLEU is the standard metric for system comparison at test time. However, BLEU is not without problems when used as a training criterion. Specifically, since BLEU is based on n-gram precision, it aggressively penalizes lexical differences even when candidates might be synonymous with or similar to the reference: if an n-gram does not exactly match a sub-sequence of the reference, it receives no credit. While the pessimistic nature of BLEU differs from human judgments and is therefore problematic, it may, in practice, pose a more substantial problem for a different reason: BLEU is difficult to *optimize* because it does not assign partial credit. As a result, learning cannot hill-climb through intermediate hypotheses with high synonymy or semantic similarity, but low n-gram overlap. Furthermore, where BLEU does assign credit, the objective is often flat: a wide variety of candidate translations can have the same degree of overlap with the reference and therefore receive the same score. This, again, makes optimization difficult because gradients in this region give poor guidance.

In this chapter we propose SimiLe, a simple alternative to matching-based metrics like BLEU for use in discriminative NMT training. As a new reward, we introduce a measure of *semantic similarity between the generated hypotheses and the reference translations* evaluated by an embedding model trained on a large external corpus of paraphrase data. Using an embedding model to evaluate similarity allows the range of possible scores to be continuous and, as a result, introduces fine-grained distinctions between similar translations. This allows for partial credit and reduces the penalties on semantically correct but lexically different translations. Moreover, since the output of SimiLe is continuous, it provides more informative gradients during the optimization process by distinguishing between candidates that would be similarly scored under matching-based metrics like BLEU. Lastly, we show in our analysis that SimiLe has an additional benefit over BLEU by translating words with heavier semantic content more accurately.

To define an exact metric, we reference the burgeoning field of research aimed at measuring semantic textual similarity (STS) between two sentences (Le and Mikolov, 2014; Pham et al., 2015; Wieting et al., 2016b; Hill et al., 2016; Conneau et al., 2017; Pagliardini et al., 2017) (Chapter 4). Specifically, we start with the method we introduced in 8, which learns paraphrastic sentence representations using a contrastive loss and a parallel corpus induced by backtranslating bitext. Also, in Chapters 4 and 5 we showed that simple models that average word or character trigram embeddings can be highly effective for semantic similarity. The strong performance, domain robustness, and computationally efficiency of these models make them good candidates for experimenting with incorporating semantic similarity into neural machine translation. For the purpose of discriminative NMT training, we augment these basic models with two modifications: we add a length penalty to avoid short translations, and calculate similarity by composing the embeddings of subword units, rather than words or character trigrams. We find that using subword units also yields better performance on the STS evaluations and is more efficient than character trigrams.

We conduct experiments with our new metric on the 2018 WMT (Bojar et al., 2018) test sets, translating four languages, Czech, German, Russian, and Turkish, into English. Results demonstrate that optimizing SimiLe during training results in not only improvements in the same metric during test, but also in consistent improvements in BLEU. Further, we conduct a human study to evaluate system outputs and find significant improvements in human-judged translation quality for all but one language. Finally, we provide an analysis of our results in order to give insight into the observed gains in performance. Tuning for metrics other than BLEU has not (to our knowledge) been extensively examined for NMT, and we hope this chapter provides a first step towards broader consideration of training metrics for NMT.

13.2 SIMILE REWARD FUNCTION

Since our goal is to develop a continuous metric of sentence similarity, we borrow from a line of work focused on domain agnostic semantic similarity metrics. We motivate our choice for applying this line of work to training translation models in Section 2.1. Then in Section 2.2, we describe how we train our similarity metric (SIM), how we compute our length penalty, and how we tie these two terms together to form SimiLe.

13.2.1 SimiLe

Our SimiLe metric is based on the sentence similarity metric from Chapter 8, which we choose as a starting point because it has state-of-the-art unsupervised performance on a host of domains for semantic textual similarity.[1] Being both unsupervised and domain agnostic provide evidence that the model generalizes well to unseen examples. This is in contrast to supervised methods which are often imbued with the bias of their training data.

MODEL. Our sentence encoder g averages 300 dimensional subword unit[2] embeddings to create a sentence representation. The similarity of two sentences, SIM, is obtained by encoding both with g and then calculating their cosine similarity.

1 In semantic textual similarity the goal is to produce scores that correlate with human judgments on the degree to which two sentences have the same semantics. In embedding based models, including the models used in this paper, the score is produced by the cosine of the two sentence embeddings.

2 We use `sentencepiece` which is available at `https://github.com/google/sentencepiece`. We limited the vocabulary to 30,000 tokens.

Model	2012	2013	2014	2015	2016
SIM (300 dim.)	69.2	60.7	77.0	80.1	**78.4**
SimiLe	**70.1**	59.8	74.7	79.4	77.8
Chapter 8	67.8	**62.7**	**77.4**	**80.3**	78.1
BLEU	58.4	37.8	55.1	67.4	61.0
BLEU (symmetric)	58.2	39.1	56.2	67.8	61.2
METEOR	53.4	47.6	63.7	68.8	61.8
METEOR (symmetric)	53.8	48.2	65.1	70.0	62.7
STS 1st Place	64.8	62.0	74.3	79.0	77.7
STS 2nd Place	63.4	59.1	74.2	78.0	75.7
STS 3rd Place	64.1	58.3	74.3	77.8	75.7

Table 93: Comparison of the semantic similarity model used in this paper (SIM) with a number of strong baselines including our strongest model from 8 and the top 3 performing STS systems for each year. Symmetric refers to taking the average score of the metric with each sentence having a turn in the reference position.

Model	newstest2015	newstest2016
SIM	58.2	53.1
SimiLe	58.4	53.2
BLEU	53.6	50.0
METEOR	**58.9**	**57.2**

Table 94: Comparison of models on machine translation quality evaluation datasets. Scores are in Spearman's ρ.

TRAINING. We follow the procedure from Chapter 8 in learning the parameters of the encoder g. The training data is a set S of paraphrase pairs[3] $\langle s, s'\rangle$ and we use a margin-based loss:

$$\begin{aligned}\ell(s,s') = &\max(0, \delta - \cos(g(s), g(s')) \\ &+ \cos(g(s), g(t)))\end{aligned}$$

where δ is the margin, and t is a *negative example*. The intuition is that we want the two texts to be more similar to each other than to their negative examples. To select t, we choose the most similar sentence in a collection of mini-batches called a *mega-batch*.

Finally, we note that SIM is robust to domain, as shown by its strong performance on the STS tasks which cover a broad range of domains. We note that SIM was trained primarily on subtitles, while we use news data to train and evaluate our NMT models. Despite this

3 We use 16.77 million paraphrase pairs filtered from the ParaNMT corpus from Chapter 8. The corpus is filtered by a sentence similarity score based on the PARAGRAM-PHRASE from Chapter 4 and word trigrams overlap, which is calculated by counting word trigrams in the reference and translation, then dividing the number of shared trigrams by the total number in the reference or translation, whichever has fewer. These form a balance between semantic similarity (similarity score) and diversity (trigram overlap). We kept all sentences in ParaNMT with a similarity score $\geqslant 0.5$ and a trigram overlap score $\leqslant 0.2$. In Chapter 9 we showed that strong performance on semantic similarity tasks can also be achieved using bitext directly without the need for backtranslation.

domain switch, we are able to show improved performance over a baseline using BLEU, providing more evidence of the robustness of this method.

LENGTH PENALTY. Our initial experiments showed that when using just the similarity metric, SIM, there was nothing preventing the model from learning to generate long sentences, often at the expense of repeating words. This is the opposite case from BLEU, where the n-gram precision is not penalized for generating too few words. Therefore, in BLEU, a brevity penalty (BP) was introduced to penalize sentences when they are shorter than the reference. The penalty is:

$$\mathrm{BP}(r, h) = e^{1-\frac{|r|}{|h|}}$$

where r is the reference and h is the generated hypothesis, with $|r|$ and $|h|$ their respective lengths. We experimented with modifying this penalty to only penalize generated sentences that are longer than the target (so we switch r and h in the equation). However, we found that this favored short sentences. We instead penalize a generated sentence if its length differs at all from that of the target. Therefore, our length penalty is:

$$\mathrm{LP}(r, h) = e^{1-\frac{\max(|r|,|h|)}{\min(|r|,|h|)}}$$

SIMILE. Our final metric, which we refer to as SimiLe, is defined as follows:

$$\textsc{SimiLe} = \mathrm{LP}(r, h)^{\alpha}\mathrm{SIM}(r, h)$$

In initial experiments we found that performance could be improved slightly by lessening the influence of LP, so we fix α to be 0.25.

13.2.2 *Motivation*

There is a vast literature on metrics for *evaluating* machine translation outputs automatically (For instance, WMT metrics task papers like Bojar et al. (2017)). In this paper we demonstrate that *training* towards metrics other than BLEU has significant practical advantages in the context of NMT. While this could be done with any number of metrics, in this paper we experiment with a single semantic similarity metric, and due to resource constraints leave a more extensive empirical comparison of other evaluation metrics to future work. That said, we designed SimiLe as a semantic similarity model with high accuracy, domain robustness, and computational efficiency to be used in minimum risk training for machine translation.[4]

While semantic similarity is not an exact replacement for measuring machine translation quality, we argue that it serves as a decent proxy at least as far as minimum risk training is concerned. To test this, we compare the similarity metric term in SimiLe (SIM) to BLEU and METEOR on two machine translation metric [5] and report their correlation with human judgments in Table 94. Machine translation quality measures account for more than semantics as they also capture other factors like fluency. A manual error analysis and the fact that the machine translation correlations in Table 94 are close, but the semantic similarity correlations[6] in Table 93 are not, suggest that the difference between METEOR and SIM largely lies in fluency. However, not capturing fluency is something that can be ameliorated by adding a down-weighted maximum-likelihood (MLE) loss to the minimum risk loss. This was done by Edunov et al. (2018), and we use this in our experiments as well.

