Linguistic Knowledge for Neural Language Generation and Machine Translation

Austin Matthews

CMU-LTI-19-012

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:

Chris Dyer
Graham Neubig
Alon Lavie
Jonathan May

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
In Language and Information Technologies

LINGUISTIC KNOWLEDGE FOR NEURAL LANGUAGE GENERATION AND MACHINE TRANSLATION

AUSTIN MATTHEWS

Ph.D. Thesis

Language Technologies Institute
Carnegie Mellon University

October 16th 2019

ABSTRACT

Recurrent neural networks (RNNs) are exceptionally good models of distributions over natural language sentences, and they are deployed in a wide range of applications that require the generation of natural language outputs. However, RNNs are general-purpose function learners that, given sufficient capacity, are capable of representing any distribution, whereas the space of possible natural languages is narrowly constrained. Linguistic theory has been concerned with characterizing these constraints, with a particular eye toward explaining the uniformity with which children acquire their first languages, despite receiving relatively little linguistic input. This thesis uses insights from linguistic theory to inform the neural architectures and generation processes used to model natural language, seeking models that make more effective use of limited amounts of training data. Since linguistic theories are incomplete, a central goal is developing models that are able to exploit explicit linguistic knowledge while still retaining the generality and flexibility of the neural network models they augment.

This thesis examines two linguistic domains: word formation and sentence structure. First, in the word formation domain, we introduce a language model that captures subword word formation using linguistic knowledge about morphological processes via finite state analyzers hand-crafted by linguistic experts. Our model is capable of using several levels of granularity, including the raw word-, character- and morpheme-levels to encode and condition on previous words as well as to construct its predicted next word. As a result, it is fully open vocabulary, capable of producing any token admitted by a language's alphabet. These properties make it ideal for modelling languages with potentially unbounded vocabulary size, such as Turkish and Finnish.

Second, in the sentence structure domain, we present a pair of dependency-based language models, leveraging syntactic theories that construct sequences of words as the outputs of hierarchical branching processes. Our models construct syntax trees either top-down or bottom-up, jointly learning language modelling and parsing. We find that these dependency-based models make good parsers, but that dependencies are less effective than phrase-structure trees for modelling language.

Finally, again looking at sentence structure, we investigate the application of syntax to the task of conditional language modelling, where data scarcity exacerbates the need for more sample efficient models. We develop a fully neural tree-to-tree translation system, leveraging syntax in both the source and target languages to do conditional language modelling. We then ablate the model, demonstrating the effects of a source-side syntax-based encoder and a target-side syntax-based decoder separately. We find that source-side syntax to hold good promise, and show that inference under neural models is trapped in a local optimum wherein biased models perversely synergize with poor inference procedures. This interaction means improvements in modelling and in decoding algorithms do not necessarily lead to improved quality metrics.

These models demonstrate the effectiveness of hybrid techniques that marry the expressive power of neural networks with explicit linguistic structure derived from human analyses. This synergy allows models to be both more sample efficient and more interpretable.

ACKNOWLEDGEMENTS

I feel incredibly fortunate to have spent the past several years at CMU, surrounded by incredibly brilliant and amazing people, doing research that combines my passions for computer science and linguistics. This thesis would not have been possible without all the friends, family, and teachers that supported me along the way.

My advisor, Chris Dyer, taught me so much and showed me just how much more there is to learn. At work he is an inspiration and outside, a friend. Alon Lavie has now landed me two amazing jobs in the NLP space and is the reason I initially got into machine translation. Graham Neubig advised me on two continents and helped me transition into the new neural world. Jon May agreed to be on my committee last minute (through no fault of his own) and offered valuable insights and feedback throughout the final stages of this work.

Additionally, I would like to thank my internship mentors Jonathan Clark, Stephen Clark, and Jörg Tiedemann, who inspired me in new directions and exposed me to the wider research world.

My education has come at the hands of so, so many amazing teachers and professors. In particular I would like to thank Cheryl Love, who encouraged me to broaden my educational horizons beyond what I thought possible, John Morrison, who fostered my love of computer science, and Fabian Monrose, who started me down the path of academic research and got me into NLP.

I could not have gotten here without the support of my friends Max Burstyn, Kartik Goyal, Greg Hanneman, and Alan Vangpat, who kept me going through the decidedly non-convex emotional landscape that accompanies a PhD. I also thank my coworkers Manaal Faruqui, Gaurav Kumar, Igor Labutov, and Marco Vetter who always listened when I was frustrated and generally made the office an enjoyable environment.

I extend a very special thanks to Yubin Kim who, in addition to giving me her unwavering love and support, has helped guide me through thesis writing, job hunting, and the pursuit of happiness.

Finally, I thank my parents who have been behind me every step of this thirty-year journey. I could never ask for more love and support. You all are the best!

CONTENTS

1

INTRODUCTION

Language generation, the task of having a computer automatically *output* text in a natural language (such as English) is a crucial step towards natural human–computer interaction. It is a essential part of many NLP tasks including summarization, image captioning, dialogue systems, and translation. Most existing methods (with a few notable exceptions, e.g. byte-pair encoding (Sennrich et al., 2015)) for language generation treat text as a linear sequence of atomic words. Such methods model a sentence (or document, etc.) as a string of words that is generated stochastically from left to right and they do not attempt to understand or break down words into any smaller pieces.

Humans, on the other hand, seem to generate sentences in a much more hierarchical manner (Chomsky and Lightfoot, 2002). For example, a typical English sentence might be composed of a subject, a verb phrase, and optionally a direct object. A subject may itself be composed into e.g. a determiner, a series of zero or more adjectives, and a noun, among other possibilities. The verb phrase may similarly be composed of one or more auxiliary verbs, a main verb, and zero or more adverbs. This hierarchical decomposition is central to the infinite expressive power of human language, and our ability as humans to both generate and understand novel sentences, the content of which we have never heard before. Linguists call the study of these hierarchical structures *syntax*, and the structures themselves *syntax trees*, an example of which is shown in Figure 1.

The leaves of a syntax tree are typically words, yet it is often possible to further decompose words into smaller pieces called *morphemes* (Haspelmath and Sims, 2013). For example, the word "denuclearization" can be broken down into the morpheme sequence de+nuclear+ize+ation. The study of such analyses of sub-word units is called *morphology*. Understanding these smaller components of words allows humans to create and understand entirely new words on the fly. For computer models it also allows for much greater statistical sharing. For example, a model should not have to separately learn that "eat", "eats", "eating", "ate", and "eaten" all have something to do with food, nor that (almost) all words that end in *-ation* are nouns. If a model treats words as non-decomposable fixed entities such abstractions are not possible, but morphologically aware models allow words

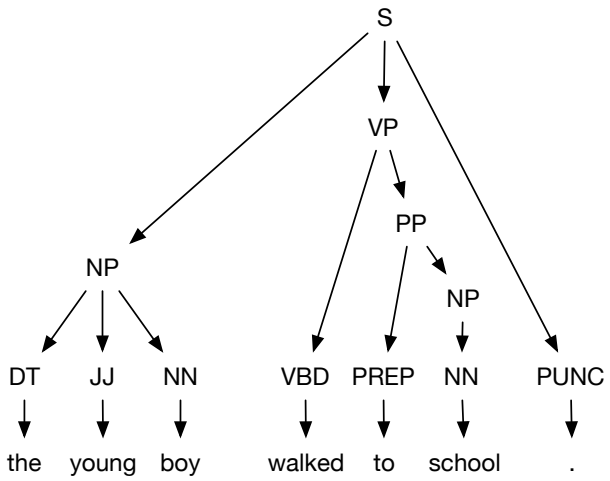


Figure 1: An example of an English syntax tree.

to share knowledge learned about their morphemes. Furthermore, such models have an *open vocabulary* – they can understand and emit rare words, never before seen by the model, like "unacrimoniousness" or "drinkability", in much the same way that a human can.

Model architectures encode biases, desired or undesired, that affect the distributions they learn. In theory, if given infinite data, language models should be able to overcome these biases and converge to a distribution that perfectly models human language. Unfortunately human language is capable of producing an infinity of utterances and has complexity far exceeding the limits of any finite data set. Humans, of course, consistently learn language from a finite amount of exposure, indicating that our minds share a set of biases for language learning (Chomsky, 2014). As such our models should ideally encode the same set of biases that humans hold so that their biases help, rather than hinder, convergence to models that generalize in the same way as humans do. Incorporating syntactic knowledge into our model architectures has been shown to aid this type of generalization, improving fluency of language models (Kuncoro et al., 2018; Maillard et al., 2019).

The task of (natural) language modelling is to be able to generate novel sentences in a way that appears human-like. A good language model is one that produces fluent, natural-sounding output. This requires, of course, that output sentences be grammatical and correctly handle phenomena like subject-verb agreement. In addition, we desire that the *distribution* of sentences and sub-sentential processes matches what a human might produce. For example, certain grammatical structures are extremely common (e.g. passive voice) while others only occur very infrequently (e.g. subjunctive mood). Moreover, humans invent new words all the time. While a few new words stick around to be added to our formal dictionaries, most of these are casual one-off words, used and then forgotten. For

example, one might discuss the "thesisiness" of this work, or ask if a recent film has been "blurayed" yet. A good language model captures these types of phenomena and is able to use them to recombine smaller pieces to form novel words, structures, and sentences.

Unconditional language models underpin technologies such as grammatical error detection (Kann et al., 2018) and spelling correction (Brill and Moore, 2000). More recently they have frequently been used to provide representations for other natural language tasks (Devlin et al., 2019; Peters et al., 2018; Yang et al., 2019). Language modelling offers a convenient and controlled sandbox in which to explore new techniques. Once settled upon a successful model, the learned representations can be applied to other tasks or the language models can be transformed into conditional models to directly complete a down stream task.

Many applications of text generation are *conditional*, indicating that they receive ("condition on") some existing information in order to choose what sentence to output. For example, a dialogue system might condition on what the interlocutors have said during previous turns, an image captioning system should obviously look at the image whose caption it is to generate, and a machine translation system should pay attention to the input ("source") sentence when generating its translation. Neural models make this type of external conditioning very easy. At each time step, instead of looking only at the generator's current state (typically a vector which encodes all the information it has previously generated) we also look at a vector embedding of all or part (e.g. by using attention) of the input data. By modelling the next generation event probability as a function of both these vectors both kinds of context are used.

But why should our models emulate humans' methods of language generation? Our primary motivation is one of sample efficiency. RNNs are great general function learners and, given infinite data, a simple left-to-right RNN model with atomic words could, in theory, learn to produce fluent language. Human children, however, are able to learn their native language fluently with just tens of millions of words of input (Hart and Risley, 2003). Large quantities of monolingual text data are available in many languages (the 2019 WMT data[1] contains over 60 **billion** words of English), but even with so much data available no models yet exist that can understand and generate language at the level of humans. Furthermore, many problems reliant on text generation have much more limited data sets. For example, the 2018 WMT English–Turkish translation task bitext provides only 4.4 million words of English data, necessitating either much more sample efficient learning methods,

1 http://www.statmt.org/wmt19/

methods to transfer knowledge from systems trained on monolingual data, or both. Linguistic theory gives us insights into the processes that underpin human languages, and incorporating this knowledge into our models can help restrict the hypothesis space, leading to faster learning and more human-like output.

In this thesis we explore methods to incorporate morphological and syntactic knowledge into unconditional neural models of language. Furthermore, we examine the effect of incorporating syntax into machine translation, a conditional language modelling task. We present methods for incorporating syntax on the source side, the target side, or both, yielding the first published neural quasi-synchronous translation model. These methods contribute an overall theme and lead to the following thesis statement: **Explicit linguistic structure can be used to design neural architectures that learn more rapidly and make better generalizations than linguistically naïve architectures.**

The remainder of this thesis is structured as follows. In Chapter 2 we present background knowledge essential to understanding the new models and results presented in this thesis. We review morphology and syntax in detail before moving on to formally describe the tasks of language modelling and machine translation.

In Chapter 3 we explore a method to incorporate morphological knowledge in a text generation system. Most prior work simply ignores morphology, treating each surface-form word as a unique token. One notable exception, Byte-Pair Encoding (Sennrich et al., 2015, BPE), does not make any attempt to capture the linguistic processes that form words, instead relying on surface-level character n-gram statistics (Papli, 2017). Furthermore, BPE-based vocabularies are not truly open — there remain word forms and morphological variants unproducable by such sytems. Our morphologically-aware language model, however, explicitly captures linguistic knowledge encoded in morphological analyzers to correctly segment text and is able to produce any output string possible within a language's alphabet.

In our morphology-aware model, we generate text left-to-right, as in most RNN-based models. At each step, we choose the next word conditioned on all the previously generated words. Unlike most previous work, however, we create embeddings for each word in the conditioning context by combining word-, morpheme-, and character-level representations instead of relying on embeddings of words as atomic units. We use an LSTM over the context to generate a single context vector, from which we will predict the next word. The model may then choose to generate the next word via one of three methods: as a single atomic word, as a sequence of morphemes, or as a sequence of characters. We marginal-

ize the probability of generating a given word under each of these three methods to get a single final probability for each candidate word. The resulting mixture model has the learnability advantages of being morphologically aware, and is also *open vocabulary* – it is able to produce any word possible given a language's alphabet, including novel word forms that were not present in its training data. These theoretical advantages also show empirical improvements in language model perplexity on three morphologically rich languages that are traditionally difficult for standard language models, as well as improvements in English-to-Turkish statistical machine translation.

In Chapter 4 we explore a method to incorporate syntactic knowledge into a language model. Previous work (Dyer et al., 2016) has shown that constituency structures help language modelling and parsing performance and that the resulting model also functions as a high-quality parser. We explore a complementary question: does dependency-based syntax similarly help language model performance? We operationalize the process of building a parse tree, converting it into a series of actions. These actions affect a stack of partially built dependency structures, which in the end will contain exactly one item: a fully constructed parse tree for the output sentence. At each time step our model examines the current stack and predicts the next action to take. We propose two models that build trees using different *construction orders*. The first builds the tree top-down, starting from the root and ending with the terminals. The second constructs the tree bottom-up, starting with the terminals and iteratively building tree structure atop them. These two models impose different hierarchical biases on the learner. Information that may be local in the partial tree structures built by one model may be farther away, and thus more difficult for the neural network to exploit, than in structures built by the other. We find that the two resulting models both exhibit competitive parsing performance but do not outperform baseline language models. Furthermore, we find that despite the wildly different biases, the two models perform very similarly, implying that they have learned different ways of capturing similar information.

In Chapter 5 we turn to conditional language modelling and seek to incorporate the syntax-based ideas of Eriguchi et al. (2016) and Dyer et al. (2016) into a tree-to-tree neural translation system. The former follows Tai et al. (2015) in embedding a source syntax tree, rather than relying solely on the flat text representation. The latter describes a tree-structured language-model, able to better capture grammatically of a sentence than a linear language model can. These additions mirror extensions to traditional statistical machine translation, including *quasi-synchronous* (Smith and Eisner, 2006) tree-to-tree models and *factored models* (Koehn and Hoang, 2007). These extensions to symbolic MT systems often

improved quality but were brittle and often involved hard combinatorial search problems. By incorporating them into a neural system we both simplify them and increase their power. We explore the performance of this syntax-aware system on translation from several typologically diverse languages into English and perform an ablation to uncover which particular modules are most responsible for translation quality gains.

We find that syntax can be helpful to translation, particularly on the source side. Furthermore we show that improvements to model design together with improvements in inference algorithms do not necessarily translate into improved quality metrics (e.g. BLEU (Papineni et al., 2002) or METEOR (Denkowski and Lavie, 2014)). We attribute this to the existence of a perverse conspiracy between neural models and greedy inference algorithms. Stahlberg and Byrne (2019) show that virtually all neural translation models have an extreme bias towards shorter hypotheses. In order to have our systems output translations with high metric scores we must purposefully use extremely limited decoding algorithms (Koehn and Knowles, 2017). This interaction renders decoding for MT very unstable — neither improvements in modelling nor improvements in search algorithms ensure gains on translation metrics. We observe this effect on our tree-based decoders. RNNGs improve sample complexity and models' ability to capture natural language distributions, and we introduce decoding algorithms with demonstrably fewer search errors, yet observe no gains in metric scores.

The major contributions of this thesis are as follows: (i) a morphologically-aware open-vocabulary language model optionally leveraging hand-crafted morphological analyzers (ii) two dependency-based language models for generation and parsing without independence assumptions (iii) a neural quasi-synchronous tree-to-tree translation system (iv) empirical comparisons of string-to-string, string-to-tree, tree-to-string, and tree-to-tree models across several languages (v) a novel batching algorithm for encoding syntax trees (vi) two new decoding algorithms for decoding with syntax-aware decoders (vii) analysis of the instability that arises from the interaction between neural models and inference algorithms.

2

BACKGROUND

In this chapter we will review some background that will be helpful in understanding the body of this thesis.

MORPHOLOGY

Morphology is the linguistic study of how words are formed from smaller parts. For example, the English noun "dogs" is composed of two parts: *dog* and *s*, which indicates the plural. These parts are called *morphemes*. Some words consist of only a single morpheme. For example "cat" cannot be further broken down into *ca+t* or *c+at*. Other words, such as "unimaginatively" (*un+imagine+itive + ly*) are composed of many morphemes. Morphemes may be recombined in novel ways to create entirely new words. One may describe their commute as too "traffic-lighty", an android's behavior as "unrobotlike", or themselves as a "antitechnologian". Such words are immediately understandable by any English speaker, even if he or she has never heard them before.

Most other languages have much more rich morphology than does English. In English most verbs have five forms (e.g. eat, eats, eating, ate, eaten), and most nouns have four (e.g. cat, cats, cat's, cats'). In Finnish, on the other hand, verbs have at least twelve tenses/moods, each of which conjugates for person and number, to ninety or more unique surface forms. Nouns also change their form, using fifteen differents endings to encode information that English encodes with prepositions. Languages across the globe, from Hungarian to Tamil, Turkish to Japanese, and German to Inuktitut use constantly use their wildly productive morphology to coin new words. As such, proper handling of morphology is a must for any language technology system for these languages since their vocabularies are of practically infinite size.

There are three ways that morphemes combine to form new words. Inflectional morphology is the process that allows "eat" and "-s" to combine to form "eats". Inflection combines a meaning-bearing stem, and an affix with no referential meaning, into a new word. The new word has the same part of speech as the root word, and its meaning is

fundamentally the same as that of the stem, but the affix provides some sort of syntactic specification, such as the number, person, case, or tense. Derivational morphology similarly combines a stem with an affix, but this time the affix effects a change in meaning and possibly part of speech. For example, "happy" (an adjective) and "-ness" combine to yield "happiness" (a noun). Compounding is the process that allows two meaning-bearing morphemes to combine into a novel word. For example "dish" and "washer" combine to give "dishwasher".

Languages fall into one of three types, according to the structure of their morphology. First, isolating (or analytic) languages have little to no morphological derivation at all. Words are usually composed of just one morpheme (i.e. a free standing root). Functions that are carried by morphology in other languages are instead performed by either adding independent words or by strict word order. Second, agglutinative languages have many morphemes per word. Each morpheme will handle one grammatical function, and multiple affix morphemes can be stacked on one root, yielding very long words. Third, fusional languages have fewer affixes, but each one may carry more than one grammatical meaning. For example, the "-s" verb suffix in English simultaneously specifies that the verb is singular, third person, and present tense.

Examples of these three language types can seen in Table 1. Notice that each word in Chinese translates to a single English word with no morphological annotations at all. In Japanese, on the other hand, each word contains multiple morphemes. This creates sentences with few words, but with each word being quite long. Finally in Russian each content word is marked with exactly one morphological suffix. Each suffix, however, indicates both the case and number of its root. For example, the very short suffix *-a* on the word *zloumyshlennik* ("attacker") indicates both genitive case and singular number (hence "of the attacker").

Most NLP models assume the existance of an atomic word and a fixed vocabulary. This assumption works well in isolating languages where words are indeed more or less atomic. For other languages, however, such models are forced to independently learn the meaning of each word. For example, such a naïve model of Russian would be unaware of the relation between the words *motiv* ("motive", singular nominative) and *motivah* ("motive", prepositional plural). In fusional languages this isn't ideal, but since an individual root in a fusional language has a limited number of morphological variants (for example, Russian nouns have twelve), learning them all individually is still tractable. In agglutinative languages like Japanese and Turkish, however, the number of possible morphological variants

(a)	世界上有一半人口正住在城市里。 shìjiè shàng yǒu yī bàn rènkǒu zhèngzhù zài chéngshì lǐ. world on exist one half population live at city place Half of the global population now lives in cities.
(b)	手を出したくなっても自分を抑えることができます。 te-wo dasi-ta-ku-na-tte-mo jibun-wo osaeru-koto-ga deki-ma-su. hand-ACC put:out-want-ADV-become-CONT-even self-ACC supress-NMZ-NOM can-POL-PRES Even when you want to lash out, you can stop yourself.
(c)	У следствия нет версий о мотивах злоумышленника. u cledstvi-ya net versi-y o motiv-ah zloumyshlennik-a. at investigation-SG.GEN NEG theory-PL.GEN about motive-PL.PREP attacker-SG.GEN The investigation does not theorize about the attacker's motives.

Table 1: Example sentences in (a) Chinese, an isolating language, (b) Japanese, an agglutinative language, and (c) Russian, a fusional language.

is nearly limitless, certainly beyond what is tractable to learn using a naïve approach. Some NLP models attempt to handle these phenomena using greedy chunking of common character n-grams (Sennrich et al., 2015), but even these models lack a full open vocabulary and often mis-segment rarely used words and endings. In Chapter 3 we propose a language model capable of handling morphological variations in a linguistically principled way and apply it to fusional and agglutinative languages.

SYNTAX

Syntax is the linguistic study of how sentences are structurally formed from words. At a surface level, syntax describes which order(s) the words in a sentence are considered grammatical or ungrammatical. Nearly all theories of syntax, however, describe sentences *hierarchically*, with linear word order arising as a product of the underlying tree structure.

There are numerous syntactic formalisms, many of which have been used in computational linguistics to great effect. In this thesis we will employ two of the most popular formalisms: dependency grammars and phrase structure grammars. We focus on these two formalisms due to their popularity and the availability of parsed data under these schemes. While this thesis derives models based on these two formalisms, similar models could be derived for other formalisms such as combinatorial categorical grammars (Steedman, 1987, CCGs), lexical functional grammars (Kaplan et al., 1982, LFGs), head-driven phrase structure grammars (Pollard and Sag, 1994, HPSGs) and more.

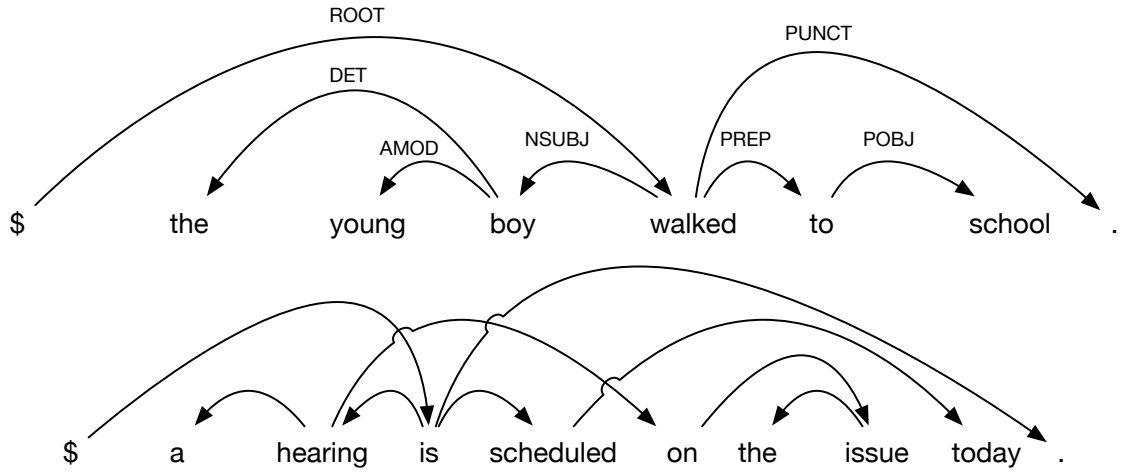


Figure 2: An example of a projective labeled dependency parse (top) and a non-projective unlabelled dependency parse (bottom)[1].

Dependency Grammars

Dependency grammars (Tesnière, 1959) are a syntactic formalism based around the notion of dependency relations between words. A dependency relation is a link between a pair of words: a head word being described and a dependent word doing the describing. A relation may optionally be annotated with the type of relationship between the head word and its dependent. For example, a dependency arc may be notated as "direct object", "subject", or "object of preposition", etc.

The dependency parse of a sentence is a collection of dependency relations such that each word in the sentence has exactly one head word that it describes. The one exception is the main finite verb, whose head is typically a special "root" symbol, frequently notated as $.

In English, most dependency trees are *projective*, i.e. drawable in two dimensions without crossing edges, and their structure mirrors that of the phrase structure parse (see §2.2.2) of the same sentence. Some rare sentences, are *non-projective*, and are not drawable without crossing edges. Non-projectivity occurs in about 0.5% of English sentences (Straka et al., 2015), but is much more common in some other languages (Buchholz and Marsi, 2006). In this thesis we follow most previous dependency parsing approaches, particularly those focusing on English, in ignoring non-projectivities due to their uncommon nature in the languages we focus on.

Figure 2 shows some examples of dependency trees. The top tree is a simple sentence and has its dependency arcs annotated to show relations. The bottom tree is unlabelled and

exhibits non-projectivity, which manifests itself as crossing lines. Each relation is depicted as a line with its tail at the headword pointing towards the dependent word. Note that each word has exactly one arrow head pointing to it, though a word may have multiple arrows emenating from it. Also note that dependency parses do not use part-of-speech information, either for terminals or non-terminal interior nodes.

In Chapter 4 we will look at generative language models using dependency grammars.

Phrase Structure Grammars

Phrase-structure trees are another syntactic representation, this time highlighting the tree-structured hierarchy underpinning sentences. Unlike dependency trees, phrase-structure trees typically use part-of-speech labels for terminals and also label internal nodes with similar labels. Some theories of phrase structure grammar (for example the popular $\bar{X}$ (pronounced "X bar") theory (Jackendoff, 1977)) require trees to be strictly binary branching, but in general phrase structure trees can contain nodes of any arity.

A phrase structure parse of a sentence is a tree structure with a singular root, typically labeled S (for "sentence") or IP (for "inflectional phrase"), and whose terminals are the words in the sentence annotated with their part-of-speech labels. Internal nodes are labelled with phrase labels (e.g. VP for "verb phrase"). See Figure 3 for an example. Well-formed subtrees, such as the noun phrase spanning the phrase "the young boy", or the prepositional phrase spanning the phrase "to school", are called constituents. Note that not all spans correspond to constituents. For example, the substring "young boy walked" is not dominatated by any tree node that does not contain any other terminals.

In Chapter 5 we will look at the problem of machine translation, using phrase structure trees in one or both languages.

LANGUAGE MODELLING

The goal of (unconditional) language modelling is to describe, usually probabilistically, which strings of words belong to a language. Most language models map strings of words to a probability $p(S)$. Since the probabilities of all strings in a language must sum to one, and there are infinitely many sentences in all natural languages, it follows that these probabilities are tiny for all but the most common sentences. (For example one could imagine

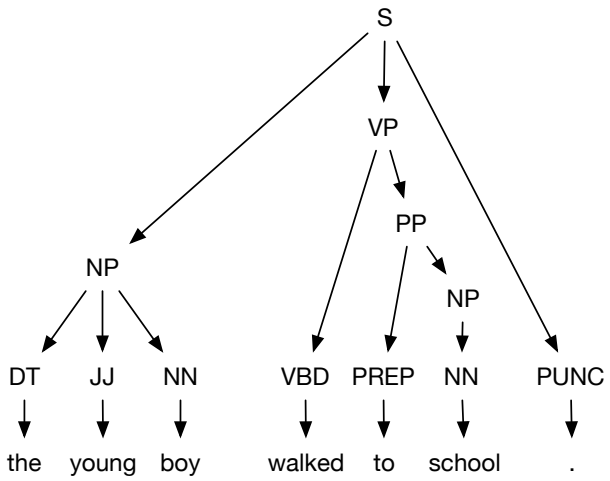


Figure 3: An example of a constituency tree.

that "How are you?" constitutes more than 0.01% of all English utterances, quite a large probability compared to that of e.g. "How might I get to Schenley Park?")

Traditionally language models have decomposed the probability of a sentence S using the chain rule, examining one word at a time in a left-to-right fashion: $p(S) = \prod_{i=1}^{|S|} p(S_i \mid S_{1:i-1})$. Nevertheless, other decompositions are possible, such as generating words hierarchically.

There are many ways of modelling $p(S_i \mid S_{1:i-1})$. Perhaps the simplest way is to make a Markov assumption that the ith word depends only on the preceeding n words, and is independent of the history before that: $p(S_i \mid S_{1:i-1}) \doteq p(S_i \mid S_{i-n:i-1})$. If $n = 0$ this is a unigram model; $p(S_i)$ is independent of all history. If $n = 1$, this corresponds to a bigram model: $p(S_i \mid S_{1:i-1}) \doteq p(S_i \mid S_{i-1})$. If $n = 2$ we would have a trigram model, and so on. Traditionally these types of n-gram based models are estimated by simply normalizing n-gram counts from a large corpus, possibly with some amount of smoothing (e.g. Ney et al. (1994)). Later n-gram models use neural networks to estimate the distribution $p(S_i \mid S_{i-n:i-1})$ (Bengio et al., 2003).