13.3 MACHINE TRANSLATION PRELIMINARIES

ARCHITECTURE. Our model and optimization procedure are based on prior work on structured prediction training for neural machine translation (Edunov et al., 2018) and are implemented in Fairseq.[7] Our architecture follows the paradigm of an encoder-decoder with soft attention (Bahdanau et al., 2015) and we use the same architecture for each language pair in our experiments. We use gated convolutional encoders and decoders (Gehring et al., 2017). We use 4 layers for the encoder and 3 for the decoder, setting the hidden state size for all layers to 256, and the filter width of the kernels to 3. We use byte pair encoding (Sennrich et al., 2016c), with a vocabulary size of 40,000 for the combined source and target vocabulary. The dimension of the BPE embeddings is set to 256.

OBJECTIVE FUNCTIONS. Following (Edunov et al., 2018), we first train models with maximum-likelihood with label-smoothing ($\mathcal{L}_{\text{TokLS}}$) (Szegedy et al., 2016; Pereyra et al.,

4 SimiLe, including time to segment the sentence, is about 20 times faster than METEOR when code is executed on a GPU (NVIDIA GeForce GTX 1080).

5 We used the segment level data, where English is the target language, from newstest2015 and newstest2016 available at http://statmt.org/wmt18/metrics-task.html. The former contains 6 language pairs and the latter 4.

6 Evaluation is on the SemEval Semantic Textual Similarity (STS) datasets from 2012-2016 (Agirre et al., 2012, 2013, 2014, 2015, 2016). In the SemEval STS competitions, teams create models that need to work well on domains both represented in the training data and hidden domains revealed at test time. Our model and those from Chapter 8, in contrast to the best performing STS systems, do not use any manually-labeled training examples nor any other linguistic resources beyond the ParaNMT corpus from Chapter 8

7 https://github.com/pytorch/fairseq

2017). We set the confidence penalty of label smoothing to be 0.1. Next, we fine-tune the model with a weighted average of minimum risk training ($\mathcal{L}_{\text{Risk}}$) (Shen et al., 2016) and ($\mathcal{L}_{\text{TokLS}}$), where the expected risk is defined as:

$$\mathcal{L}_{\text{Risk}} = \sum_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} \text{cost}(\mathbf{t}, \mathbf{u}) \frac{p(\mathbf{u}|\mathbf{x})}{\sum_{\mathbf{u}' \in \mathcal{U}(\mathbf{x})} p(\mathbf{u}'|\mathbf{x})}$$

where $\mathbf{u}$ is a candidate hypothesis, $\mathcal{U}(\mathbf{x})$ is a set of candidate hypotheses, and $\mathbf{t}$ is the reference. Therefore, our fine-tuning objective becomes:

$$\mathcal{L}_{\text{Weighted}} = \gamma \mathcal{L}_{\text{TokLS}} + (1 - \gamma) \mathcal{L}_{\text{Risk}}$$

We tune γ from the set $\{0.2, 0.3, 0.4\}$ in our experiments. In minimum risk training, we aim to minimize the expected cost. In our case that is $1 - \text{BLEU}(t, h)$ or $1 - \text{SimiLe}(t, h)$ where t is the target and h is the generated hypothesis. As is commonly done, we use a smoothed version of BLEU by adding 1 to all n-gram counts except unigram counts. This is to prevent BLEU scores from being overly sparse (Lin and Och, 2004). We generate candidates for minimum risk training from n-best lists with 8 hypotheses and do not include the reference in the set of candidates.

OPTIMIZATION. We optimize our models using Nesterov's accelerated gradient method (Sutskever et al., 2013) using a learning rate of 0.25 and momentum of 0.99. Gradients are renormalized to norm 0.1 (Pascanu et al., 2012). We train the $\mathcal{L}_{\text{TokLS}}$ objective for 200 epochs and the combined objective, $\mathcal{L}_{\text{Weighted}}$, for 10. Then for both objectives, we anneal the learning rate by reducing it by a factor of 10 after each epoch until it falls below 10^{-4}. Model selection is done by selecting the model with the lowest validation loss on the validation set. To select models across the different hyperparameter settings, we chose the model with the highest performance on the validation set for the evaluation being considered.

13.4 EXPERIMENTS

13.4.1 *Data*

Training models with minimum risk is expensive, but we wanted to evaluate in a difficult, realistic setting using a diverse set of languages. Therefore, we experiment on four language

Lang.	Train	Valid	Test
`cs-en`	218,384	6,004	2,983
`de-en`	284,286	7,147	2,998
`ru-en`	235,159	7,231	3,000
`tr-en`	207,678	7,008	3,000

Table 95: Number of sentence pairs in the training/validation/test sets for all four languages.

	`de-en`		`cs-en`		`ru-en`		`tr-en`	
Model	BLEU	SIM	BLEU	SIM	BLEU	SIM	BLEU	SIM
MLE	27.52	76.19	17.02	67.55	17.92	69.13	14.47	65.97
BLEU	27.92‡	76.28‡	17.38‡	67.87‡	17.97	69.29‡	15.10‡	66.53‡
SIMILE	**28.56**†‡	**77.52**†‡	**17.60**†‡	**68.89**†‡	**18.44**†‡	**70.69**†‡	**15.47**†‡	**67.76**†‡
Half	28.25†‡	76.92†‡	17.52†‡	68.26†‡	18.26†‡	70.32†‡	15.40†‡	67.14†‡

Table 96: Results on translating four languages to English for MLE, BLEU, SIMILE and Half. † denotes statistical significance ($p < 0.05$) over BLEU and ‡ denotes statistical significance over MLE. Statistical significance was computed using paired bootstrap resampling (Koehn, 2004).

pairs: Czech (`cs-en`), German (`de-en`), Russian (`ru-en`), and Turkish (`tr-en`) translating to English (`en`). For training data, we use News Commentary v13[8] provided by WMT (Bojar et al., 2018) for `cs-en`, `de-en`, and `ru-en`. For training the Turkish system, we used the WMT 2018 parallel data which consisted of the SETIMES2[9] corpus. The validation and development sets for `de-en`, `cs-en`, and `ru-en` were the WMT 2016 and WMT 2017 validation sets. For `tr-en`, the validation set was the WMT 2016 validation set and the WMT 2017 validation and test sets. Test sets for each language were the official WMT 2018 test sets.

13.4.2 *Automatic Evaluation*

We first use corpus-level BLEU and the corpus average SIM score to evaluate the outputs of the different experiments. It is important to note that in this case, SIM is not the same as SIMILE. SIM is only the semantic similarity component of SIMILE and therefore lacks the length penalization term. We used this metric to estimate the degree to which the semantic content of a translation and its reference overlap. When evaluating semantic similarity, we find that SIM outperforms SIMILE marginally as shown in Table 93.

We compare systems trained with 4 objectives:

- MLE: Maximum likelihood with label smoothing

8 `http://data.statmt.org/wmt18/translation-task/training-parallel-nc-v13.tgz`
9 `http://opus.lingfil.uu.se/SETIMES2.php`

- BLEU: Minimum risk training with 1-BLEU as the cost
- SimiLe: Minimum risk training with 1-SimiLe as the cost
- Half: Minimum risk training with a new cost that is half BLEU and half SimiLe: $1 - \frac{1}{2}(\text{BLEU} + \text{SimiLe})$

The results are shown in Table 96. From the table, we see that using SimiLe performs the best when using BLEU and SIM as evaluation metrics for all four languages. It is interesting that using SimiLe in the cost leads to larger BLEU improvements than using BLEU alone, the reasons for which we examine further in the following sections. It is important to emphasize that increasing BLEU was not the goal of our proposed method, human evaluations were our target, but this is a welcome surprise. Similarly, using BLEU as the cost function leads to large gains in SIM, though these gains are not as large as when using SimiLe in training.

13.4.3 *Human Evaluation*

We also perform human evaluation, comparing MLE training with minimum risk training using SimiLe and BLEU as costs. We selected 200 sentences along with their translation from the respective test sets of each language. The sentences were selected nearly randomly with the only constraints that they be between 3 and 25 tokens long and also that the outputs for SimiLe and BLEU were not identical. The translators then assigned a score from 0-5 based on how well the translation conveyed the information contained in the reference. The annotation instructions used by translators are:

- 0. The meaning is completely different or the output is meaningless
- 1. The topic is the same but the meaning is different
- 2. Some key information is different
- 3. The key information is the same but the details differ
- 4. Meaning is essentially equal but some expressions are unnatural
- 5. Meaning is essentially equal and the two sentences are well-formed English

From the results shown in Table 97, we see that minimum risk training with SimiLe as the cost scores the highest across all language pairs except Turkish. Turkish is also the language

	Avg. Score		
Lang.	MLE	BLEU	SimiLe
`cs-en`	0.98	0.90	**1.02**†
`de-en`	0.93	0.85	**1.00**†
`ru-en`	1.22	1.21	**1.31**†‡
`tr-en`	0.98*	**1.03***	0.78

Table 97: Average human ratings on 200 sentences from the test set for each of the respective languages. † denotes statistical significance ($p < 0.05$) over BLEU, except for the case of `cs-en`, where $p = 0.06$. ‡ denotes statistical significance over MLE, and * denotes statistical significance over SimiLe. Statistical significance was computed using paired bootstrap resampling.

with the lowest test BLEU (See Table 96). An examination of the human-annotated outputs shows that in Turkish (unlike the other languages) repetition was a significant problem for the SimiLe system in contrast to MLE or BLEU. We hypothesize that one weakness of SimiLe may be that it needs to start with some minimum level of translation quality in order to be most effective. The biggest improvement over BLEU is on `de-en` and `ru-en`, which have the highest MLE BLEU scores in Table 96 which further lends credence to this hypothesis.

13.5 QUANTITATIVE ANALYSIS

We next analyze our model using the validation set of the `de-en` data unless stated otherwise. We chose this dataset for the analysis since it had the highest MLE BLEU scores of the languages studied.

13.5.1 *Partial Credit*

We analyzed the distribution of the cost function for both SimiLe and BLEU on the `de-en` validation set before any fine-tuning. Again, using an n-best list size of 8, we computed the cost for all generated translations and plotted their histogram in Figure 10. The plots show that the distribution of scores for SimiLe and BLEU are quite different. Both distributions are not symmetrical Gaussian, however the distribution of BLEU scores is significantly more skewed with much higher costs. This tight clustering of costs provides less information during training.