More recent models have forgone Markov assumptions entirely. Such models rely either on recurrent neural networks (Elman, 1990; Mikolov et al., 2010, RNNs) or self-attention (Vaswani et al., 2017) to model $p(S_i \mid S_{1:i-1})$ without independence assumptions.

In this thesis we primarily use RNN-based language models so we will examine them in more detail here. Like most LMs, RNNLMs decompose the probability of a sentence using the chain rule from left to right: $p(S) = \prod_{i=1}^{|S|} p(S_i \mid S_{1:i-1})$. Unlike other models, however, RNNLMs encode the sequence of previously seen words $S_{1:i-1}$ as a vector h_i. The initial state h_0, from which all sentences begin, represents the state of having seen no

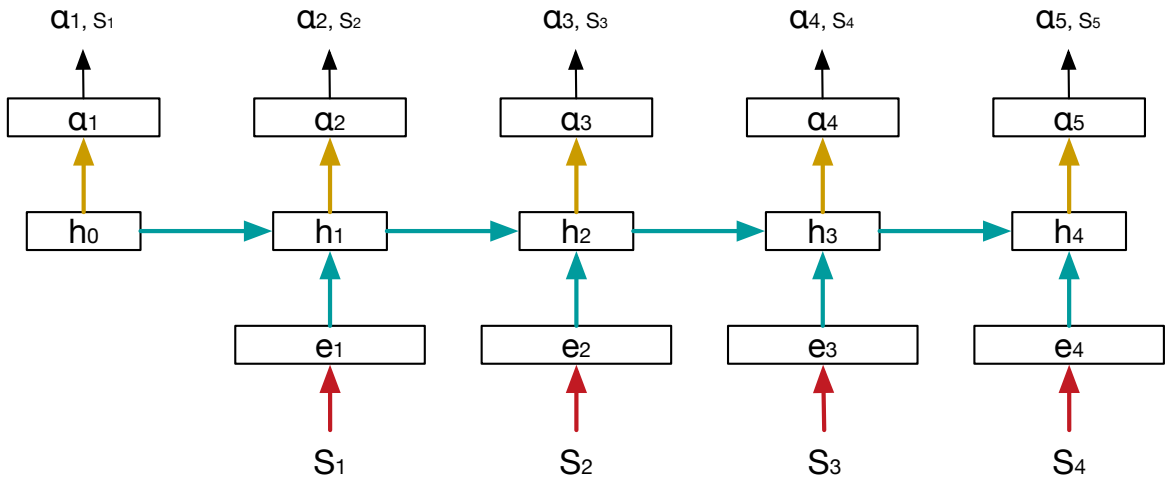


Figure 4: The architecture of an RNN language model. The red arrows represent the $\mathbb{E}$ function, teal arrows represent the f function, and gold arrows represent the p function.

words yet. It is treated as a learnable parameter of the model. Each individual word S_i in the sentence is passed through an embedding function $\mathbb{E}$, which turns a word into a dense vector $e_i = \mathbb{E}(S_i)$. The other h_is are then computed in a recurrent fashion from the previous state h_{i-1} and the embedding ith word of the input sentence e_i: $h_i = f(h_{i-1}, e_i)$. This allows us to rewrite $p(S)$ as $\prod_{i=1}^{|S|} p(S_i \mid h_{t-1})$.

While this general formulation underlies all RNNLMs, the particular choice of implementation of the functions $\mathbb{E}$, f, $p(S_i \mid h_{t-1})$ leads to many different variants. Given a finite vocabulary V, the simplest and most common implementation of $\mathbb{E}$ is to simply use an embedding matrix with $|V|$ rows and d columns. The embedding of the jth word in the vocabulary is then simply the jth row of this matrix, which yields a d-dimensional vector.

The probability of a word given a hidden state is typically modelled with a simple affine transform followed by a softmax. For example, one could define $a = Wh_i + b$, where W and b are learned parameters, and then write $p(w \mid h_i) = \alpha_{i,w} = \frac{e^{a_w}}{\sum_{j=1}^{|V|} e^{a_j}}$. Here a is a $|V|$-dimensional vector that gives a raw score a_j to the jth word in the vocabulary. To ensure that the probability distribution is well-formed (i.e. that it's positive and sums to one), we exponentiate (to ensure positivity) and normalize (to ensure the distribution sums to one). This produces a probability distribution α_i over the $|V|$ vocabulary words, and we can write the probability $p(S_i \mid h_i) = \alpha_{i,S_i}$.

The function $f(h_{i-1}, e_i)$ is more intricate. Early work used $f(h_{i-1}, e_i) = \sigma(Wh_{i-1} + Ve_i + b)$ where W, V, and b are learned parameters and σ is the sigmoid function (Mikolov et al., 2010). Subsequently more complex models such as LSTMs (Hochreiter and Schmid-

huber, 1997) and GRUs (Cho et al., 2014) have been developed, mostly focusing on ensuring proper flow of gradient information over long time scales.

In Chapter 3 we explore how to do language modelling for morphologically complex languages by introducing a novel $\mathbb{E}$ function. In Chapter 4 we investigate whether dependency trees can help us model language by changing the chain rule decomposition to be not strictly left-to-right and by introducing a new f function that captures syntactic information.

MACHINE TRANSLATION

Machine translation is the process of automatically translating text (or perhaps speech) from one language into another. The study of machine translation goes back to the 1950s, when Warren Weaver famously wrote "One naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.' " (Weaver, 1955). For the next sixty years machine virtually all translation approaches would follow this cryptography-inspired paradigm.

Formally machine translation systems define a model of $p(T \mid S)$, where S is a sentence in the source langauge and T is a sentence in the target language. Given an input sentence S, the system then seeks to find $T^* = \arg\max_T p(T \mid S)$. The cryptography-inspired systems then rewrite $p(T \mid S)$ using Bayes' Rule: $p(T \mid S) = \frac{p(S|T) \cdot p(T)}{p(S)}$ Since the denominator does not depend on T one may note that $T^* = \arg\max_T p(T \mid S) = \arg\max_T \frac{p(S|T) \cdot p(T)}{p(S)} = \arg\max_T p(S \mid T) \cdot p(T)$. This formulation allows us to split the modelling of $p(T \mid S)$ into two pieces: a reverse translation model $p(S \mid T)$ and a language model $p(T)$. But does this reformulation really buy us anything? Isn't modelling $p(S \mid T)$ just as hard as the original problem of modelling $p(T \mid S)$? In addition we have added a language model as an additional component! In fact the addition of a language model is the big advantage of this reformulation. By introducing a language model we take a large amount of the modelling load off of the translation model, allowing us to use simpler techniques with better effect. Furthermore, the language model allows us to use monolingual training data, which is far more plentiful than the sentence-aligned bitexts needed to train the translation model.

When looking at a source sentence S (say French) that we would like to translate into a target language T (say English), this model captures a generative story that goes as follow. Following the above quote by Weaver, we think this sentence was originally in English! We

have a prior $p(T)$ on what types of sentences English speakers are likely to produce. After selecting an English sentence to utter, however, the sentence goes through a corruption process, becoming a French sentence along the way. The probability of a French sentence S arising from an English sentence T is the translation model probability $p(S \mid T)$. We then multiply these two probabilities together to get the overall probability of a French sentence and its "underlying" English: $p(S, T) = p(S \mid T) \cdot p(T)$.

The language model assigns a probability $p(T)$ to each possible sentence T in the target language. Traditionally LMs in statistical MT systems have been implemented using n-gram language models (with n between 3 and 5) smoothed with improved Kneser-Ney smoothing (Ney et al., 1994) due to their simplicity, and speed. While their fixed-length memories make n-gram language models generally inferior to recurrent ones, they allow hypothesis recombination, allowing for faster decoding. The LM could, of course also use any of the other techniques descrbed in §2.3. Such a change would likely produce gains in translation quality while making inference take substantially longer. Hybrid systems with symbolic translation models and neural language models did see some popularity during the transition from symbolic to fully neural systems (Baltescu et al., 2014).

The translation model assigns a conditional probability $p(S \mid T)$ to a source sentence S conditioned on a target sentence T. How can we estimate such a complicated distribution? We might naïvely think we can just gather a big corpus and count and normalize to get the required probabilities. With such a simple model, however, what do we do when presented a sentence not in our corpus? No matter how large our corpus may be, given the infinite complexity of human language we're certain to encounter many new sentences.

Since we can't directly model translation using entire sentences we might next think to model it using words. In this case we can write $p(T \mid S)$ as the product of word-level translation probabilities: $\prod_{i=0}^{|T|} p(T_i \mid S_{a_i})$. Here a is a latent variable indicating the alignment between the source and target sentences. The alignment assigns exactly one source index a_i to each target word T_i indicating that T_i is a translation of S_{a_i}. Note that this scheme allows one source word to translate zero times, one time, or indeed many times. In order to allow for target words with no equivalent in the source sentence to appear it is common to add a special NULL token to the source side. In this way we maintain the property that each target word aligns to exactly one source word. Figure 5 shows an example alignment between a Japanese source sentence and an English target sentence. Naturally we can simply reverse this model to compute $p(S \mid T)$ rather than $p(T \; mid S)$.

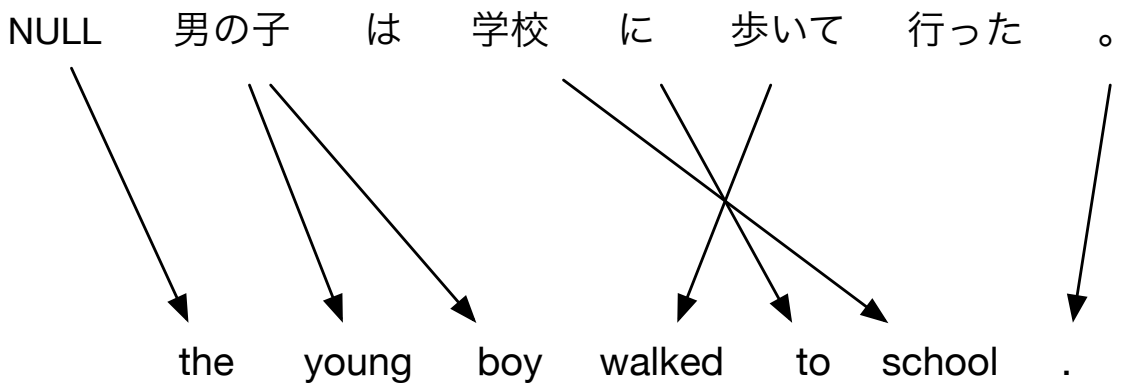


Figure 5: An example alignment between a Japanese source sentence (with a special NULL token) and an English target sentence.

Given an alignment for each sentence in a corpus, it becomes fairly straight forward to estimate a probabilistic translation dictionary from the corpus via simple counting and normalizing. Given such a dictionary we can produce a maximum likelihood estimate of the alignment for each sentence. This chicken and egg style problem lends itself nicely to the EM algorithm. We initialize by assuming a uniform translation dictionary, use it to construct our best estimate of the alignments, then use those alignments to re-estimate the probabilities in the translation dictionary. Then we repeat these two steps until convergence.

At inference time we can use this translation model and a language model to decode. For each word in a new source sentence we simply look up their top few target side translations. We then look through all combinations of these translations and all possible reorderings of the target words and score each with our language model. The overall translation that maximizes $p(S \mid T) \cdot p(T)$ is the winner. Exhaustively listing out all the possible combinations and reorderings would, of course, be infeasible, so decoders use a combination of clever heuristics (e.g. hypothesis recombination) and pruning schemes (e.g. threshold pruning) to make this inference tractable.

What we have described here is an overview of how a statistical machine translation system works. Naturally there have been many extensions over the years that have increased performance at the cost of complexity. For example, more complex word alignment models have been used (Brown et al., 1993; Dyer et al., 2013), and phrase-level dictionaries can be used in place of word-level ones (Koehn et al., 2003).

NEURAL MACHINE TRANSLATION

Naturally one may also use neural networks to model the complex probability distributions involved in machine translation. While one could continue to use the noisy channel model and model $p(S \mid T)$ and $p(T)$ with neural nets (Yu et al., 2016), the decoding problem of finding $T^* = \arg\max_T p(S \mid T) \cdot p(T)$ becomes intractable. As such, neural machine translation (NMT) models focus on modelling $p(T \mid S)$ directly, relying on the high capacity of neural models instead of offloading the burden to a language model.

One intuitive approach to NMT is to use an RNN or similar to encode a source sentence a single vector h, and then use a separate RNN-based decoder to expand h into a target sentence (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014).

While this encoder-decoder framework underpins all popular NMT models, the addition of attention (Bahdanau et al., 2015) led to substantially improved performance. The idea is that instead of forcing the model to encode the entire source sentence, which could be arbitrarily long and complicated, into just one vector, we instead allow the decoder to choose a single source word (or perhaps a few source words) to look at before emitting each target word. Formally, we write that the encoder takes a sentence $S = s_1, s_2, \ldots, s_N$ and produces a sequence of context-aware vectors, $H = h_1, h_2, \ldots, h_N$, one for each source word. The decoder is an RNN that begins in some initial state d_0. At the ith time step, the model uses the source word vectors and the decoder's state to compute a score for each source word representing how much it wants to look at that word next. Those scores are then passed through a softmax, yielding a distribution over source word positions. The in-context source vectors are combined using a weighted sum according to this attention distribution, creating a single context vector c_t. The decoder then receives its own previous state d_t and the context vector c_t and chooses one target word $\hat{t}_t$ to emit. Finally d_t and $\hat{t}_t$ are used to update the decoder's internal state, yielding a new decoder state vector d_{t+1}, and the process repeats.

3

NEURAL MORPHOLOGY FOR OPEN-VOCABULARY LANGUAGE MODELS

Language modelling of morphologically rich languages is particularly challenging due to the vast set of potential word forms and the sparsity with which they appear in corpora. Traditional *closed vocabulary* models treat words as atomic units and as such they are unable to produce word forms unseen in training data or to generalize from sub-word patterns found in data.

The most straightforward solution is to treat language as a sequence of characters (Sutskever et al., 2011). However, models that operate at two levels—a character level and a word level—have better performance (Chung et al., 2017; Kawakami et al., 2017). Byte-pair Encoding (Sennrich et al., 2015, BPE) offers a technique to break words into frequent character n-grams and translate those instead of full words. While this approach does allow for pieces to recombine, yielding an infinite vocabulary size, there are still words it cannot emit so it is not truly open vocabulary. Another solution is to use morphological information from hand-crafted analyzers, which has shown benefits in non-neural models (Chahuneau et al., 2013b). In this chapter, we present a model that combines character-, morpheme-, and word-level information in a fully neural framework.

Our model incorporates explicit morphological knowledge (e.g. from a finite-state morphological analyzer/generator) into a neural language model, combining it with existing word- and character-level modelling techniques, in order to create a model capable of successfully modelling morphologically complex languages. In particular, our model achieves three desirable properties.

First, it conditions on all available (intra-sentential) context, allowing it, in principle, to capture long-range dependencies, such as the verb agreement between "students" and "are" in the sentence "The students who studied the hardest are getting the highest grades". While traditional n-gram based language models lack this property, RNN-based language models fulfill it.

This chapter was previously published at NAACL 2018.

Second, it explicitly captures morphological variation, allowing sharing of information between variants of the same word. This allows faster, smoother training as well as improved predictive generalization. For example, if the model sees the phrase "gorped the ball" in data, it is able to infer that "gorping the ball" is also likely to be valid. Similarly, the model is capable of understanding that morphological consistency within noun phrases is important. For example in Russian, one might say *malen'kaya chërniya koshka* ("small black cat", nominative), or *malen'kuyu chërniyu koshku* (accusative), but *malen'kiy chërnuyu koshke* (mixing nominative, accusative and dative) would have much lower probability.

Third, the language model seamlessly handles out of vocabulary items and their morphological variants. For example, even if the word *Obama* was never seen in a Russian corpus, we expect *Ya dal eto prezidentu Obame* ("I gave it to president Obama") to have higher probability using the dative *Obame* than *Ya dal eto prezidentu Obama*, which uses the nominative *Obama*. The model can also learn to decline proper nouns, including OOVs. Here it can recognize that *dal* ("gave") requires a dative, and that nouns ending with *-a* generally do not meet that requirement while nouns ending with *-e* do.

In order to capture these properties, our model combines two pieces: an alternative embedding module that uses sub-word information such as character and morpheme-level information, and a generation module that allows us to output words at the word, morpheme, or character-level. The embedding module allows for the model to share information between morphological variants of surface forms and produce sensible word embeddings for tokens never seen during training. The generation model allows us to emit tokens never seen during training, either by combining a lemma and a sequence of affixes to create a novel surface form, or by directly spelling out the desired word character by character. We then demonstrate the effectiveness both intrinsically, showing reduced perplexity on several morphologically rich languages, and extrinsically on machine translation and morphological disambiguation tasks.

MULTI-LEVEL RNNLMS

Recurrent neural network language models are composed of three parts: (a) an encoder, which turns a context word into a vector, (b) a recurrent backbone that turns a sequence of word vectors that represent the ordered sequence of context vectors into a single vector, and (c) a generator, which assigns a probability to each word that could follow the given context.

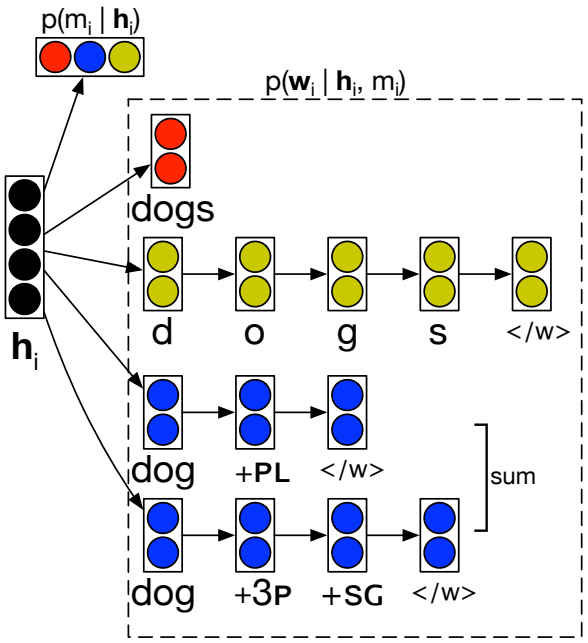


Figure 6: We allow the model to generate an output word at the word, morpheme, or character level, and marginalize over these three options to find the total probability of a word.

RNNLMs often use the same process for (a) and (c), but there is no reason why these processes cannot be decoupled. For example, Kim et al. (2016) and Ling et al. (2015) compose character-level representations for their word encoder, but generate words using a softmax whose probabilities rely on inner products between the current context vector and type-specific word embeddings.

In our model both the word generator (§3.1.1) and the word encoder (§3.1.2) compute representations that leverage three different views of words: frequent words have their own parameters, words that can be analyzed/generated by an analyzer are represented in terms of sequences of abstract morphemes, and all words are represented as a sequence of characters.

Word generation mixture model

In typical RNNLMs the probability of the ith word in a sentence, w_i given the preceding words is computed by using an RNN to encode the context followed by a softmax:

$$\begin{aligned} p(w_i \mid \boldsymbol{w}_{<i}) &= p(w_i \mid \mathbf{h}_i = \varphi_{\text{RNN}}(w_1, \ldots, w_{i-1})) \\ &= \text{softmax}(\mathbf{W}\mathbf{h}_i + \mathbf{b}) \end{aligned}$$

where φ_{RNN} is an RNN that reads a sequence of words and returns a fixed sized vector encoding, $\mathbf{W}$ is a weight matrix, and $\mathbf{b}$ is a bias.

In this work, we will use a mixture model over M different models for generating words in place of the single softmax over words (Miyamoto and Cho, 2016; Neubig and Dyer, 2016):

$$\begin{aligned} p(w_i \mid \mathbf{h}_i) &= \sum_{m_i=1}^{M} p(w_i, m_i \mid \mathbf{h}_i) \\ &= \sum_{m_i=1}^{M} p(m_i \mid \mathbf{h}_i) p(w_i \mid \mathbf{h}_i, m_i), \end{aligned}$$

where $m_i \in [1, M]$ indicates the model used to generate word w_i. To ensure tractability for training and inference, we assume that m_i is conditionally independent of all $\boldsymbol{m}_{<i}$, given the sequence of word forms $\boldsymbol{w}_{<i}$.

We use three ($M = 3$) component models: (1) directly sampling a word from a finite vocabulary ($m_i = \textsc{word}$), (2) generating a word as a sequence of characters ($m_i = \textsc{chars}$), and (3) generating as a sequence of (abstract) morphemes which are then stitched together using a hand-written morphological transducer that maps from abstract morpheme sequences to surface forms ($m_i = \textsc{morphs}$). Figure 6 illustrates the model components, and we describe in more detail here:

WORD GENERATOR. Select a word by directly sampling from a multinomial distribution over surface form words. Here the vocabulary is the $|V_w|$ most common full-form words seen during training. All less frequent words are assigned zero probability by this model, and must be generated by one of the remaining models.

CHARACTER SEQUENCE GENERATOR. Generate a word as a sequence of characters. Each character is predicted conditioned on the LM hidden state $\mathbf{h}_i$ and the partial word generated so far, encoded with an RNN. The product of these per-character probabilities is the total probability assigned to a full word form.

MORPHEME SEQUENCE GENERATOR. Similarly to the character sequence generator, we can generate a word as a sequence of morphemes. We first generate a root r, followed by a sequence of affixes $a_1, a_2, \ldots$. For example the word "devours" might be generated as devour+3P+SG+EOW.

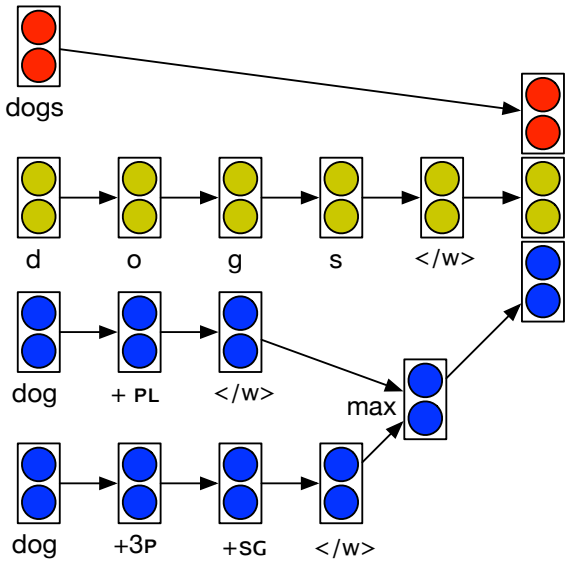


Figure 7: We concatenate word- morpheme- and character-level vectors to build a better input vector for our RNNLM.

Since multiple sequences of abstract morphemes may in general give rise to a single output form,[1] we marginalize these, i.e.,

$$p(w_i \mid \mathbf{h}_i, m_i = \textsc{morphs}) = \sum_{\boldsymbol{a}_i \in \{\boldsymbol{a} \mid \textsc{gen}(\boldsymbol{a}) = w_i\}} p_{morphs}(\boldsymbol{a}_i \mid \mathbf{h}_i).$$

where $\textsc{gen}(\boldsymbol{a})$ gives the surface word form produced from the morpheme sequence $\boldsymbol{a}$.

Due to the model's ability to produce output at the character level, it is able to produce any output sequence at all within the language's alphabet. This is critical as it allows the model to generate unknown words, such as novel names or declensions thereof. Furthermore, the morphological level facilitates the model's generation of word forms whose lemmas may be known, but whose surface form was nevertheless unattested in the training data. Finally the word-level generation model handles generating words that the model has seen many times during training.

Morphologically aware context vectors

Word vectors are typically learned with a single, independent vector for each word type. This independence means, for example, that the vectors for the word "apple" and the word "apples" are completely unrelated. Seeing the word "apple" gives no information at all about the word "apples".

Ideally we would like to share information between such related words. Nevertheless, sometimes words have idiomatic usage, so we'd like not to tie them together too tightly.

We accomplish this by again using three different types of word vectors for each word in the vocabulary. The first is a standard per-type word vector. The second is the output of a character-level RNN using Long Short-Term Memory (LSTM) units (Hochreiter and Schmidhuber, 1997). The third is the output of a morphology-level LSTM over a lemma and a sequence of affixes, as output by a morphological analyzer.

Typically language models first generate a word w_i given some (initially empty) prior context c_{i-1}, and then that word is combined with the context to generate a new context c_i that includes the new word. Since we have just used one or more of our three modes to generate each word, intuitively we would like to use the same mode(s) to generate the embedding used to progress the context.

Unfortunately, doing so introduces dependencies among the latent variables $p(\text{mode} \mid c_i)$ in our model, making exact inference intractable. As such, we instead drop the dependency on how a word was generated and instead represent the word at all three levels, regardless of the mode(s) actually used to generate it, and combine them by concatenating the three representations. A visual representation of the embedding process is shown in Figure 7.

Additionally, should a morphological analyzer produce more than one valid analysis for a surface form, we independently produce embeddings for each candidate analysis, and combine them using a per-dimension maximum operation. Mathematically, the ith dimension of the morphological embedding e_m is given by

$$e_{mi} = \max_j e_{a_j i}$$

1 In general analyzers encode many-to-many relations, but our model assumes that any sequence of morphs in the underlying language generates a single surface form. This is generally true, although free spelling variants of a morph (e.g., American *-ize* vs. British *-ise* as well as alternative realizations like *shined/shone* and *learned/learnt*) violate this assumption.

where e_{a_j} is the embedding of the jth possible analysis, as computed by the LSTM over the lemma and its sequence of affixes.

The intuition behind the use of all analyses plus a pooling operation can be seen by observing the case of the word "does", which could be *do+3-person+singular* or *doe+plural*. If this word appears after the word "he", what we care about more is whether "does" could feasibly be a third person singular verb, thus agreeing with the subject. The max-pooling operation captures this intuition by ensuring that if a feature is active for one of these two analyses, it will also be active in the pooled representation. This procedure affords us the capability to efficiently marginalize over all three possible values of the latent variable at each step, and compute the full marginal of the word w_i given the context c_{i-1} during generation.

This formulation allows words with the same stem to share vector information through the character or morphological embeddings, but still affords each word the ability to capture idiomatic usages of individual words though the word embeddings. Furthermore, it allows a language model to explicitly capture morphological information, for example that third person singular subjects should co-occur with third person singular verbs. Finally, the character-level segment of the embedding allows the model to at least attempt to build sensible embeddings for completely unknown words. For example in Russian where names can decline with case this formulation allows the model to know that *Obame* is probably dative, even if it's an OOV at the word level, and even if the morphological analyzer is unable to produce any valid analyses.

We combine our three-layer input vectors, our factored output model, and a standard LSTM backbone to create a morphologically-enabled RNNLM that, as we will see in the next section, performs well on morphologically complex languages.

INTRINSIC EVALUATION

We demonstrate the effectiveness of our model by experimenting on three languages: Finnish, Turkish, and Russian. We use varying amounts of News Commentary data provided by the Workshop on Machine Translation (WMT) for each of the three languages. Statistics of our experimental corpora can be found in Table 2.

Each data set was pre-processed by UNKing all but the top ≈20k words and lemmas by frequency. No characters or affixes were UNKed. This step is not strictly required—our model is, after all, capable of producing arbitrary words— but it speeds up training im-

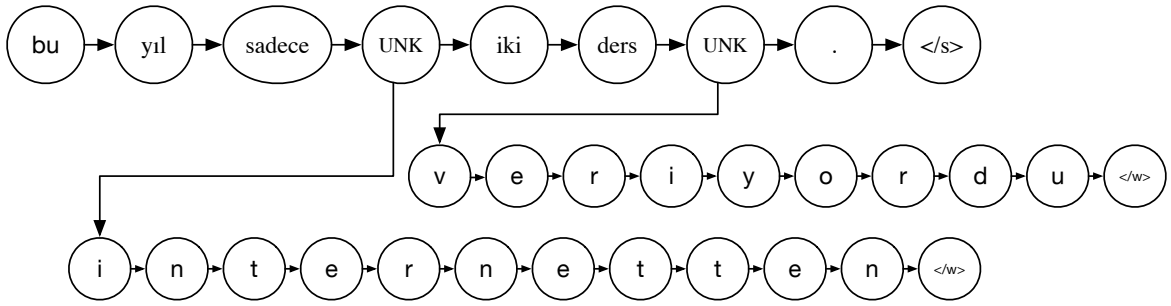


Figure 8: An example of our 4-gram Kneser-Ney baseline that backs off to an additional character-level model for out of vocabulary words generating a Turkish sentence that contains two out-of-vocabulary items.

mensely by reducing the size of the word and lemma softmaxes. Since the morphology and/or character-level embeddings can still capture information about the original forms of these words, the degradation in modelling performance is minimal.

For morphological analysis we use Omorfi[2] (Pirinen, 2015) for Finnish, the analyzer of Oflazer (1994) for Turkish, and PyMorphy[3] (Korobov, 2015) for Russian. PyMorphy and Omorfi were created by inducing DAFSA from morphological analyses crowd sourced from native speakers using a series of questions designed by linguists. The Turkish analyzer is based on a context-free grammar applied to a curated dictionary of Turkish words (ank, 1981).

Baseline Models

Since models are not accurately comparable unless they share output vocabularies, our baselines must also allow for the generation of arbitrary word forms, including out-of-vocabulary items. We compare to three such models: an improved Kneser-Ney (Kneser and Ney, 1995) 4-gram baseline, with an additional character-level backoff model for OOVs, an RNNLM with character-level backoff, and a pure character-based RNN language model (Sutskever et al., 2011).