Next, for all n-best lists, we computed all differences between scores of the hypotheses in the beam. Therefore, for a beam size of 8, this results in 28 different scores. We found that

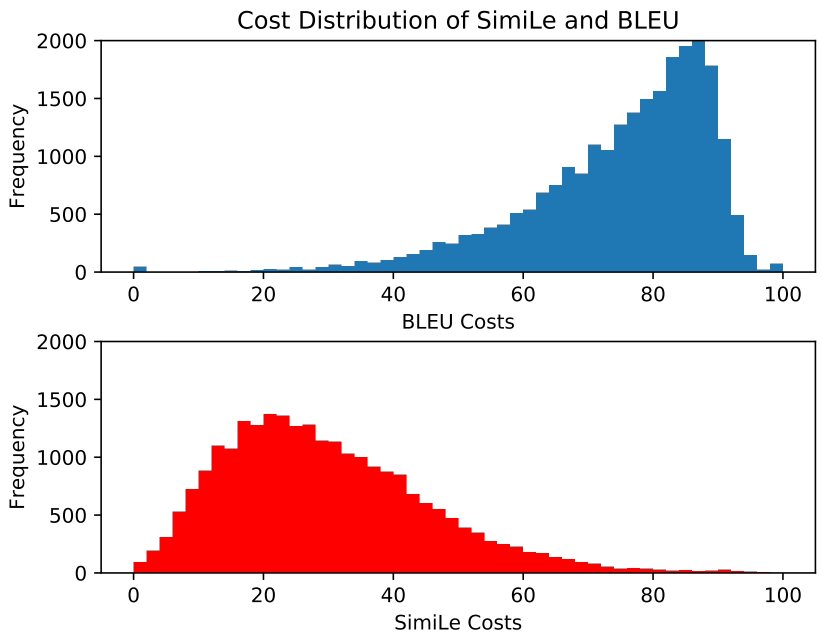


Figure 10: Distribution of scores for SimiLe and BLEU.

of the 86,268 scores, the difference between scores in an n-best list is $\geqslant 0$ 99.0% of the time for SimiLe, but 85.1% of the time for BLEU. The average difference is 4.3 for BLEU and 4.8 for SimiLe, showing that SimiLe makes finer grained distinctions among candidates.

13.5.2 *Validation Loss*

We next analyze the validation loss during training of the `de-en` model for both using SimiLe and BLEU as costs. We use the hyperparameters of the model with the highest BLEU on the validation set for model selection. Since the distributions of costs vary significantly between SimiLe and BLEU, with BLEU having much higher costs on average, we compute the validation loss with respect to both cost functions for each of the two models.

In Figure 11, we plot the risk objective for the first 10 epochs of training. In the top plot, we see that the risk objective for both BLEU and SimiLe decreases much faster when using SimiLe to train than BLEU. The expected BLEU also reaches a significantly lower value on the validation set when training with SimiLe. The same trend occurs in the lower plot, this time measuring the expected SimiLe cost on the validation set.

From these plots, we see that optimizing with SimiLe results in much faster training. It also reaches a lower validation loss, and from Table 96, we've already shown that the SimiLe and BLEU on the test set are higher for models trained with SimiLe. To hammer home

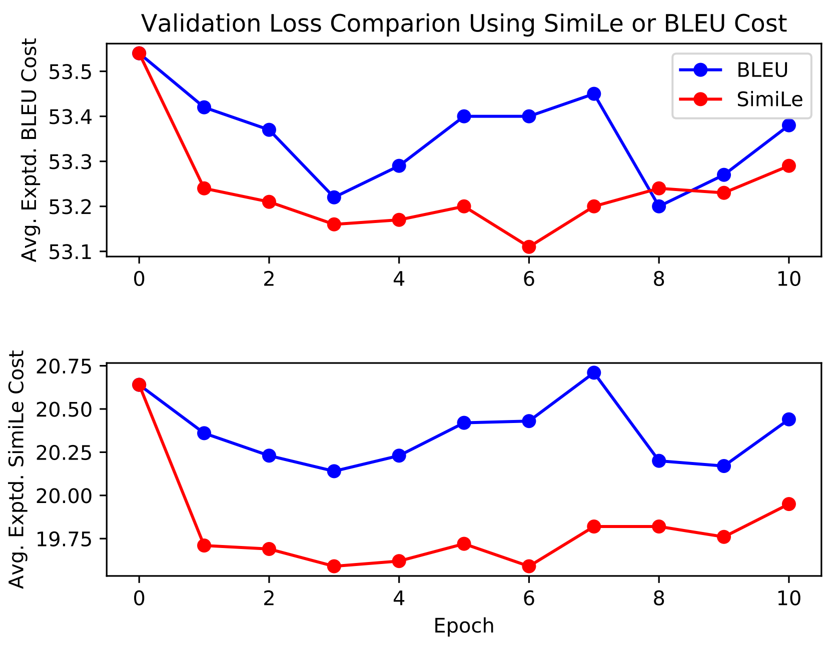


Figure 11: Validation loss comparison for SimiLe and BLEU. The top plot shows the expected BLEU cost when training with BLEU and SimiLe. The bottom plot shows the expected SimiLe cost when training with BLEU and SimiLe.

the point at how much faster the models trained with SimiLe reach better performance, we evaluated after just 1 epoch of training and found that the model trained with BLEU had SIM/BLEU scores of 86.71/27.63 while the model trained with SimiLe had scores of 87.14/28.10. A similar trend was observed in the other language pairs as well, where the validation curves show a much larger drop-off after a single epoch when training with SimiLe than with BLEU.

13.5.3 *Effect of $n$best List Size*

As mentioned in Section 13.3, we used an n-best list size of 8 in our minimum risk training experiments. In this section, we train `de-en` translation models with various n-best list sizes and investigate the relationship between beam size and test set performance when using SimiLe or BLEU as a cost. We hypothesize that since BLEU is not as fine-grained a metric as SimiLe, expanding the number of candidates would close the gap between BLEU and SimiLe as BLEU would have access to a more candidates with more diverse scores. The results of our experiment on the are shown in Figure 12 and show that models trained with SimiLe actually improve in BLEU and SIM more significantly as n-best list size increases. This is possibly due to small n-best sizes inherently upper-bounding performance regardless of

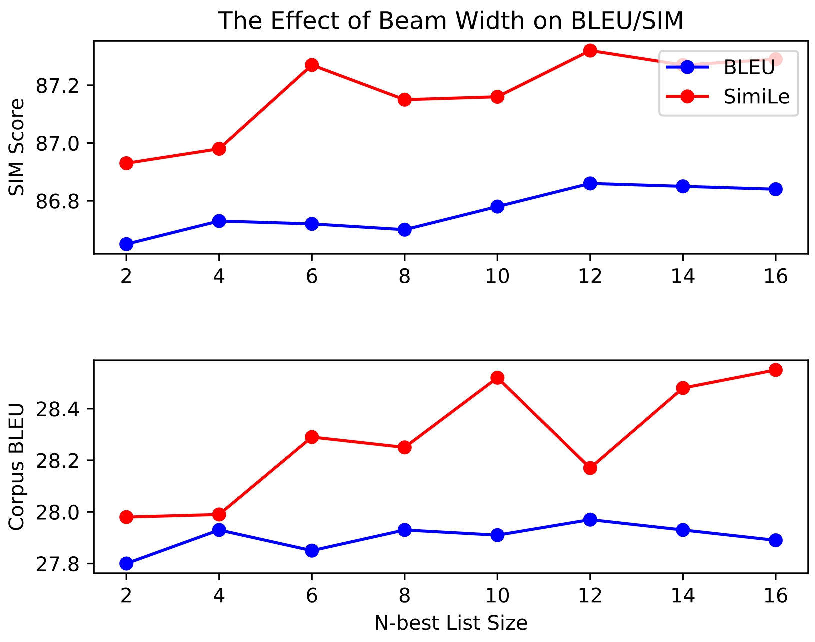


Figure 12: The relationship between n-best list size and performance as measured by average SIM score or corpus-level BLEU when training using SIMILE or BLEU as a cost.

Lang./Bucket	`cs-en` Δ	`de-en` Δ	`ru-en` Δ	`tr-en` Δ	Avg.
1	0.1	0.8	0.2	0.1	0.30
2-5	1.2	0.6	*0.0*	0.2	0.50
6-10	0.4	0.7	1.4	*-0.3*	0.55
11-100	0.2	0.6	0.6	0.4	0.45
101-1000	*-0.3*	0.3	0.4	0.2	0.15
1001+	*-0.2*	0.5	0.4	*-0.0*	0.08
DET	0.1	*-0.1*	0.7	-0.5	0.03
PRON	0.6	*-0.3*	0.1	0.9	0.33
PREP	0.2	*-0.3*	0.5	0.5	0.24
CONJ	0.1	1.1	0.3	-0.5	0.27
PUNCT	*-0.4*	1.3	0.8	-0.4	0.34
NUM	0.6	2.2	1.8	1.3	**1.48**
SYM	0.3	3.6	4.4	1.7	**2.50**
INTJ	3.2	*-1.1*	3.2	-2.6	**0.66**
VERB	0.2	0.3	0.0	*0.0*	0.13
ADJ	0.2	0.7	0.3	*-0.2*	0.25
ADV	*-0.2*	0.1	0.8	0.7	0.34
NOUN	0.3	1.1	0.8	0.4	**0.63**
PRNOUN	0.5	1.2	0.6	0.4	**0.65**

Table 98: Difference in F1 score for various buckets of words. The values in the table are the difference between the F1 obtained when training using SIMILE and when training using BLEU (positive values means SIMILE had a higher F1). The first part of the table shows F1 scores across bins defined by word frequency on the test set. So words appearing only 1 time are in the first row, between 2-5 times are in the second row, etc. The next part of the table buckets words by coarse part-of-speech tags.