Since Kneser-Ney language models (and other count-based models) are typically word-level and do not model out-of-vocabulary items, we employ a two-level approach with separate Kneser-Ney models at the word and character levels. We train the word-level model after UNKing low frequency words, and we train the character-level model on the same list of low frequency words. Now when we want to predict a word w_i given some

2 https://github.com/flammie/omorfi
3 https://github.com/kmike/pymorphy

context c we can use the word-level model to directly predict $p(w_i|c)$ unless w_i is an out-of-vocabulary item. In that case we model $p(w_i \mid c)$ as

$$p(w_i \mid c) = p(\text{UNK} \mid c) \cdot p(w_i \mid \text{UNK})$$

where the first factor is the probability of the word-level model emitting UNK, and the second is the probability of the actual out-of-vocabulary word form under the character-level model. See Figure 8.

Secondly we compare to a similar hybrid RNN model that first predicts the word-level probability for each word, and if it predicted UNK then also predicts a sequence of characters using a separate network. This model uses 256-dimensional character and word embeddings, and a 1024-dimensional recurrent hidden layer.

Finally we also compare to a standard RNN language model trained purely on the character level. For this baseline we also use 256-dimensional character embeddings and a 1024-dimensional recurrent hidden layer.

Multi-factored Models

For our model we use 128-dimensional word and root embeddings, 64-dimensional affix and character embeddings, 128-dimensional word-internal recurrent hidden layers for characters and morphemes, and a 256-dimensional recurrent hidden layer for the main inter-word LSTM.

The network is trained to stochastically optimize the log likelihood of the training data using Adam (Kingma and Ba, 2015). After each 10k training examples (Finnish, Turkish) or 100k training examples (Russian) we evaluate the model on a development set. (We evaluate less frequently on Russian since the dev set is much larger.) At the end of training we revert the model to the version that scored the best on the development set, thereby mitigating overfitting via early stopping. No other regularization is used.

For each language we run four variants of our model. In order to preserve the ability to model and emit any word in the modelled language, it is essential that we keep the character-level part of our model intact. The morpheme- and word-level models, however, may be removed without compromising the generality of the model. As such, we present our model using only character-level input and outputs (C), using character- and

	Finnish	Russian	Turkish
Train Sents	2.1M	1.1M	188K
Train Words	38M	26M	3.9M
Dev Sents	1K	38K	1K
Dev Words	16K	705K	16K
Test Sents	500	91K	3K
Test Words	7.6K	1.6M	51K
Word Vocab	20K	21K	42K
Lemma Vocab	20K	20K	13K
Affix Vocab	140	34	180
Char Vocab	229	150	80

Table 2: Details of our data sets. Each cell indicates the number of sentences and the number of words in each set.

(a) Finnish

Model	Dev	Test
KN4+OOV	2.04	1.94
RNN+OOV	2.03	1.92
PureC	2.69	2.63
C	2.40	2.32
CM	1.95	1.85
CW	2.03	1.94
CWM	**1.91**	**1.81**

(b) Turkish

Model	Dev	Test
KN4+OOV	2.01	2.06
RNN+OOV	1.99	2.05
PureC	2.21	2.30
C	2.05	2.16
CM	1.88	1.99
CW	1.78	1.85
CWM	**1.74**	**1.82**

(c) Russian

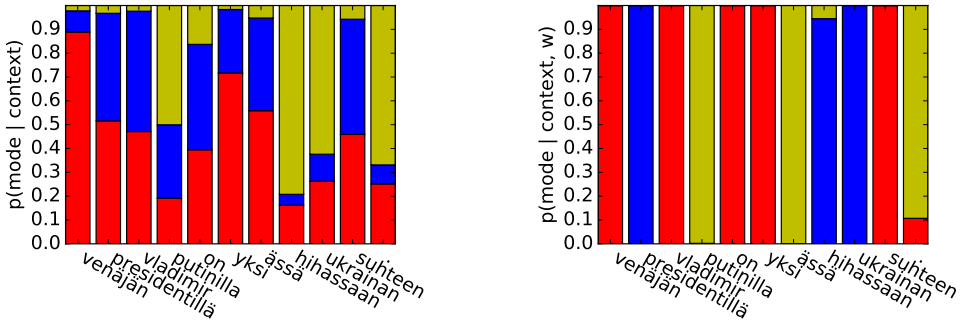
Model	Dev	Test
KN4+OOV	1.68	1.70
RNN+OOV	1.62	1.66
PureC	1.91	2.05
C	1.85	1.87
CM	1.47	1.50
CW	**1.44**	**1.47**
CWM	1.49	1.52

Table 3: Intrinsic evaluation of language models for three morphologically rich languages. Entropy for each test set is given in bits per character. Lower is better, with 1.0 being perfect.

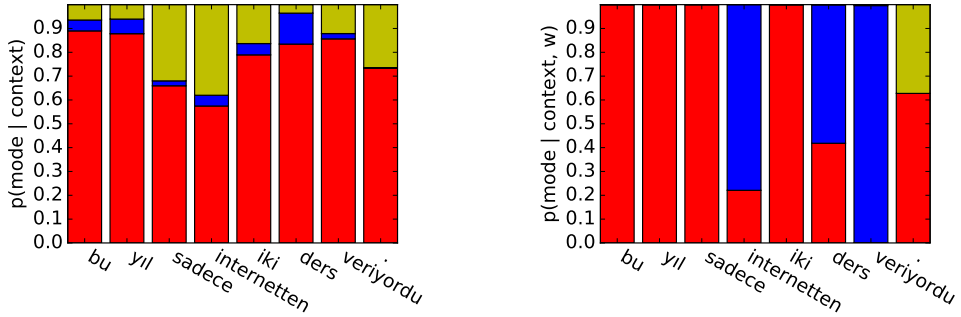
morpheme-level inputs and outputs (CM), using character- and word-level inputs, but no morphology (CW), and using all three levels as per the full model (CMW).

Results and Analysis

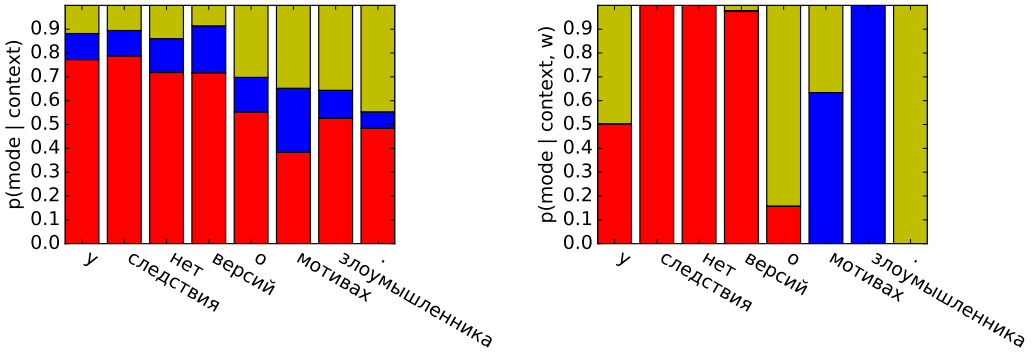
Our experimental results (Table 3) show that our multi-modal model significantly outperforms all three baselines: a naïve n-gram model, a purely character-level RNNLM, and a hybrid RNNLM for open-vocabulary language models. Furthermore, they confirm that morphological analyzers can improve performance of such language models on particularly morphologically rich languages. We observe that across all three languages the space-aware character-level model outperforms the purely character-based model that treats spaces just as any other character. We additionally find that the relative success of the n-gram model



Finnish: "Russian President Vladimir Putin has an ace up his sleeve in the Ukrainian relationship."



Turkish: "He gave only two lectures on the internet this year."



Russian: "The investigation does not theorize about the attacker's motives."

Figure 9: Some examples of the priors (left) and posteriors (right) over modes used to generate each word in some sample sentences. Probability given to the word-, morpheme-, and character-level models are shown in red, blue, and gold respectively.

and the hybrid model over the character only models underscores the importance of access to word-level information, even when using a less sophisticated model.

Our methods outperform the n-gram model in all languages with either set of just two models, CM or CW. The same models outperform the hybrid baseline in Turkish and Russian, and achieves comparable results in Finnish. Finally, in the agglutinative languages, using all three modes performs best, while in Russian, a fusional language, characters and words alone edge out the model with morphology. We hypothesize that our morphology model is better able to model the long strings of morphemes found in Turkish and Finnish, but gains little from the more idiosyncratic fusional morphemes of Russians.

Some examples of the priors and posteriors of the modes used to generate some randomly selected sentences from the held out test set can be seen in Figure 9. The figures show that all of the models tend a priori to prefer to generate words directly when possible, but that context can certainly influence its priors. In Finnish, after seeing the word *Vladmir*, the model suddenly assigns significantly more weight to the character-level model to generate the following word, which is likely to be a surname. In Russian, after the preposition *o*, the following noun is required to be in a rare case. As such, the model suddenly assigns more probability mass to the following word being generated using the morpheme-level model.

The posteriors tell a similarly encouraging story. In Finnish we see that the word *presidentillä* is overwhelmingly likely to be produced by the morphology model due to its peculiar adessive ("at") case marking. *Vladmir* is common enough in the data that it can be generated wholly, but the last name *Putin* is again inflected into the adessive case, forming *Putinilla*. Unfortunately the morphological analyzer is unfamiliar with the stem *Putin*, forcing the word to be generated by the character-level model. In our Turkish example, all of the short words are generated at the word level, while the primary nouns *internetten* ("internet") and *ders* ("lecture") are possible to generate either as words or as a sequence of morphemes. The verb, which has much more complex morphology (progressive past tense with a third person singular agent), is generated via the morphological model.

EXTRINSIC EVALUATION

In addition to evaluating our model intrinsically using perplexity, we evaluate it on two downstream tasks. The first is machine translation between English and Turkish. The second is Turkish morphological analysis disambiguation.

Lang. Pair	System	BLEU
TR-EN	Baseline	15.0
	Morph. Input	**15.2**
EN-TR	Baseline	10.1
	Morph. Output	**10.5**

Table 4: Machine Translation Results

Model	Supervised?	Ambiguous Words	All words
Random Chance	no	34.08%	52.66%
Unidirectional	no	55.15%	80.28%
Bidirectional	no	**63.85**%	**84.11**%
Shen et. al	yes	91.03%	96.43%

Table 5: Morphological disambiguation accuracy results for Turkish.

Machine Translation

We introduce the score of our LM as an additional feature to a cdec (Dyer et al., 2010) hierarchical MT system trained on the WMT 2016 Turkish–English data set, and perform n-best reranking after retuning weights with the new feature. In these experiments we focus our efforts on modelling the Turkish morphology, and do not assume access to an English analyzer.

The results, shown in Table 4 demonstrate small but significant gains in both directions, particularly into Turkish, where modelling productive morphologically should be more important.

Morphological Disambiguation

Our model is a joint model over words and the latent processes giving rise to those words (i.e., which generation process was selected and, for the morpheme process, which morpheme sequence was generated). While our model is not directly trained to perform morphological disambiguation, it still performs this task quite admirably. Given a trained morphological language model, a sentence $\mathbf{s}$, and a set of morphological analyses $\mathbf{z}$, we can query the model to find $p(\mathbf{s}, \mathbf{z}) = p(w_1, w_2, \ldots, w_N)$ for a given sentence. Most notably, each w_i may have a set of possible morphological analyses $\{a_1, a_2, \ldots a_{M_i}\}$ from which we would like to choose the most likely *a posteriori*. To perform this task, we simply query the

model M_i times, each time hiding all but the jth possible analysis from the model. We can then re-normalize the resulting probabilities to find $p(a_j|\mathbf{s})$ for each $j \in 1 \dots M_i$.

To make training and inference with our model tractable, we have assumed independence between previous adjacent events and the next word generation given the previous surface word forms (§3.1.1). Thus, the posterior probability over the analysis is only determined by the left context—subsequent decisions are independent of the process used to generate a word at time t. However, since disambiguating information may be present in either direction, we introduce a model variant that conditions on information in both directions. Bidirectional dependencies mean that we can no longer use the chain rule to factor the probability distribution from left-to-right. Rather we have to switch to a globally normalized, undirected model (i.e., a Markov random field) to define the probabilities of selecting the mode of generation and generation probability (conditional on the mode). The factors used to parameterize the model are defined in terms of two LSTMs, one encoding from left-to-right the prefix of the ith word ($\mathbf{h}_i$, defined exactly as above), and a second encoding from right-to-left its suffix ($\mathbf{h}'_i$). These two vector representations are used to compute a score using a locally normalized mixture model for each word. Intuitively, the morphological analysis generated at each time step should be compatible with both the preceding words and the following words.

Optimizing this model with the same MLE criterion we used in the direct model is, unfortunately, intractable since a normalizer would need to be computed. Instead, we use a pseudo-likelihood objective (Besag, 1975).

$$\begin{aligned}\mathcal{L}_{\text{PL}} &= \prod_i p(w_i \mid \boldsymbol{w}_{-i}) \\ &= \prod_i \sum_m p(m_i = m \mid \boldsymbol{w}_{-i}) p(w_i \mid m, \boldsymbol{w}_{-i})\end{aligned}$$

We note that although this model has a very different semantics from the directed one, the PL training objective is identical to the directed model's, the only difference is that features are based both on the past and future, rather than only the past.

Although evaluating sentence likelihoods in this model is intractable (a normalization factor would need to be computed), posterior inference over m_i and a_i is feasible since the normalization factors cancel and therefore do not need to be computed.

For our experiments we use the data set of Yuret and Türe (2006) who manually disambiguated from among the possible forms identified by an FST. We significantly out-perform the simple baseline of randomly guessing, and our results are competitive with Yatbaz and

Yuret (2009), although they evaluated on a different dataset so they are not directly comparable. Furthermore, we also compare to a supervised model (Shen et al., 2016). While unsupervised techniques can't hope to exceed supervised accuracies, this comparison provides insight into the difficulty of the problem.

RELATED WORK

PURELY CHARACTER-BASED OR SUBWORD-BASED LMS have a rich history going all the way back to Markov (1906)'s work modelling Russian character-by-character with his namesake models. More recently Sutskever et al. (2011) were the first to apply RNNs to character-level language modelling, leveraging their ability to handle the long-range dependencies required to model language at the character level. It is also possible to alleviate the closed vocabulary problem by training models on automatically acquired subword units (Mikolov et al., 2012; Sennrich et al., 2015). Neubig and Duh (2013) also studied character-level LMs on a variety of languages, including many morphologically rich ones, but used more traditional count-based language models. Character-level models are also used in morphological re-inflection, either with weighted FSTs (Rastogi et al., 2016) or in a fully neural sequence-to-sequence framework (Faruqui et al., 2016). While these approaches allow for an open vocabulary (or nearly open, in the case of subwords) they discard a large amount of higher-level information, inhibiting learning.

CHARACTER-AWARE LANGUAGE MODELS, which combine character- and word-level information have shown promise (Kang et al., 2011; Ling et al., 2015; Kim et al., 2016). Unsupervised morphology has also been shown to improve the representations used by a log-bilinear LM (Botha and Blunsom, 2014). Jozefowicz et al. (2016) explore many interesting such architectures, and compare with fully character-based models. While these models allow for the elegant encoding of novel word forms they lack an open vocabulary.

OPEN-VOCABULARY HYBRID MODELS alleviate this problem, extending the benefits of character-level representations to the generation. Such hybrid models with open vocabularies have been around since Brown et al. (1992). More recently, Chung et al. (2017) and Hwang and Sung (2017) describe methods of modelling sentences at both the word and character levels, using mechanisms to allow both a word-internal model that captures short-range dependencies and a word-external model to capture longer-range dependen-

cies. These models have been successfully applied to machine translation by Luong and Manning (2016), who use a character-level model to predict translations of out of vocabulary words. Our work falls in this category—we combine multiple representation levels while maintaining the ability to generate any character sequence. In contrast to these previous works, we demonstrate the utility of incorporating morphological information in these open-vocabulary models.

MIXTURE MODEL LANGUAGE GENERATION where the mixture coefficients are predicted by a neural net are becoming quite common. Neubig and Dyer (2016) use this strategy to combine a count-based model and a neural language model. Ling et al. (2016) interpolate between character- and word-based models to translate between natural language text and computer code. Merity et al. (2017) also use multiple output models, allowing a word to either be generated by a standard softmax or by copying a word from earlier in the input sentence.

Vilar et al. (2007) were among the first to explore the idea of using both word and character information for machine translation. The idea of doing machine translation at the morpheme level, using a lemma and series of affixes, was explored by Chahuneau et al. (2013a). They use a conditional random field to learn to translate source language lemmas, part of speech tags, dependency trees, and a host of other linguistically-motivated features into a target language lemma and set of morphological tags. Finally, they use a morphological analyzer to combine the lemmas and affixes into surface form words. More recently Liu et al. (2018) use character-level recurrent models to predict translations of OOV tokens in a statistical MT framework.

CONCLUSION

This chapter demonstrated that morphological information can be used to inform both the input representation and word generation processes. We introduced a technique for language modelling that works particularly well on morphologically rich languages where having an open vocabulary is desirable. We achieve this by using a multi-modal architecture that allows words to be input and output at the word, morpheme, or character levels. We show that knowledge of the existence of word boundaries is of critical importance for language modelling tasks, even when otherwise operating entirely at the character level,

resulting in a surprisingly large reduction in per-character entropy across all languages studied.

Furthermore, we demonstrate that information from hand-crafted morphological analyzers combined with neural networks are able to out-perform linguistically naïve models. This indicates that the linguistic knowledge imbued by the analyzers is able to aid the learning and generalization of neural models, providing strong evidence for the thesis statement.

4

NEURAL DEPENDENCY GENERATION

In the previous chapter we examined the formation of words via morphological processes. In this chapter we look at the formation of sentences via syntactic processes. Recurrent neural network grammars (Dyer et al., 2016, RNNGs) are syntactic language models that use predicted syntactic structures to determine the topology of the recurrent networks they use to predict subsequent words. Not only do these models learn to model language better than non-syntactic language models, but the conditional distributions over parse trees given sentences produce excellent parsers (Fried et al., 2017). The downside of these models is that they use phrase structure parses, which are expensive to annotate and only available in a small number of languages. In this chapter we investigate whether similar gains can be obtained using dependency trees, which exist in many more languages (Nivre et al., 2017).

In this chapter, we introduce and evaluate two new dependency syntax language models which are based on a recurrent neural network (RNN) backbone (§4.1).[1] Dependency syntax is particularly appealing as many more languages have dependency treebanks (e.g. the Universal Dependencies Project (Nivre et al., 2017)) than have large numbers of phrase structure annotations.

Like RNNGs, our proposed models predict structure and words jointly, and the predicted syntactic structure is used to determine the structure of the neural network that is used to represent the history of actions taken by the model and to make a better estimate of the distribution over subsequent structure-building and word-generating actions. Because we use RNNs to encode the derivation history, our models do not make any explicit independence assumptions, but instead condition on the complete history of actions. The two proposed models do, however, differ in the order that they construct the trees. The first model operates top-down (§4.1.1), starting at the root and recursively generating dependents until the last modifier has been generated. The second operates bottom-up (§4.1.2), generating words from left to right and interleaving decisions about how they fit together to form tree fragments and finally a fully formed dependency tree.[2]

This chapter is to be presented at CoNLL 2019.

1 We release code for these two models, which can be found at https://github.com/armatthews/dependency-lm.

2 In this chapter we limit ourselves to models that are capable only of generating projective dependency trees.

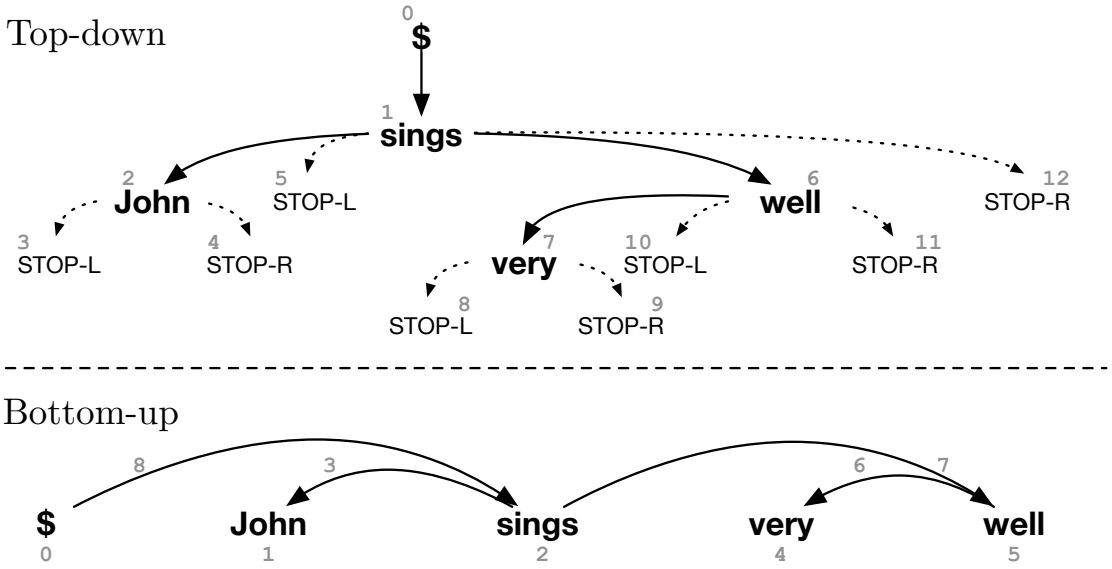


Figure 10: Generation process for the same dependency tree under the top-down and bottom-up models. As the indices of the generation events show, the top-down model generates recursively from the root, whereas the bottom-up model generates from left to right.

Because neither model makes explicit independence assumptions, given enough capacity, infinite data, and a perfect learner, both models would converge to the same estimate. However, in our limited, finite, and imperfect world, these two models will impose different biases on the learner: in one order, relevant conditioning information may be more local (which could mean the neural networks have an easier time learning to exploit the relevant information rather than becoming distracted by accidental correlations), while in the other it may be more distant. These differences thus imply that the two models will have different structural biases, but it is not at all clear whether one should outperform the other. We therefore explore to what extent this choice of construction order affects performance, and we evaluate the proposed models on language modelling and parsing tasks across three typologically different languages (§4.2).

Our findings (§4.3) show that, like RNNGs, generative dependency models make good parsers. Given the small scale of the Universal Dependency corpora, this result is also in line with previous work which shows that joint generative models offer very sample-efficient estimates of conditional distributions (Yogatama et al., 2017). Second, we find that both dependency models are *less effective* as language models than phrase structure RNNGs or than standard LSTM language models. This negative result is not entirely surprising. Although information about syntactic dependencies seems intuitively that it would be helpful for defining good conditioning contexts for language models, since its earliest days (Tesnière, 1959), work on dependency syntax has largely focused on discriminative models of *existing*

sentences. In contrast, the phrase structure annotations found in, e.g., the Penn Treebank that were used to demonstrate improved language modelling performance with RNNGs are indebted to linguistic theories (e.g. government and binding theory, X-bar theory) which are broadly concerned with determining which sentences are grammatical and which are not—a crucial aspect of language modelling (Marcus et al., 1993). Finally, we observe only minimal differences in language modelling performance for top-down and bottom-up models. This result is surprising in light of how different the estimation problems are, but it is a clear demonstration of the ability of RNNs to learn to extract relevant features from data presented in any different but consistent orders.

MODELS

We present two models for jointly generating projective dependency trees and sentences. The processes are illustrated in Fig. 10. The first is a top-down model (§4.1.1), which starts by generating the root of the sentence, and then recursively generating its left and right modifiers. The second is a bottom-up model (§4.1.2), which generates terminals in a left to right order. In both cases, there is a deterministic mapping from well-formed sequences of generation actions into dependency trees. Following convention in parsing literature, we refer to such action sequences as oracles.

Both models both are parameterized with recursively structured neural networks that have access to the complete history of generation events. Thus, the factorization of the tree probability is justified by the chain rule. However, because of the difference in build orders, the conditional probabilities being estimated are quite different, and we thus expect these models might be more or less effective at either language modelling or (when used in conjunction with Bayes' rule) parsing.

To illustrate the different estimation problems posed by the two models, consider the first generation event in both cases. In the top-down model, the root word (usually the main verb of the sentence) is generated first; whereas in the bottom-up model, the first probability modelled is the probability of the first word in the sentence. Also, in the top-down model a verb always generates its dependentents (which has implications for how agreement is modelled), whereas in the bottom-up model, it the left dependents (whatever their function) will be generated first, and then the verb generation will be conditional on them. Again, we emphasize that these differences potentially result in differences in the

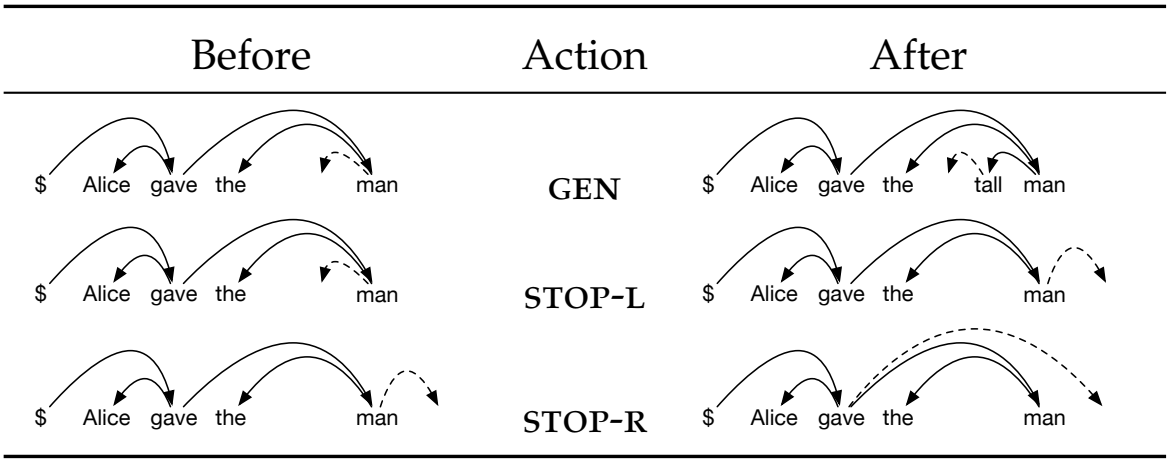


Figure 11: Examples of the three actions of our top-down model. The dotted arrow indicates where the new word will go if the GEN action is chosen next. GEN creates a new terminal node and moves the dotted arrow to point to the left of the new token. STOP-L moves the dotted arrow from the left of the current token to the right thereof. STOP-R moves the dotted arrow from the right of the current token back up to its parent node.

difficulty of the estimation problem (or how much capacity the model needs to represent accurate conditionals), but do not impact the expressivity or correctness of the models.

Top Down

Our first model is a top-down model. The model begins with an empty root node [3] Starting from the root the model recursively generates child nodes using its GEN action. When the model chooses the GEN action it then selects a new head word from its vocabulary and creates a new node descended from the most recent open constituent. Each node created in this way is pushed onto a stack of open constituents, and begins creating its left children.

To indicate that the current node (i.e. the top node in the stack) is done generating left children the model takes its STOP-L ("stop left") action, after which the current node begins generating its right children. Analogously, to indicate that the current node is done generating right children the model selects its STOP-R ("stop right") action. With this the current constituent is complete and thus popped off the stack and attached as a child of the new top-most constituent. See Figure 11 for examples of the effects of each of these three actions on a partially built tree structure and Algorithm 1 for a sketch of their implementation.

At each decision point the model conditions on the output of an LSTM over the partially completed constituents on the stack, beginning with the root and ending with the top-most constituent. The result is passed through an MLP and then a softmax that decides which

3 The root node may never have left children. In this way it is though the root node has already generated its STOP-L, though this step is not explicitly modelled

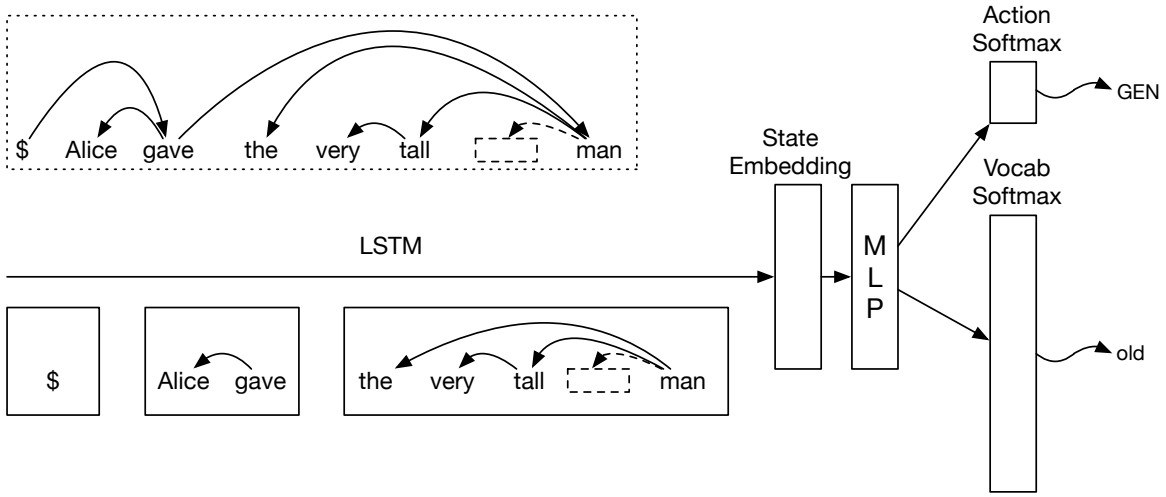


Figure 12: To encode the history of generation events in the top-down process, we use an LSTM over *subtree embeddings* (See Figure 13). The LSTM proceeds from the root of the tree down to the most recent open node. Each item in the LSTM is an embedding of a word and its already generated descendants. STOP symbols have been suppressed for clarity.