Reference	System	Human Score	Translation
I will tell you my personal opinion of him.	BLEU	2	I will have a personal opinion on it.
	SimiLe	4	I will tell my personal opinion about it.
	MLE	2	I will have a personal view of it.
In my case, it was very varied.	BLEU	0	I was very different from me.
	SimiLe	4	For me, it was very different.
	MLE	1	In me, it was very different.
We're making the city liveable.	BLEU	0	We make the City of Life Life.
	SimiLe	3	We make the city viable.
	MLE	0	We make the City of Life.
The head of the White House said that the conversation was ridiculous.	BLEU	0	The White House chairman, the White House chip called a ridiculous.
	SimiLe	4	The White House's head, he described the conversation as ridiculous.
	MLE	1	The White House chief, he called the White House, he called a ridiculous.
According to the former party leaders, so far the discussion "has been predominated by expressions of opinion based on emotions, without concrete arguments".	BLEU	3	According to former party leaders, the debate has so far had to be "elevated to an expression of opinion without concrete arguments."
	SimiLe	5	In the view of former party leaders, the debate has been based on emotions without specific arguments."
	MLE	4	In the view of former party leaders, in the debate, has been based on emotions without specific arguments."
We are talking about the 21st century: servants.	BLEU	4	We are talking about the 21st century: servants.
	SimiLe	1	In the 21st century, the 21st century is servants.
	MLE	0	In the 21st century, the 21st century is servants.
Prof. Dr. Caglar continued:	BLEU	3	They also reminded them.
	SimiLe	0	There are no Dr. Caglar.
	MLE	3	They also reminded them.

Table 99: Translation examples for min-risk models trained with SimiLe and BLEU and our baseline MLE model.

training metric, and SimiLe being a better measure overall when the n-best is sufficiently large to learn.

13.5.4 *Lexical F1*

We next attempt to elucidate exactly which parts of the translations are improving due to using SimiLe cost compared to using BLEU. We compute the F1 scores for target word types based on their frequency and their coarse part-of-speech tag (as labeled by SpaCy[10]) on the test sets for each language and show the results in Table 98.[11]

From the table, we see that training with SimiLe helps produce low frequency words more accurately, a fact that is consistent with the part-of-speech tag analysis in the second part of the table. Wieting and Gimpel (2017) noted that highly discriminative parts-of-

10 `https://github.com/explosion/spaCy`
11 We use `compare-mt` (Neubig et al., 2019) available at `https://github.com/neulab/compare-mt`.

speech, such as nouns, proper nouns, and numbers, made the most contribution to the sentence embeddings. Other works (Pham et al., 2015; Wieting et al., 2016b) have also found that when training semantic embeddings using an averaging function, embeddings that bear the most information regarding the meaning have larger norms. We also see that these same parts-of-speech (nouns, proper nouns, numbers) have the largest difference in F1 scores between SimiLe and BLEU. Other parts-of-speech like symbols and interjections have high F1 scores as well, and words belonging to these classes are both relatively rare and highly discriminative regarding the semantics of the sentence.[12] In contrast, parts-of-speech that in general convey little semantic information and are more common, like determiners, show very little difference in F1 between the two approaches.

13.6 QUALITATIVE ANALYSIS

System	Sentence	BLEU	SIM	ΔBLEU	ΔSIM
Reference	Workers have begun to clean up in Röszke.	-	-	-	-
BLEU	Workers are beginning to clean up workers.	29.15	69.12	-	-
SimiLe	In Röszke, workers are beginning to clean up.	25.97	95.39	-3.18	26.27
Reference	All that stuff sure does take a toll.	-	-	-	-
BLEU	None of this takes a toll.	25.98	54.52	-	-
SimiLe	All of this is certain to take its toll.	18.85	77.20	-7.13	32.46
Reference	Another advantage is that they have fewer enemies.	-	-	-	-
BLEU	Another benefit : they have less enemies.	24.51	81.20	-	-
SimiLe	Another advantage: they have fewer enemies.	58.30	90.76	56.69	9.56
Reference	I don't know how to explain - it's really unique.	-	-	-	-
BLEU	I do not know how to explain it - it is really unique.	39.13	97.42	-	-
SimiLe	I don't know how to explain - it is really unique.	78.25	99.57	39.12	2.15

Table 100: Translation examples where the $|\Delta\text{BLEU}| - |\Delta\text{SIM}|$ statistic is among the highest and lowest in the validation set. The top two rows show examples where the generated sentences have similar sentence-level BLEU scores but quite different SIM scores. The bottom two rows show the converse. Negative values indicate the SimiLe system had a higher score for that sentence.

12 Note that in the data, interjections (INTJ) often correspond to words like *Yes* and *No* which tend to be very important regarding the semantics of the translation in these cases.

We show examples of the output of all three systems in Table 99 from the test sets, along with their human scores which are on a 0-5 scale. The first 5 examples show cases where SimiLe better captures the semantics than BLEU or MLE. In the first three, the SimiLe model adds a crucial word that the other two systems omit. This makes a significant difference in preserving the semantics of the translation. These words include verbs (*tells*), prepositions (*For*), adverbs (*viable*) and nouns (*conversation*). The fourth and fifth examples also show how SimiLe can lead to more fluent outputs and is effective on longer sentences.

The last two examples are failure cases of using SimiLe. In the first, it repeats a phrase, just as the MLE model does and is unable to smooth it out as the BLEU model is able to do. In the last example, SimiLe again tries to include words (*Dr. Caglar*) significant to the semantics of the sentence. However it misses on the rest of translation, despite being the only system to include this noun phrase.

13.7 METRIC COMPARISON

We took all outputs of the validation set of the `de-en` data for our best SimiLe and BLEU models, as measured by BLEU validation scores, and we sorted the outputs by the following statistic:

$$|\Delta\text{BLEU}| - |\Delta\text{SIM}|$$

where BLEU in this case refers to sentence-level BLEU. Examples of some of the highest and lowest scoring sentence pairs are shown in Table 100 along with the system they came from (either trained with a BLEU cost or SimiLe cost).

The top half of the table shows examples where the difference in SIM scores is large, but the difference in BLEU scores is small. From these examples, we see that when SIM scores are very different, there is a difference in the meanings of the generated sentences. However, when the BLEU scores are very close, this is not the case. In fact, in these examples, less accurate translations have higher BLEU scores than more accurate ones. In the first sentence, an important clause is left out (*in Röszke*) and in the second, the generated sentence from the BLEU system actually negates the reference, despite having a higher BLEU score than the sentence from the SimiLe system.

Conversely, the bottom half of the table shows examples where the difference in BLEU scores is large, but the difference in SIM scores is small. From these examples, we can see that when BLEU scores are very different, the semantics of the sentence can still be preserved. However, the SIM score of these generated sentences with the references are close to each other, as we would hope to see. These examples illustrate a well-known problem with BLEU where synonyms, punctuation changes, and other small deviations from the reference can have a large impact on the score. As can be seen from the examples, these are less of a problem for the SIM metric.

13.8 RELATED WORK

The seminal work on training machine translation systems to optimize particular evaluation measures was performed by Och (2003), who introduced minimum error rate training (MERT) and used it to optimize several different metrics in statistical MT (SMT). This was followed by a large number of alternative methods for optimizing machine translation systems based on minimum risk (Smith and Eisner, 2006a), maximum margin (Watanabe et al., 2007), or ranking (Hopkins and May, 2011), among many others.

Within the context of SMT, there have also been studies on the stability of particular metrics for optimization. Cer et al. (2010) compared several metrics to optimize for SMT, finding BLEU to be robust as a training metric and finding that the most effective and most stable metrics for training are not necessarily the same as the best metrics for automatic evaluation. The WMT shared tasks included tunable metric tasks in 2011 (Callison-Burch et al., 2011) and again in 2015 (Stanojević et al., 2015) and 2016 (Jawaid et al., 2016). In these tasks, participants submitted metrics to optimize during training or combinations of metrics and optimizers, given a fixed SMT system. The 2011 results showed that nearly all metrics performed similarly to one another. The 2015 and 2016 results showed more variation among metrics, but also found that BLEU was a strong choice overall, echoing the results of Cer et al. (2010). We have shown that our metric stabilizes training for NMT more than BLEU, which is a promising result given the limited success of the broad spectrum of previous attempts to discover easily tunable metrics in the context of SMT.

Some researchers have found success in terms of improved human judgments when training to maximize metrics other than BLEU for SMT. Lo et al. (2013) and Beloucif et al. (2014) trained SMT systems to maximize variants of MEANT, a metric based on semantic roles. Liu et al. (2011) trained systems using TESLA, a family of metrics based on softly matching n-grams using lemmas, WordNet synsets, and part-of-speech tags. We have demonstrated that our metric similarly leads to gains in performance as assessed by human annotators, and our method has an auxiliary advantage of being much simpler than these previous hand-engineered measures.

Shen et al. (2016) explored minimum risk training for NMT, finding that a sentence-level BLEU score led to the best performance even when evaluated under other metrics. These results differ from the usual results obtained for SMT systems, in which tuning to optimize a metric leads to the best performance on that metric (Och, 2003). Edunov et al. (2018) compared structured losses for NMT, also using sentence-level BLEU. They found risk to be an effective and robust choice, so we use risk as well in this paper.

13.9 CONCLUSION

We have proposed SimiLe, an alternative to BLEU for use as a reward in minimum risk training. We have found that SimiLe not only outperforms BLEU on automatic evaluations, it correlates better with human judgments as well. Our analysis also shows that using this metric eases optimization and the translations tend to be richer in correct, semantically important words.

This is the first time to our knowledge that a continuous metric of semantic similarity has been proposed for NMT optimization and shown to outperform sentence-level BLEU, and we hope that this can be the starting point for more research in this direction.

14 CONCLUSION

This thesis describes our work in paraphrastic representations and their applications. We have discussed multiple approaches for learning these embeddings, at the sub-word, word, and sentence level. We also explored the cross-lingual setting, showing that our approaches are effective for measuring similarity between sentences of different languages. We then proposed our final models, the cross-lingual probabilistic source separation model BGT and the multilingual probabilistic source separation model MGT. We show that the BGT is the first deep architecture that generalizes better than simple methods for semantic similarity tasks. We then generalized this BGT to the multilingual setting showing how it can be effectively trained to accommodate many languages at once. Lastly, we discussed applications of these works including the first large (more than 50 millions examples) paraphrasing corpus, syntactically controlled paraphrase generation, and semantic similarity rewards for fine-tuning neural machine translation systems that outperform the prior convention (using BLEU).