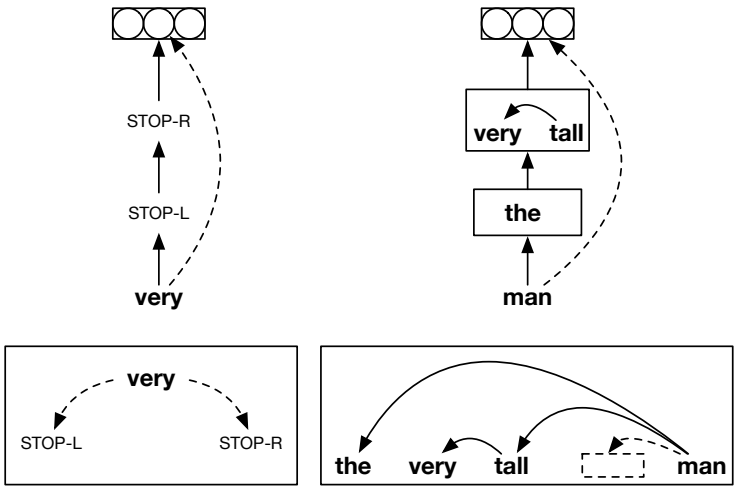


Figure 13: Examples of embedding two subtrees in the top-down model. A subtree is embedded using an LSTM over its child subtrees (solid lines) with a gated residual connection from the root word to the final embedding (dotted lines).

Algorithm 1 Top-Down Tree Generation

```
procedure EmbedTree(node)
    state = lstm_initial_state
    for child in node do
        if child is terminal then
            state.append(WordEmbs[child])
        else
            state.append(EmbedTree(child))
    return state
procedure PickNextAction(stack)
    h = MLP_action(EmbedTree(stack))
    action ~ softmax(h)
    return action
procedure PickWord(stack)
    h = MLP_word(EmbedTree(stack))
    word ~ softmax(h)
    return word
procedure GenerateNode(stack)
    action = PickNextAction(stack)
    if action == GEN then
        word = PickWord(STACK)
        stack.push(new Node(word))
    else if action == STOP-L then
        stack.back().add_child(STOP-L)
    else if action == STOP-R then
        stack.back().add_child(STOP-R)
        child_emb = stack.pop()
        stack.back().add_child(child_emb)
```

action to take next (Figure 12). If the model chooses the GEN action, the hidden vector from the MLP is used to separately choose a terminal.

To embed each subtree on the stack we use another LSTM. First we feed in the head word of the constituent, followed by the embeddings of each of the constituent's children, including the special STOP-L and STOP-R symbols. We then additionally add a gated residual connection from the head word to final subtree representation to allow salient information of the head word to be captured without needing to pass through an arbitrary number of LSTM steps (Figure 13).

Bottom-Up

Our second model generates sentences bottom-up, in the same manner as a shift-reduce parser. A sentence is modelled as a series of actions (related to the arc-standard transitions used in parsing (Nivre, 2013)) that manipulate a stack of embedded tree fragments. There

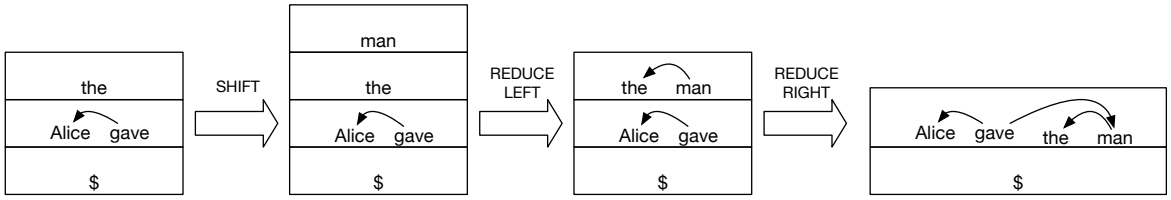


Figure 14: Examples of the three actions of our bottom-up model and their effects on the internal stack. SHIFT adds a new terminal to the top of the stack. REDUCE-L combines the top two elements of the stack with a left arc from the head of the top-most element to the head of the second element. REDUCE-R combines the top two elements with a right arc from the head of the second element to the head of the top-most element.

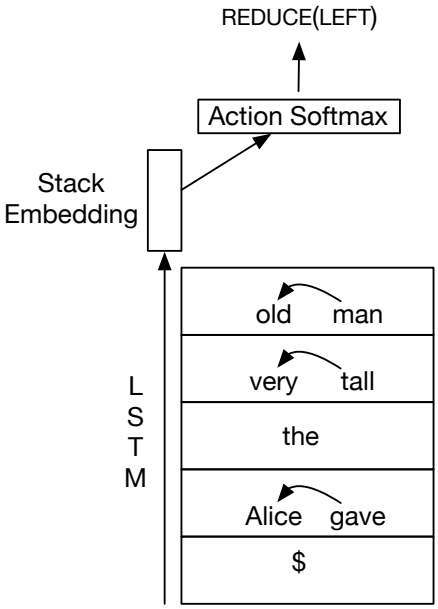


Figure 15: Our bottom-up model emulates a shift-reduce parser and maintains an explicit stack. At each timestep, we use the output of an LSTM over the stack to choose the next action, which is then executed to produce a new stack state.

are three types of actions: SHIFT(x), which pushes a new terminal x onto the stack, REDUCE-L, which combines the two top elements on the stack into one single subtree with the left of the two as the head (i.e. with a leftward arrow), and REDUCE-R which again combines the top two elements of the stack, this time making the right one the head. See Figure 14 for examples of how these three actions affect the stack of partially built tree structures during the parsing of an example sentence.

At each time step the model conditions on the state of the stack using an LSTM running over entries from oldest (bottom) to newest (top). The resulting vector $\boldsymbol{h}$ is then passed through an MLP, and then a softmax over the three possible action types. If the SHIFT action is taken, the vector $\boldsymbol{h}$ is re-used and passed through a separate MLP and softmax over the vocabulary to choose an individual word to generate. If one of the two REDUCE actions is chosen, the top two elements from the stack are popped, concatenated (with the head-to-be first, followed by the child), and passed through an MLP. The result is a vector representing a new subtree that is then pushed onto the stack. Kuncoro et al. (2016) showed that this type of stack-based representation alone is sufficient for language modelling and parsing, and indeed that more involved models actually damage model performance. See Figure 15 for an example of how this bottom-up model chooses an action.

Marginalization

Traditionally a language model takes a sentence $\boldsymbol{x}$ and assigns it a probability $p(\boldsymbol{x})$. Since our syntax-based language models jointly predicts the probability $p(\boldsymbol{x}, \boldsymbol{y})$ of a sequence of terminals $\boldsymbol{x}$ and a tree $\boldsymbol{y}$, we must marginalize over trees to get the total probability assigned to a sentence $\boldsymbol{x}$, $p(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{T}(\boldsymbol{x})} p(\boldsymbol{x}, \boldsymbol{y})$, where $\mathcal{T}(\boldsymbol{x})$ represents the set of all possible dependency trees over a sentence $\boldsymbol{x}$. Unfortunately the size of $\mathcal{T}(\boldsymbol{x})$ grows exponentially in the length of $\boldsymbol{x}$, making explicit marginalization infeasible.

Instead we use importance sampling to approximate the marginal (Dyer et al., 2016). We use the parser of Dyer et al. (2015), a discriminative neural stack-LSTM-based bottom-up parser, as our proposal distribution $q(\boldsymbol{x}, \boldsymbol{y})$ and compute the approximate marginal using $N = 1000$ samples per sentence: $p(\boldsymbol{x}) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{p(\boldsymbol{x}, \boldsymbol{y})}{q(\boldsymbol{x}, \boldsymbol{y})}$.

		Train		Dev		Test		Vocab
Language	Words	Sents	Words	Sents	Words	Sents	Singletons	Non-S'tons
English	204585	12543	25148	2002	25096	2077	9799	9873
Japanese	161900	7164	11556	511	12615	557	13091	9222
Arabic	223881	6075	30239	909	28264	680	9907	13242

Table 6: Statistics of the universal dependency data sets used in this chapter. Size of the train, dev, and test sets are given in tokens. Vocabulary information is number of types.

Parsing Evaluation through Reranking

In order to evaluate our model as a parser we would ideally like to efficiently find the MAP parse tree given an input sentence. Unfortunately, due to the unbounded dependencies across the sequences of actions used by our models this inference is infeasible. As such, we instead rerank a list of 1000 samples produced by the baseline discriminative parser, a model combination process that has been shown to improve performance by combining the different knowledge learned by the discriminative and generative models (Fried et al., 2017).

For each hypothesis parse in the sample list we query the discriminative parser, our top-down model, and our bottom-up model to obtain a score for the parse from each. We can then either simply output the best-scoring hypothesis from any one model, or we can learn a set of weights that optimizes performance on the dev set by combining the scores of two or more of these models. While we could conceivably optimize these weights using simple grid search, we are able to find good solutions much more quickly by employing Minimum Error Rate Training (MERT) (Och, 2003) to this end. MERT is particularly suitable for our task since we have a metric (number of correct arcs) that decomposes cleanly across sentences, and a small number of weights. Indeed in the case where we use only two models, MERT is guaranteed to find a set of weights that is globally optimal on the dev set in just one iteration. For experiments involving all three models we initialize the weights to all be negative one, and then perform 20 iterations of MERT, each starting from the output of the last, and searching in the direction of a randomly chosen three-dimensional vector.

EXPERIMENTAL SETUP

Our primary goal is to discover whether dependency-based generative neural models are able to improve the performance of their discriminative brethren, as measured on parsing and language modelling tasks. We also seek to determine the effect construction order and the biases implicit therein has on performance on these two tasks. To this end, we test a baseline discriminative parser, our two models, and all combinations of these three models on a parsing task in several languages, and we test a baseline and our two models' performance on a language modelling task on the same set of languages.

Data Sets

We use the Universal Dependency (UD) corpora (Nivre et al., 2017) for three languages with very different structures: English, Japanese, and Arabic, as provided for the 2017 CoNLL shared task on universal dependency parsing. In all languages we convert all singleton terminal symbols to a special UNK token. See Table 6 for details regarding the size of these data sets.

For Arabic and Japanese the dependency trees were hand-annotated in language-specific formats and then converted automatically to the UD format as part of the UD project. Sentences in the English UD corpus had their POS tags and dependency relations manually annotated natively in the UD format. While these annotations certainly contain some errors and are not precisely aligned with the most recent research on syntax (e.g. Chomsky, 1993b), they are still based on well-principled models (e.g. Chomsky, 1993a) and encode biases that are likely beneficial to the task of language modelling.

For language modelling we evaluate using the gold sentence segmentations, word tokenizations, and part of speech tags given in the data. For parsing, we evaluate in two scenarios. In the first, we train and test on the same gold-standard data using in our language modelling experiments. In the second, we again train on gold data, but we use UDPipe (Straka and Straková, 2017) to segment, tokenize, and POS tag the dev and test sets starting from raw text, following the default scenario and most participants in the CoNLL 2017 shared task.

Baseline Models

On the language modelling task we compare against a standard LSTM-based language model baseline (Mikolov et al., 2010), using 1024-dimensional 2-layer LSTM cells, and optimized using Adam (Kingma and Ba, 2015).

For the parsing task we compare against the discriminative parser of Dyer et al. (2015), a bottom-up transition-based parser that uses stack-LSTMs, as well as the overall top system (Dozat et al., 2017) from the 2017 CoNLL shared task on multilingual dependency parsing (Zeman et al., 2017). That work uses a discriminative graph-based parser that uses a biaffine scoring function to score each potential arc. Moreover, it uses character-level representations to deal with morphology and a PoS tagger more sophisticated than UDPipe – two major changes from the shared task's default pipeline. These two differences afford them a substantial advantage over our approach which only modifies the parsing step of the pipeline.

Finally, we show the results of an oracle system looking at the 1000-best lists used for our reranking experiments. Note that since this oracle system is constrained to using only this list of samples it is not able to achieve 100% parsing accuracy.

Hyperparameters

All models use two-layer 1024-unit LSTMs and 1024-dimensional word/action embeddings. All other MLPs have a single hidden layer, again with 1024 hidden units. We implement all models using DyNet (Neubig et al., 2017a), and train using Adam (Kingma and Ba, 2015) with a learning rate of 0.001, dropout with $p = 0.5$, and minibatches of 32 sentences. We evaluate the model on a held out dev set after 150 updates, and save the model to disk whenever the score is a new best. All other settings use DyNet defaults.

RESULTS

Results on the parsing task can be found in Table 7.

We observe that in English with the gold-standard preprocessing our models perform particularly well, showing an improvement of 1.16% UAS F1 for the top-down and 0.82% UAS F1 for the bottom-up model when individually combined with our discriminative

		English				Japanese				Arabic			
		Gold → Gold		Gold → UDPipe		Gold → Gold		Gold → UDPipe		Gold → Gold		Gold → UDPipe	
Model	Reranked?	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test
CoNLL Baseline	✗	-	-	-	79.24	-	-	-	74.40	-	-	-	70.14
Dozat et al. (2017)	✗	-	-	-	84.74	-	-	-	75.42	-	-	-	76.59
Disc (Greedy)	✗	87.00	85.92	78.85	78.12	96.16	95.20	76.67	75.70	82.14	82.39	69.74	70.34
Disc (Reranked)	✓	87.48	86.30	78.99	78.23	96.04	95.17	76.66	75.58	81.70	81.34	69.20	68.79
Top-Down	✓	82.94	82.48	76.73	76.88	92.99	92.51	74.84	74.18	80.99	80.85	69.78	69.64
Bottom-Up	✓	83.11	82.70	76.79	77.13	94.56	93.25	75.85	74.18	80.61	80.70	69.30	69.24
Disc + TD	✓	88.47	87.46	80.46	79.56	96.07	95.43	76.59	74.62	82.87	82.37	70.35	70.25
Disc + BU	✓	88.29	87.12	80.09	79.33	96.17	95.54	76.82	75.92	82.48	82.18	70.18	69.99
TD + BU	✓	84.93	84.56	78.71	78.72	94.87	94.03	76.15	75.30	81.84	81.56	70.52	70.03
Disc + TD + BU	✓	**88.74**	**87.76**	**80.49**	**80.22**	**96.18**	**95.58**	**76.86**	**75.98**	**83.06**	**82.58**	**70.85**	**70.40**
Oracle	✓	97.68	97.27	91.07	90.24	99.39	99.25	79.67	80.34	91.20	89.06	77.75	76.17

Table 7: Results of parsing using our baseline discriminative parser, our two generative models, combinations thereof, and two contrastive systems from the CoNLL 2017 shared task. Scores in bold are the highest of our models. Note that Dozat et al. (2017) use substantially different preprocessing. See §4.2.2 for details.