There are many avenues for future work. For learning representations of sentences, none of the models presented in this thesis make use of large contextual models or incorporate monolingual text in any substantial way. Since monolingual text is more ubiquitous than parallel data and is easier to obtain in more domains, this would be a fruitful avenue to explore. Language model pretraining has shown to learn much about syntax, discourse, and semantics – however since it only has a single view of the data, it likely has a weaker picture of semantics than models trained with multiple views. This is similar to what has occurred with word embeddings, where they were trained on monolingual task, but conflated *relatedness* and *similarity*, and could be greatly improved at paraphrastic similarity using data with two views 3. There are many ways monolingual data could be incorporated, from simply using pretrained models to using objectives that account for both monolingual and parallel data.

Another interesting avenue for learning sentence embeddings is designing new pretraining objectives for monolingual text. In the previous paragraph, we discussed incorporating language models as pretrained models, but other objectives that focus more on the sentence level could lead to even more significant gains in downstream tasks. One example of s possible route here is a generative model for discourse where we aim to have latent variables that captures information for the next and previous sentences. Then the information shared between these views could be seen as encoding the semantics of the central sentence – similar to the BGT, where as the next-sentence and previous-sentence variable encode only the specific information for generating the next and previous sentences. This approach could be extended to predict multiple sentences in a long discourse.

Paraphrase generation is another area where there is a lot of potential for impactful future work. In Chapter 12, we introduced controlled paraphrase generation via pre-specifying the desired syntactic structure of the paraphrase. We found this to be effective for increasing the robustness of NLP models and even lead to some small improvements through data augmentation. Recently in preliminary experiments, we have shown that leveraging large pre-trained language models such as GPT-2 (Radford et al., 2019), can be trained on selected *diverse* paraphrases to lead to a *diverse* paraphraser. This *diverse* paraphraser can then be used to generate paraphrases with vastly different lexical choice and syntactic structure. While this work is currently in its early stages, we think these *diverse* paraphraser models hold huge promise for tasks like style transfer (since they can *normalize* the text prior to generating in a desired style) and data augmentation and robustness. Open questions for this line of research include how to select and automatically obtain the training data, what type of pre-trained models should be used, and also what other strategies can be used to increase paraphrase diversity. For instance, perhaps new diverse decoding techniques using paraphrase embeddings would be an effective strategy.

Lastly, in 13, we proposed using our paraphrase embeddings as a reward for fine-tuning neural machine translation models, and found it to be highly effective in terms of improved accuracy and convergence speed, even on state-of-the-art Transformer models. We view this as just the start for this line of research as our reward was generated from just a simple averaging model. While some new evaluation metrics have been recently such as BertScore (Zhang et al., 2019a) which use a matching algorithm built on BERT, these have a weakness where they focus on detecting sentences that deviate little from the reference and are not strong paraphrase detection models when there is a lot of diversity in the surface forms of the sentences. Perhaps the rich embedding approaches we propose above could lead to better rewards that can be tuned for more paraphrastic similarity (useful for style transfer or paraphrase generation) versus the more literal type of paraphrase detection used by BertScore and related approaches (useful for machine translation evaluation). Evaluation metrics and reward functions are closely aligned and the NLP field is in need of both. New distributional rewards could also bring progress to difficult generation tasks like generating coherent multi-paragraph text and for generating output more human-like outputs.

BIBLIOGRAPHY

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics.

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, German Rigau, Larraitz Uria, and Janyce Wiebe. 2015. SemEval-2015 task 2: Semantic textual similarity, English, Spanish and pilot on interpretability. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*.

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2014. SemEval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*.

Eneko Agirre, Carmen Banea, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2016. SemEval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. *Proceedings of SemEval*, pages 497–511.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. * sem 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, volume 1, pages 32–43.

Eneko Agirre, Mona Diab, Daniel Cer, and Aitor Gonzalez-Agirre. 2012. SemEval-2012 task 6: A pilot on semantic textual similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*. Association for Computational Linguistics.

Andrei Alexandrescu and Katrin Kirchhoff. 2006. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*.

Ion Androutsopoulos and Prodromos Malakasiotis. 2010. A survey of paraphrasing and textual entailment methods. *Journal of Artificial Intelligence Research*.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *Proceedings of the International Conference on Learning Representations*.

Mikel Artetxe and Holger Schwenk. 2018a. Margin-based parallel corpus mining with multilingual sentence embeddings. *arXiv preprint arXiv:1811.01136*.

Mikel Artetxe and Holger Schwenk. 2018b. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *arXiv preprint arXiv:1812.10464*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*.

Collin F Baker, Charles J Fillmore, and John B Lowe. 1998. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Colin Bannard and Chris Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Marco Baroni, Raffaela Bernardi, and Roberto Zamparelli. 2014. Frege in space: A program of compositional distributional semantics. *Linguistic Issues in Language Technology*, 9.

Marco Baroni and Roberto Zamparelli. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Regina Barzilay and Lillian Lee. 2003. Learning to paraphrase: an unsupervised approach using multiple-sequence alignment. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Regina Barzilay and Kathleen R McKeown. 2001. Extracting paraphrases from a parallel corpus. In *Proceedings of the 39th annual meeting on Association for Computational Linguistics*, pages 50–57.

Yonatan Belinkov and James Glass. 2015. Arabic diacritization with recurrent neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Meriem Beloucif, Chi-kiu Lo, and Dekai Wu. 2014. Improving MEANT based semantically tuned SMT. In *Proceedings of 11th International Workshop on Spoken Language Translation (IWSLT 2014)*.

Islam Beltagy, Stephen Roller, Pengxiang Cheng, Katrin Erk, and Raymond J. Mooney. 2015. Representing meaning with a combination of logical form and vectors. *arXiv preprint arXiv:1505.06816*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of ACL*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python*. O'Reilly Media, Inc.

Johannes Bjerva, Johan Bos, Rob van der Goot, and Malvina Nissim. 2014. The meaning factory: Formal semantics for recognizing textual entailment and determining semantic similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*.

William Blacoe and Mirella Lapata. 2012. A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, et al. 2017. Findings of the 2017 conference on machine translation (wmt17). In *Proceedings of the Second Conference on Machine Translation*, pages 169–214.

Ondřej Bojar, Ondřej Dušek, Tom Kocmi, Jindřich Libovický, Michal Novák, Martin Popel, Roman Sudarikov, and Dušan Variš. 2016. Czeng 1.6: enlarged czech-english parallel corpus with processing tools dockered. In *International Conference on Text, Speech, and Dialogue*, pages 231–238. Springer.

Ondřej Bojar, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Philipp Koehn, and Christof Monz. 2018. Findings of the 2018 conference on machine translation (wmt18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 272–303. Association for Computational Linguistics.

Igor Bolshakov and Alexander Gelbukh. 2004. Synonymous paraphrasing using WordNet and Internet. *Natural Language Processing and Information Systems*.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014a. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Antoine Bordes, Jason Weston, and Nicolas Usunier. 2014b. Open question answering with weakly supervised embedding models. In *Proceedings of the 2014th European Conference on Machine Learning and Knowledge Discovery in Databases-Volume Part I*, pages 165–180. Springer-Verlag.

Wauter Bosma and Chris Callison-Burch. 2007. Paraphrase substitution for recognizing textual entailment. In *Proceedings of the 7th International Conference on Cross-Language Evaluation Forum: Evaluation of Multilingual and Multi-modal Information Retrieval*.

Jan A. Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. In *Proceedings of ACL*.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

Elia Bruni, Nam-Khanh Tran, and Marco Baroni. 2014. Multimodal distributional semantics. *Journal of Artificial Intelligence Research*, 49:1–47.

Chris Callison-Burch. 2008. Syntactic constraints on paraphrases extracted from parallel corpora. In *Proceedings of Empirical Methods in Natural Language Processing*.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Omar Zaidan. 2011. Findings of the 2011 workshop on statistical machine translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 22–64. Association for Computational Linguistics.

Chris Callison-Burch, Philipp Koehn, and Miles Osborne. 2006. Improved statistical machine translation using paraphrases. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 17–24, New York City, USA.

Michel Carl, Paul Schmidt, and Jörg Schütz. 2005. Reversible template-based shake & bake generation. In *MT Summit X*.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 Task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada.

Daniel Cer, Christopher D. Manning, and Daniel Jurafsky. 2010. The best lexical metric for phrase-based statistical mt system optimization. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 555–563. Association for Computational Linguistics.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Chih-Chung Chang and Chih-Jen Lin. 2011. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3).

Mingda Chen, Qingming Tang, Sam Wiseman, and Kevin Gimpel. 2019. A multi-task approach for disentangling syntax and semantics in sentence representations. *arXiv preprint arXiv:1904.01173*.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Pengfei Liu, and Xuanjing Huang. 2015a. Long short-term memory neural networks for Chinese word segmentation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Shiyu Wu, and Xuanjing Huang. 2015b. Sentence modeling with gated recursive neural network. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Xinxiong Chen, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huanbo Luan. 2015c. Joint learning of character and word embeddings. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Muthuraman Chidambaram, Yinfei Yang, Daniel Cer, Steve Yuan, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Learning cross-lingual sentence representations via a multi-task dual-encoder model. *arXiv preprint arXiv:1810.12836*.

Grzegorz Chrupała. 2013. Text segmentation with character-level text embeddings. *arXiv preprint arXiv:1309.4628*.

Grzegorz Chrupała. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 2016. A character-level decoder without explicit segmentation for neural machine translation. *arXiv preprint arXiv:1603.06147*.

Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29.

Alexander Clark. 2003. Combining distributional and morphological information for part of speech induction. In *10th Conference of the European Chapter of the Association for Computational Linguistics*.

Paul Clough and Mark Stevenson. 2011. Developing a corpus of plagiarised short answers. *Language resources and evaluation*, 45(1):5–24.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12.

Alexis Conneau and Douwe Kiela. 2018. Senteval: An evaluation toolkit for universal sentence representations. *arXiv preprint arXiv:1803.05449*.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark.

Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018a. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. *arXiv preprint arXiv:1805.01070*.

Alexis Conneau, Guillaume Lample, Ruty Rinott, Adina Williams, Samuel R Bowman, Holger Schwenk, and Veselin Stoyanov. 2018b. Xnli: Evaluating cross-lingual sentence representations. *arXiv preprint arXiv:1809.05053*.

Marta R. Costa-Jussà and José A. R. Fonollosa. 2016. Character-based neural machine translation. *arXiv preprint arXiv:1603.00810*.

William Coster and David Kauchak. 2011. Simple English Wikipedia: a new text simplification task. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 665–669.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6).

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Bill Dolan, Chris Quirk, and Chris Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of COLING*.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Li Dong, Jonathan Mallinson, Siva Reddy, and Mirella Lapata. 2017. Learning to paraphrase for question answering. In *Proceedings of Empirical Methods in Natural Language Processing*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12.

Myroslava O. Dzikovska, Johanna D. Moore, Natalie Steinhauser, Gwendolyn Campbell, Elaine Farrow, and Charles B. Callaway. 2010. Beetle II: a system for tutoring and computational linguistics experimentation. In *Proceedings of the ACL 2010 System Demonstrations*.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for nlp. *arXiv preprint arXiv:1712.06751*.

Sergey Edunov, Myle Ott, Michael Auli, David Grangier, et al. 2018. Classical structured prediction losses for sequence to sequence learning. In *Proceedings of NAACL*, volume 1, pages 355–364.

Andreas Eisele and Yu Chen. 2010. Multiun: A multilingual corpus from united nation documents. In *LREC*.

Amr El-Desoky Mousa, Hong-Kwang Jeff Kuo, Lidia Mangu, and Hagen Soltau. 2013. Morpheme-based feature-rich language models using deep neural networks for LVCSR of Egyptian Arabic. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE.

Cristina Espana-Bonet, Adám Csaba Varga, Alberto Barrón-Cedeño, and Josef van Genabith. 2017. An empirical analysis of nmt-derived interlingual embeddings and their use in parallel sentence identification. *IEEE Journal of Selected Topics in Signal Processing*, 11(8):1340–1350.

Allyson Ettinger, Ahmed Elgohary, and Philip Resnik. 2016. Probing for semantic evidence of composition by means of simple classification tasks. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 134–139, Berlin, Germany. Association for Computational Linguistics.

Allyson Ettinger, Sudha Rao, Hal Daumé III, and Emily M Bender. 2017. Towards linguistically generalizable nlp systems: A workshop and shared task. In *Proceedings of the First Workshop on Building Linguistically Generalizable NLP Systems*.

Kilian Evang, Valerio Basile, Grzegorz Chrupała, and Johan Bos. 2013. Elephant: Sequence labeling for word and sentence segmentation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Manaal Faruqui and Chris Dyer. 2014. Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 462–471.

Manaal Faruqui and Chris Dyer. 2015. Non-distributional word vector representations. *arXiv preprint arXiv:1506.05230*.

Song Feng, Ritwik Banerjee, and Yejin Choi. 2012. Characterizing stylistic elements in syntactic structure. In *Proceedings of Empirical Methods in Natural Language Processing*.

Jessica Ficler and Yoav Goldberg. 2017. Controlling linguistic style aspects in neural language generation. *arXiv preprint arXiv:1707.02633*.

Katja Filippova, Enrique Alfonseca, Carlos A. Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*. ACM.

J.R. Firth. 1957. *A Synopsis of Linguistic Theory, 1930-1955*.

Zhe Gan, Yunchen Pu, Ricardo Henao, Chunyuan Li, Xiaodong He, and Lawrence Carin. 2017. Learning generic sentence representations using convolutional neural networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2380–2390, Copenhagen, Denmark.

Juri Ganitkevitch and Chris Callison-Burch. 2014. The multilingual paraphrase database. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. PPDB: The Paraphrase Database. In *Proceedings of HLT-NAACL*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. 2003. Learning precise timing with LSTM recurrent networks. *The Journal of Machine Learning Research*, 3.

Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. 2016. Simverb-3500: A large-scale evaluation set of verb similarity. *arXiv preprint arXiv:1608.00869*.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations*.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Alex Graves, Marcus Liwicki, Horst Bunke, Jürgen Schmidhuber, and Santiago Fernández. 2008. Unconstrained on-line handwriting recognition with recurrent neural networks. In *Advances in neural information processing systems*, pages 577–584.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2015. LSTM: A search space odyssey. *arXiv preprint arXiv:1503.04069*.

Francis Grégoire and Philippe Langlais. 2018. Extracting parallel sentences with bidirectional recurrent neural networks to improve machine translation. *arXiv preprint arXiv:1806.05559*.

Mandy Guo, Qinlan Shen, Yinfei Yang, Heming Ge, Daniel Cer, Gustavo Hernandez Abrego, Keith Stevens, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Effective parallel corpus mining using bilingual sentence embeddings. *arXiv preprint arXiv:1807.11906*.

Ankush Gupta, Arvind Agarwal, Prawaan Singh, and Piyush Rai. 2017. A deep generative framework for paraphrase generation. *arXiv preprint arXiv:1709.05074*.

Kelvin Guu, Tatsunori B Hashimoto, Yonatan Oren, and Percy Liang. 2017. Generating sentences by editing prototypes. *arXiv preprint arXiv:1709.08878*.

Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka. 2014. Jointly learning word representations and composition functions using predicate-argument structures. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Hua He, Kevin Gimpel, and Jimmy Lin. 2015. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Junxian He, Daniel Spokoyny, Graham Neubig, and Taylor Berg-Kirkpatrick. 2019. Lagging inference networks and posterior collapse in variational autoencoders. *arXiv preprint arXiv:1901.05534*.

Kenneth Heafield. 2011. KenLM: faster and smaller language model queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pages 187–197.

Benjamin Heinzerling and Michael Strube. 2018. Bpemb: Tokenization-free pre-trained subword embeddings in 275 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Karl Moritz Hermann and Phil Blunsom. 2014. Multilingual models for compositional distributed semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*.

Felix Hill, Kyunghyun Cho, Sebastien Jean, Coline Devin, and Yoshua Bengio. 2014a. Embedding word similarity with neural machine translation. *arXiv preprint arXiv:1412.6448*.

Felix Hill, KyungHyun Cho, Sebastien Jean, Coline Devin, and Yoshua Bengio. 2014b. Not all neural embeddings are born equal. *arXiv preprint arXiv:1410.0718*.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Felix Hill, Roi Reichart, and Anna Korhonen. 2015. SimLex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4).

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8).

Mark Hopkins and Jonathan May. 2011. Tuning as ranking. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362. Association for Computational Linguistics.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014a. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems*.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014b. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050.

J Edward Hu, Rachel Rudinger, Matt Post, and Benjamin Van Durme. 2019. Parabank: Monolingual bitext generation and sentential paraphrasing via lexically-constrained neural machine translation. *arXiv preprint arXiv:1901.03644*.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM.

Ozan Irsoy and Claire Cardie. 2014. Deep recursive neural networks for compositionality in language. In *Advances in neural information processing systems*, pages 2096–2104.

Mohit Iyyer, Jordan Boyd-Graber, and Hal Daumé III. 2014. Generating sentences from semantic vector space representations. In *NIPS Workshop on Learning Semantics*.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Bushra Jawaid, Amir Kamran, Miloš Stanojević, and Ondřej Bojar. 2016. Results of the wmt16 tuning shared task. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 232–238. Association for Computational Linguistics.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10.

Yacine Jernite, Samuel R Bowman, and David Sontag. 2017. Discourse-based objectives for fast unsupervised sentence representation learning. *arXiv preprint arXiv:1705.00557*.

Yangfeng Ji and Jacob Eisenstein. 2013. Discriminative improvements to distributional sentence similarity. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 891–896, Seattle, Washington, USA.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of Empirical Methods in Natural Language Processing*.

Youxuan Jiang, Jonathan K. Kummerfeld, and Walter S. Lasecki. 2017. Understanding task design trade-offs in crowdsourced paraphrase collection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 103–109, Vancouver, Canada.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *CoRR*, abs/1602.02410.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2015. Character-aware neural language models. *CoRR*, abs/1508.06615.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in Neural Information Processing Systems 28*, pages 3294–3302.

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics.

Reinhard Kneser and Hermann Ney. 1993. Forming word classes by statistical clustering for statistical language modelling. In *Contributions to Quantitative Linguistics*, pages 221–226. Springer.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 388–395.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180.

Stanley Kok and Chris Brockett. 2010. Hitting the right paraphrases in good time. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Alice Lai and Julia Hockenmaier. 2014. Illinois-LH: A denotational and distributional approach to semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 329–334, Dublin, Ireland.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.

Wuwei Lan, Siyu Qiu, Hua He, and Wei Xu. 2017a. A continuously growing dataset of sentential paraphrases. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1224–1234, Copenhagen, Denmark.

Wuwei Lan, Siyu Qiu, Hua He, and Wei Xu. 2017b. A continuously growing dataset of sentential paraphrases. In *Proceedings of EMNLP*, Copenhagen, Denmark. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Angeliki Lazaridou, Marco Marelli, Roberto Zamparelli, and Marco Baroni. 2013. Compositional-ly derived representations of morphologically complex words in distributional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*.

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185.

Jiwei Li, Michel Galley, Chris Brockett, Georgios Spithourakis, Jianfeng Gao, and Bill Dolan. 2016a. A persona-based neural conversation model. In *Proceedings of the Association for Computational Linguistics*.

Jiwei Li and Dan Jurafsky. 2015. Do multi-sense embeddings improve natural language understanding? *arXiv preprint arXiv:1506.01070*.

Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. 2015. A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057*.

Jiwei Li, Will Monroe, and Dan Jurafsky. 2016b. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics-Volume 1*, pages 1–7.

Zichao Li, Xin Jiang, Lifeng Shang, and Hang Li. 2017. Paraphrase generation with deep reinforcement learning. *arXiv preprint arXiv:1711.00279*.

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. 2017. Deep text classification can be fooled. *arXiv preprint arXiv:1704.08006*.

Chin-Yew Lin and Franz Josef Och. 2004. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of the Conference on Computational Linguistics*, pages 501–507.

Dekang Lin. 1998. Automatic retrieval and clustering of similar words. In *COLING 1998 Volume 2: The 17th International Conference on Computational Linguistics*.

Dekang Lin and Patrick Pantel. 2001. Discovery of inference rules for question answering. *Natural Language Engineering*, 7(4):342–360.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015a. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Wang Ling, Isabel Trancoso, Chris Dyer, and Alan W. Black. 2015b. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*.

Pierre Lison and Jörg Tiedemann. 2016. Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. European Language Resources Association.

Patrick Littell, David R. Mortensen, Ke Lin, Katherine Kairis, Carlisle Turner, and Lori Levin. 2017. Uriel and lang2vec: Representing languages as typological, geographical, and phylogenetic vectors. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 8–14. Association for Computational Linguistics.

Chang Liu, Daniel Dahlmeier, and Hwee Tou Ng. 2011. Better evaluation metrics lead to better machine translation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 375–384, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Pengfei Liu, Xipeng Qiu, Xinchi Chen, Shiyu Wu, and Xuanjing Huang. 2015. Multi-timescale long short-term memory neural network for modelling sentences and documents. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Chi-kiu Lo, Karteek Addanki, Markus Saers, and Dekai Wu. 2013. Improving machine translation by training against an automatic semantic frame based evaluation metric. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 375–381. Association for Computational Linguistics.

Ang Lu, Weiran Wang, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2015. Deep multilingual correlation for improved word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv preprint arXiv:1604.00788*.

Thang Luong, Richard Socher, and Christopher Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. 2019. FlowSeq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4273–4283, Hong Kong, China. Association for Computational Linguistics.

Nitin Madnani and Bonnie J. Dorr. 2010. Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36(3):341–387.

Jonathan Mallinson, Rico Sennrich, and Mirella Lapata. 2017. Paraphrasing revisited with neural machine translation. In *Proceedings of the European Chapter of the Association for Computational Linguistics*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics System Demonstrations*.

Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. 2014. SemEval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*.

Yuval Marton, Chris Callison-Burch, and Philip Resnik. 2009. Improved statistical machine translation using monolingually-derived paraphrases. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6297–6308.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

Kathleen R McKeown. 1983. Paraphrasing questions using given and new information. *Computational Linguistics*, 9(1).

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

George A Miller and Walter G Charles. 1991. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive Science*, 34(8).

Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Counter-fitting word vectors to linguistic constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–148, San Diego, California.

Preslav Nakov, Francisco Guzman, and Stephan Vogel. 2012. Optimizing for sentence-level BLEU+1 yields short translations. In *Proceedings of COLING 2012*, pages 1979–1994.

Courtney Napoles, Chris Callison-Burch, and Matt Post. 2016. Sentential paraphrasing as black-box machine translation. In *Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*.

Graham Neubig, Zi-Yi Dou, Junjie Hu, Paul Michel, Danish Pruthi, Xinyi Wang, and John Wieting. 2019. compare-mt: A tool for holistic comparison of language generation systems. In *Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL) Demo Track*, Minneapolis, USA.

Allen Nie, Erin D Bennett, and Noah D Goodman. 2017. Dissent: Sentence representation learning from explicit discourse relations. *arXiv preprint arXiv:1710.04334*.

Xing Niu, Marianna Martindale, and Marine Carpuat. 2017. A study of style in machine translation: Controlling the formality of machine translation output. In *Proceedings of Empirical Methods in Natural Language Processing*.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*.

Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. 2017. Unsupervised learning of sentence embeddings using compositional n-gram features. *arXiv preprint arXiv:1703.02507*.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106.

Bo Pang, Kevin Knight, and Daniel Marcu. 2003. Syntax-based alignment of multiple translations: Extracting paraphrases and generating new sentences. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 102–109, Edmonton, Canada.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 271–278, Barcelona, Spain.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 115–124, Ann Arbor, Michigan.

Denis Paperno, Nghia The Pham, and Marco Baroni. 2014. A practical and linguistically-motivated approach to compositional distributional semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*.

Ellie Pavlick and Chris Callison-Burch. 2016. Simple ppdb: A paraphrase database for simplification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 143–148.

Ellie Pavlick, Pushpendre Rastogi, Juri Ganitkevich, Benjamin Van Durme, and Chris Callison-Burch. 2015. PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of Empirical Methods in Natural Language Processing*.

Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. 2017. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237.

Nghia The Pham, Germán Kruszewski, Angeliki Lazaridou, and Marco Baroni. 2015. Jointly optimizing word representations for lexical and sentential tasks with the c-phrase model. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Mohammad Taher Pilehvar, Dimitri Kartsaklis, Victor Prokhorov, and Nigel Collier. 2018. Card-660: Cambridge rare word dataset-a reliable benchmark for infrequent word representation models. *arXiv preprint arXiv:1808.09308*.

Tamara Polajnar, Laura Rimell, and Stephen Clark. 2015. An exploration of discourse-based sentence spaces for compositional distributional semantics. In *Proceedings of the First Workshop on Linking Computational Models of Lexical, Sentential and Discourse-level Semantics*.

Martin Popel and Ondřej Bojar. 2018. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70.

Aaditya Prakash, Sadid A Hasan, Kathy Lee, Vivek Datla, Ashequl Qadir, Joey Liu, and Oladimeji Farri. 2016. Neural paraphrase generation with stacked residual LSTM networks. In *Proceedings of International Conference on Computational Linguistics*.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Co-learning of word representations and morpheme representations. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*.

Chris Quirk, Chris Brockett, and William Dolan. 2004. Monolingual machine translation for paraphrase generation. In *Proceedings of Empirical Methods in Natural Language Processing*.

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. 2017. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremb. 2016. Sequence level training with recurrent neural networks. In *Proceedings of the 4th International Conference on Learning Representations*.

Pushpendre Rastogi and Benjamin Van Durme. 2014. Augmenting FrameNet via PPDB. In *Proceedings of the Second Workshop on EVENTS: Definition, Detection, Coreference, and Representation*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Fabio Rinaldi, James Dowdall, Kaarel Kaljurand, Michael Hess, and Diego Mollá. 2003. Exploiting paraphrases in a question answering system. In *Proceedings of the Second International Workshop on Paraphrasing*.

Herbert Rubenstein and John B Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

Gerard Salton, Anita Wong, and Chung-Shu Yang. 1975. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Cicero dos Santos and Victor Guimarães. 2015. Boosting named entity recognition with neural character embeddings. In *Proceedings of the Fifth Named Entity Workshop*.

Cicero dos Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*.

Karin Kipper Schuler. 2005. Verbnet: a broad-coverage, comprehensive verb lexicon.

Hinrich Schütze. 1993. Word space. In *Advances in neural information processing systems*, pages 895–902.

Roy Schwartz, Roi Reichart, and Ari Rappoport. 2015. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*.

Holger Schwenk. 2018. Filtering and mining parallel data in a joint multilingual space. *arXiv preprint arXiv:1805.09822*.

Holger Schwenk and Matthijs Douze. 2017. Learning joint multilingual sentence representations with neural machine translation. *arXiv preprint arXiv:1704.04154*.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the Association for Computational Linguistics*.

Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. 2017. Nematus: a toolkit for neural machine translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 65–68, Valencia, Spain.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Controlling politeness in neural machine translation via side constraints. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016c. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

Yang Shao. 2017. HCTI at SemEval-2017 task 1: Use convolutional neural network to evaluate semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 130–133.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2016. Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1683–1692. Association for Computational Linguistics.

Siwei Shen, Dragomir R Radev, Agam Patel, and Güneş Erkan. 2006. Adding syntax to dynamic programming for aligning comparable texts for the generation of paraphrases. In *Proceedings of International Conference on Computational Linguistics*.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. In *Proceedings of Advances in Neural Information Processing Systems*.

Karan Singla, Dogan Can, and Shrikanth Narayanan. 2018. A multi-task approach to learning multilingual representations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 214–220.

David A. Smith and Jason Eisner. 2006a. Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 787–794. Association for Computational Linguistics.

David A Smith and Jason Eisner. 2006b. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings of the Workshop on Statistical Machine Translation*.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Richard Socher, Andrej Karpathy, Quoc V. Le, Christopher D. Manning, and Andrew Y. Ng. 2014. Grounded compositional semantics for finding and describing images with sentences. *TACL*, 2.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

Radu Soricut and Franz Och. 2015. Unsupervised morphology induction using word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Lucia Specia. 2011. Exploiting objective annotations for measuring translation post-editing effort. In *Proceedings of the 15th conference of the European Association for Machine Translation*, pages 73–80.

Henning Sperr, Jan Niehues, and Alex Waibel. 2013. Letter n-gram-based input encoding for continuous space language models. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1).

Miloš Stanojević, Amir Kamran, and Ondřej Bojar. 2015. Results of the wmt15 tuning shared task. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 274–281. Association for Computational Linguistics.

James H Steiger. 1980. Tests for comparing elements of a correlation matrix. *Psychological Bulletin*, 87(2).

Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J Pal. 2018. Learning general purpose distributed sentence representations via large scale multi-task learning. *arXiv preprint arXiv:1804.00079*.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Yui Suzuki, Tomoyuki Kajiwara, and Mamoru Komachi. 2017. Building a non-trivial paraphrase corpus using multiple machine translation systems. In *Proceedings of ACL 2017,* *Student Research Workshop*, pages 36–42, Vancouver, Canada. Association for Computational Linguistics.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Junfeng Tian, Zhiheng Zhou, Man Lan, and Yuanbin Wu. 2017. ECNU at SemEval-2017 task 1: Leverage kernel-based traditional NLP features and neural networks to build a universal model for multilingual and cross-lingual semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 191–197, Vancouver, Canada. Association for Computational Linguistics.

Ran Tian, Naoaki Okazaki, and Kentaro Inui. 2015. The mechanism of additive composition. *arXiv preprint arXiv:1511.08407*.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in opus. In *Lrec*, volume 2012, pages 2214–2218.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

Peter D Turney. 2001. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *European conference on machine learning*, pages 491–502. Springer.

Peter D Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2014. Grammar as a foreign language. *arXiv preprint arXiv:1412.7449*.

Di Wang, Nebojsa Jojic, Chris Brockett, and Eric Nyberg. 2017. Steering output style and topic in neural response generation. In *Proceedings of Empirical Methods in Natural Language Processing*.

Di Wang and Eric Nyberg. 2015. A long short-term memory model for answer sentence selection in question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2007. Online large-margin training for statistical machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Jason Weston, Samy Bengio, and Nicolas Usunier. 2010. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning*, 81(1).

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39(2):165–210.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016a. Charagram: Embedding words and sentences via character n-grams. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1504–1515.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016b. Towards universal paraphrastic sentence embeddings. In *Proceedings of the International Conference on Learning Representations*.

John Wieting, Mohit Bansal, Kevin Gimpel, Karen Livescu, and Dan Roth. 2015. From paraphrase database to compositional paraphrase model and back. *Transactions of the Association for Computational Linguistics*.

John Wieting, Taylor Berg-Kirkpatrick, Kevin Gimpel, and Graham Neubig. 2019a. Beyond bleu: Training neural machine translation with semantic similarity. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4344–4355.

John Wieting and Kevin Gimpel. 2017. Revisiting recurrent networks for paraphrastic sentence embeddings. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2078–2088, Vancouver, Canada.

John Wieting and Kevin Gimpel. 2018. ParaNMT-50M: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 451–462. Association for Computational Linguistics.

John Wieting, Kevin Gimpel, Graham Neubig, and Taylor Berg-Kirkpatrick. 2019b. Simple and effective paraphrastic similarity from parallel translations. In *Proceedings of ACL*.

John Wieting and Douwe Kiela. 2019. No training required: Exploring random encoders for sentence classification. *arXiv preprint arXiv:1901.10444*.

John Wieting, Jonathan Mallinson, and Kevin Gimpel. 2017. Learning paraphrastic sentence embeddings from back-translated bitext. In *Proceedings of EMNLP*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas. Association for Computational Linguistics.

Kristian Woodsend and Mirella Lapata. 2011. Learning to simplify sentences with quasi-synchronous grammar and integer programming. In *Proceedings of Empirical Methods in Natural Language Processing*.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015a. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*.

Wei Xu, Chris Callison-Burch, and William B Dolan. 2015b. SemEval-2015 task 1: Paraphrase and semantic similarity in Twitter (PIT). In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval)*.

Wei Xu, Alan Ritter, Chris Callison-Burch, William B. Dolan, and Yangfeng Ji. 2014. Extracting lexically divergent paraphrases from Twitter. *Transactions of the Association for Computational Linguistics*, 2:435–448.

Yan Xu, Lili Mou, Ge Li, Yunchuan Chen, Hao Peng, and Zhi Jin. 2015c. Classifying relations via long short term memory networks along shortest dependency paths. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. 2019a. Paws-x: A cross-lingual adversarial dataset for paraphrase identification. *arXiv preprint arXiv:1908.11828*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019b. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763.

Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. 2017. Improved variational autoencoders for text modeling using dilated convolutions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3881–3890. JMLR. org.

Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. 2013. Semi-markov phrase-based monolingual alignment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Wen-tau Yih, Kristina Toutanova, John C. Platt, and Christopher Meek. 2011. Learning discriminative projections for text similarity measures. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*.

Wenpeng Yin and Hinrich Schütze. 2015. Convolutional neural network for paraphrase identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Mo Yu and Mark Dredze. 2015. Learning composition models for phrase embeddings. *Transactions of the Association for Computational Linguistics*, 3.

Fabio Massimo Zanzotto, Ioannis Korkontzelos, Francesca Fallucchi, and Suresh Manandhar. 2010. Estimating linear models for compositional distributional semantics. In *Proceedings of the 23rd International Conference on Computational Linguistics*.

Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019a. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Xiang Zhang and Yann LeCun. 2015. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*.

Yuan Zhang, Jason Baldridge, and Luheng He. 2019b. Paws: Paraphrase adversaries from word scrambling. *arXiv preprint arXiv:1904.01130*.

Han Zhao, Zhengdong Lu, and Pascal Poupart. 2015. Self-adaptive hierarchical sentence model. In *Proceedings of IJCAI*.

Shiqi Zhao, Xiang Lan, Ting Liu, and Sheng Li. 2009. Application-driven statistical paraphrase generation. In *Proceedings of the Association for Computational Linguistics*.

Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. 2016. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. In *Proceedings of COLING*, pages 3485–3495, Osaka, Japan.

Zachary Ziegler and Alexander Rush. 2019. Latent normalizing flows for discrete sequences. In *International Conference on Machine Learning*, pages 7673–7682.

Pierre Zweigenbaum, Serge Sharoff, and Reinhard Rapp. 2018. Overview of the third bucc shared task: Spotting parallel sentences in comparable corpora. In *Proceedings of 11th Workshop on Building and Using Comparable Corpora*, pages 39–42.

A
APPENDIX

A.1 LOCATION OF SENTENCE EMBEDDING IN DECODER FOR LEARNING REPRESENTATIONS

As mentioned in Section 10.2.1, we experimented with 4 ways to incorporate the sentence embedding into the decoder: Word, Hidden, Attention, and Logit. We also experimented with combinations of these 4 approaches. We evaluate these embeddings on the STS tasks and show the results, along with the time to train the models 1 epoch in Table 101.

For these experiments, we train a single layer bidirectional LSTM (BiLSTM) Eng. Trans. model with embedding size set to 1024 and hidden states set to 512 dimensions (in order to be roughly equivalent to our Transformer models). To form the sentence embedding in this variant, we mean pool the hidden states for each time step. The cell states of the decoder are initialized to the zero vector.

Architecture	STS	Time (s)
BiLSTM (Hidden)	54.3	1226
BiLSTM (Word)	67.2	1341
BiLSTM (Attention)	68.8	1481
BiLSTM (Logit)	69.4	1603
BiLSTM (Word + Hidden)	67.3	1377
BiLSTM (Word + Hidden + Attention)	68.3	1669
BiLSTM (Word + Hidden + Logit)	69.1	1655
BiLSTM (Word + Hidden + Attention + Logit)	68.9	1856

Table 101: Results for different ways of incorporating the sentence embedding in the decoder for a BiLSTM on the Semantic Textual Similarity (STS) datasets, along with the time taken to train the model for 1 epoch. Performance is measured in Pearson's $r \times 100$.

From this analysis, we see that the best performance is achieved with Logit, when the sentence embedding is place just prior to the softmax. The performance is much better than Hidden or Hidden+Word used in prior work. For instance, recently (Artetxe and Schwenk, 2018b) used the Hidden+Word strategy in learning multilingual sentence embeddings.

A.1.1 *VAE Training*

We also found that incorporating the latent code of a VAE into the decoder using the Logit strategy increases the mutual information while having little effect on the log likelihood. We trained two LSTM VAE models following the settings and aggressive training strategy in (He et al., 2019), where one LSTM model used the Hidden strategy and the other used the

Hidden + Logit strategy. We trained the models on the en side of our `en-fr` data. We found that the mutual information increased form 0.89 to 2.46, while the approximate negative log likelihood, estimated by importance weighting, increased slightly from 53.3 to 54.0 when using Logit.

A.2 relationship between batch size and performance for transformer and lstm

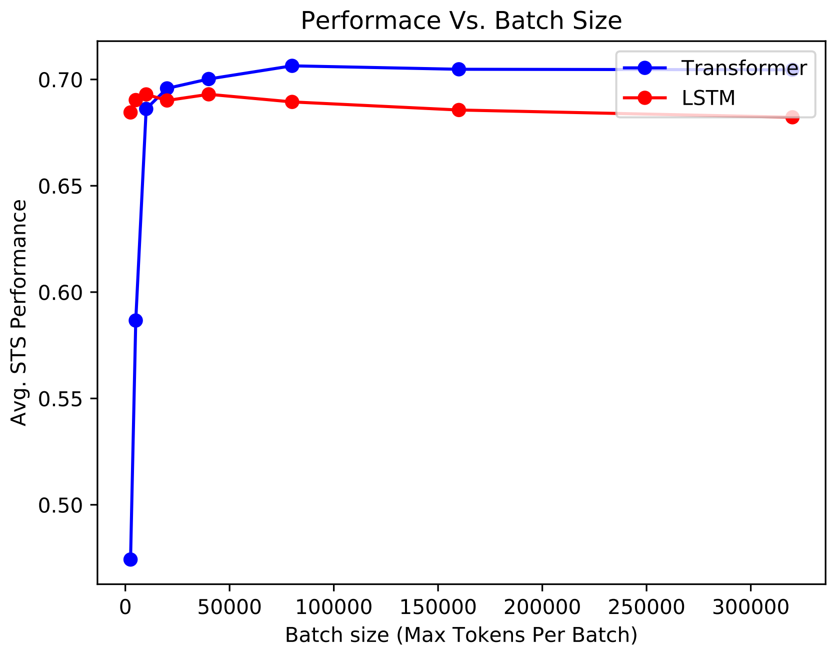


Figure 13: The relationship between average performance for each year of the STS tasks 2012-2016 (Pearson's $r \times 100$) and batch size (maximum number of words per batch).

It has been observed previously that the performance of Transformer models is sensitive to batch size (Popel and Bojar, 2018). We found this to be especially true when training sequence-to-sequence models to learn sentence embeddings. Figure 13 shows plots of the average 2012-2016 STS performance of the learned sentence embedding as batch size increases for both the BiLSTM and Transformer. Initially, at a batch size of 2500 tokens, sentence embeddings learned are worse than random, even though validation perplexity does decrease during this time. Performance rises as batch size increases up to around 100,000 tokens. In contrast, the BiLSTM is more robust to batch size, peaking much earlier around 25,000 tokens, and even degrading at higher batch sizes.

Architecture	STS	Time (s)
Transformer (5L/1L)	70.3	1767
Transformer (3L/1L)	70.1	1548
Transformer (1L/1L)	70.0	1244
Transformer (5L/5L)	69.8	2799

Table 102: Results on the Semantic Textual Similarity (STS) datasets for different configurations of ENG. TRANS., along with the time taken to train the model for 1 epoch. (XL/YL) means X layers were used in the encoder and Y layers in the decoder. Performance is measured in Pearson's $r \times 100$.

A.3 MODEL ABLATIONS

In this section, we vary the number of layers in the encoder and decoder in BGT W/O PRIOR. We see that performance increases as the number of encoder layers increases, and also that a large decoder hurts performance, allowing us to save training time by using a single layer. These results can be compared to those in Table 102 showing that Transformers outperform BiLSTMS in these experiments.