		$p(x, y)$		$p(x)$	
Lang.	Model	Dev	Test	Dev	Test
EN	RNNLM	-	-	**5.24**	**5.18**
	Top-Down	5.80	5.72	5.73	5.66
	Bottom-Up	5.63	5.56	5.53	5.47
JA	RNNLM	-	-	**4.41**	**4.58**
	Top-Down	4.82	5.00	4.73	4.93
	Bottom-Up	4.83	5.03	4.75	4.95
AR	RNNLM	-	-	**5.42**	**4.34**
	Top-Down	6.08	6.23	5.98	4.79
	Bottom-Up	6.11	6.21	5.94	4.75

Table 8: Language modelling cross entropy of our model and an RNNLM baseline. Lower is better. All scores are expressed in nats.

parser. Combining all three models together gives a total of 1.46% absolute improvement over the baseline, indicating that the models are able to capture knowledge lacking in the baseline model, and knowledge that is complementary to each other.

The story is similar in Japanese and Arabic, though the gains are smaller in Japanese. We hypothesize that this is due to the fact that parsing Japanese is relatively easy because of its strict head-final and left-branching nature, and thus our baseline is already a remarkably strong parser. This hypothesis is backed up by the fact that the baseline parser alone is only 3-4% UAS away from the oracle by itself, compared to about 10% away on English and Arabic. Thus our relative improvement, measured in terms of the percentage of possible improvement achieved, is quite consistent across the three languages, at roughly 13%.

Results on the test set using UDPipe's noisy preprocessing also saw encouraging results from the three-model ensemble gaining 1.99%, 0.40%, and 1.61% on English, Japanese, and Arabic respectively, solidly outperforming the 2017 CoNLL shared task baselines across the board, and beating Dozat et al. (2017), the overall shared task winner's, submission on Japanese.

Of particular note is that on both the gold and non-gold data, and across all three languages, the performance of the top-down and bottom-up models is quite similar; neither model consistently outperforms the other. In Japanese we do find the bottom-up parser beats the top-down one when used alone, but when combined with the discriminative model the lead evaporates, and in both of the other languages there is no clear trend. We hypothesize that the bottom-up model benefits from the strictly head-final nature of Japanese, which offers a clear signal as to the end of a constitent. English and Arabic, on the other hand, have mixed headedness, allowing the top-down model to outperform the incrementally building bottom-up model.

Overall these results are consistent with Fried et al. (2017) that has shown that generative models are particularly good at improving discriminative models through reranking, as they have an effect similar to ensembling dissimilar models.

We find that despite successes on parsing, our dependency models are not empirically suitable for language modelling. Table 8 shows the performance of our models on the language modelling task. Across all three languages, both of our models underperform a baseline RNNLM by a consistent margin of about 0.5 nats per word.

Again we note that the two models perform remarkably similarly, despite their completely different construction orders, and thus the completely different sets of information

they condition on at each time step. Again neither model is a clear overall victor, and in each individual language the models are extremely close in performance.

ANALYSIS

One of our most intriguing findings is that our two proposed models perform remarkably similarly in spite of their differing construction orders. One would naturally assume that the differing orders, as well as the wildly different history information available at each decision point, would lead to performance differences. We seek to hone in on *why* the two models' performances are so similar.

Unfortunately the fact that the models use different conditioning contexts makes direct comparison of sub-sentential scores impossible. The top-down model, which generates the verb before its subject noun, may have large entropy when choosing the verb, but an easier time choosing the subject since it can condition on the verb limiting its choices to appropriate semantic classes, person, number, et cetera. The bottom-up model, on the other hand, will generate the subject noun from the entire list of possible nouns first, and then will focus its probability on relevant and agreeing verb forms when generating the verb.

To this end we plot the scores (i.e. the negative log probabilities) the models assign to each gold tree in the English dev set. The raw scores between the top-down and bottom-up models are highly correlated (Pearson's $r = 0.995$), largely due to the fact that longer sentences naturally have lower probabilities than shorter sentences. As such, we examine length-normalized scores, dividing each sentence's score by its length. The results are still largely correlated ($r = 0.88$), with a few outliers, all of which are very short (<3 tokens) sentences.

We hypothesize that much of this correlation stems from the fact that for a given sentence both models must generate the same sequence of terminal symbols. Some sentences will have rare sequences of terminals while others have more common words, leading to an obvious, but perhaps uninformative, correlation. To examine this possibility we factor our models' scores into a *terminal* component and a *structure* component so that the overall negative log likelihood of a sentence is decomposed as $\text{NLL} = \text{NLL}_{\text{terminals}} + \text{NLL}_{\text{structure}}$. We then examine the correlation between the two models' scores' terminal components and their structure components separately. We find that the terminal components are still strongly correlated ($r = 0.91$), while the structure components are largely uncorrelated ($r = 0.09$), hinting that information the two models learned about the correct structure of

	Structure	Terminals
Top-Down	4.97	67.9
Bottom-Up	7.42	63.3

Table 9: Our models' average negative log likelihoods on the English dev set broken down into structure and terminal components

English sentences differs. See Figure 16 for a visual representation of these data. Overall the top-down model also assigns much higher probabilities to correct structures, but lower probabilities to the correct terminal sequences (Table 9).

RELATED WORK

Most work on discriminative dependency parsing follows the bottom-up paradigm (Nivre, 2003; Nivre et al., 2007; Dyer et al., 2015; Kiperwasser and Goldberg, 2016), but top-down models have also shown some promise (Zhang et al., 2015).

Most existing generative dependency models whether used for parsing, unsupervised dependency induction, or language modelling (Buys and Blunsom, 2015; Jiang et al., 2016) have relied on independence assumptions. Buys and Blunsom (2015) also describe a generative bottom-up neural parser, but they use hand-crafted input features and limit the model to third-order features. They show that their model is able to out-perform several previous generative parsers (though lagging behind the discriminative Stanford Parser (Chen and Manning, 2014)), and is able to out-perform n-gram language models. Titov and Henderson (2010) is perhaps closest to the models explored in this paper in that define a generative parsing model that makes no independence assumptions. However, rather than using RNNs to encode history, they use incremental sigmoid belief networks (Neal, 1992), which use stochastic latent variables to represent unbounded histories. In their model, trees are generated in a hybrid bottom-up and top-down build order. While their model could, in theory, be applied to language modelling (although the latent variables in the ISBNs make marginalization even more difficult), they do not test their model on that task.

The CoNLL 2017 shared task saw many different models succeed at parsing Universal Dependencies. Most of the top contenders, including the best scoring systems on the languages discussed in this chapter, use discriminative models.

Kanayama et al. (2017) had tremendous success on Japanese using a wildly different approach. They train a model to identify likely syntactic heads, then assume that all other

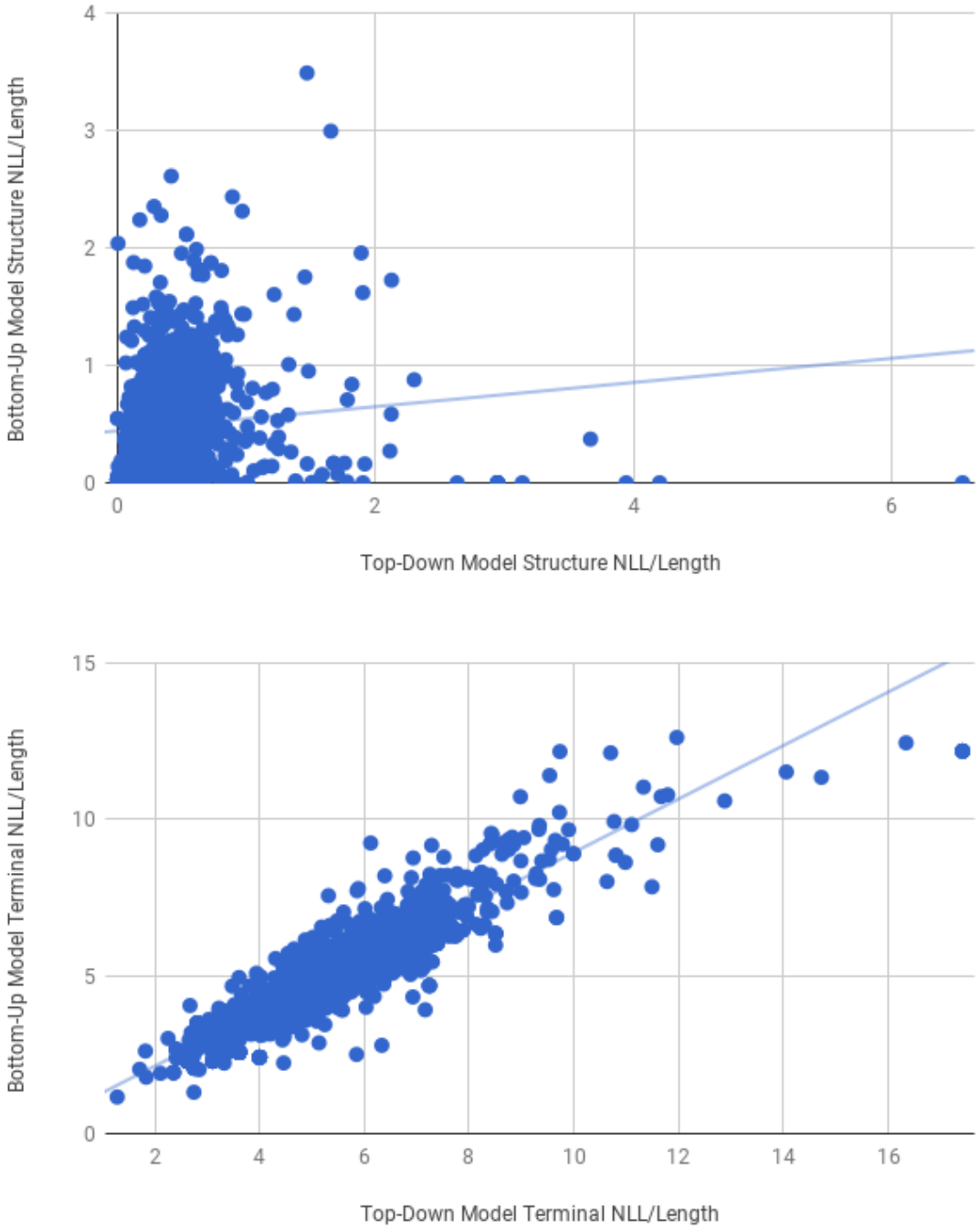


Figure 16: Analysis of the structure (top) and terminal (bottom) scores of our two models' performance on the English development set. We find that the structure scores are not correlated, while the terminal scores of the two models are highly correlated.

words simply attach in a left-branching structure, an assumption that works out well due to the strictly head-final nature of Japanese.

Dozat et al. (2017) train a discriminative neural parser which uses a BiLSTM to generate hidden representations of each word (Kiperwasser and Goldberg, 2016). These representations are then transformed to produce a pair of hidden states – one that represents the word as a dependent seeking its head, and the other that represents the word as a head seeking all its dependents.

Björkelund et al. (2017) perform best on Arabic, using an ensemble of many different types of bottom-up discriminative parsers. They have each of twelve parsers score potential arcs, learn a weighting function to combine them, and use the Chu-Liu-Edmonds algorithm (Chu, 1965; Edmonds, 1967) to output final parses.

All three of these discriminative models are very effective for analysis of a sentence, none of them are able to be converted into a similar generative model. At best, the biaffine model of Dozat et al. (2017) could generate a bag of dependencies without order information, which makes it impractical as the basis for a generative model.

There has been past work on building recurrent neural models that condition on the buffer to make parsing decisions in a shift-reduce parser. Henderson (2004) was among the first to introduce such a model. They introduce both a generative and discriminative model based on Simple Synchrony Networks (Lane and Henderson, 1998), and use a pre-cursor to attention mechanisms to choose which previous states are most relevant at the current timestep. More recently Dyer et al. (2015) created a similar model based on stack LSTMs.

There has also been past work on language modelling with generation orders other than the typical left-to-right. Ford et al. (2018) examine a variety of possibilities, but stop short of syntax-aware orderings. Buys and Blunsom (2018) investigate neural language models with latent dependency structure, also concluding that while dependencies perform well on parsing they underperform for language modelling.

CONCLUSION

In this chapter we test our hypothesis that dependency structures can improve performance on language modelling and machine translation tasks, in the same way that constituency parsers have been shown to help. We developed two new dependency-based models and test their effectiveness on language modelling and parsing tasks. We conclude that generative dependency models do indeed make very good parsing models and, as has been

observed in phrase structure parsing, combining a generative dependency parser with a traditional discriminative one does indeed improve parsing performance. We however also find that using dependency information to structure the intermediate representations in language modelling does not easily lead to better outcomes than using linguistically naïve models.

This pattern of results has a more complicated interaction with the thesis statement. We know that phrase-structure trees improve language modelling, and thus that that kind of syntactic knowledge does indeed provide good biases to improve generalization. The biases learned from dependencies, on the other hand, do not help. We conclude that the biases inherent to dependencies may be a useful tool for text analysis, but are less suited to characterizing a generation process of sentences than phrase structure grammars are. Finally, we find that the choice of top-down or bottom-up construction order affects performance minimally on both the parsing and language modelling tasks despite the large differences in the local conditioning contexts of each action choice.

5

SYNTAX AND NEURAL MACHINE TRANSLATION

In the previous chapter we introduced syntax into language models for various languages. We will now turn our attention to *conditional* modelling of language using syntax. We examine the problem of machine translation (MT) from a variety of languages into English and the effects of syntax-aware models thereupon. This task is particularly interesting due to its inherent multilingualism, allowing us to model two languages' syntax simultaneously. This exposes our models to more varied syntactic structures and forces them to learn correlations between potentially very different structures.

We posit that syntax-aware models should excel at machine translation for two reasons. The first is that sentence-aligned parallel data is much more scarce than monolingual data. We know that linguistics-aware models are more sample efficient than naïve models for unconditional language modelling (Dyer et al., 2016; Eriguchi et al., 2017), and thus suspect that the same would hold true for conditional language modelling. Furthermore, while monolingual data suitable for language modelling is passively produced by billions of people in thousands of languages every day, parallel data for machine translation is a much more limited resource. The sample efficiency gains brought by syntax should thus be even more impactful on this problem.

The second reason is that we have seen historically that syntax-based statistical translation models were able to out-perform naïve phrase-based machine translation (Galley et al., 2004; Chiang, 2005; Lavie et al., 2008; Hanneman et al., 2011). Most previous tree-to-tree approaches, however, make an unrealistic assumption that the source- and target-language trees are fully isomorphic. Quasi-synchronous grammars (Smith and Eisner, 2006; Gimpel and Smith, 2011) offer a solution by learning to use well-aligned subtrees while still allowing for some disparity between the overall tree structures. They permit a more direct account of the fact that there may be large syntactic divergences (e.g. head swapping) between languages. Symbolic syntax-based systems (both fully- and quasi-synchronous) were slow and brittle, relying on sparse statistics and complex combinatorial global search algorithms. Neural models, on the other hand, are able to capture complex distributions found in the data by allowing, but not requiring, the model to look at some or all of a source

tree while generating a target tree. Furthermore these neural models yet admit decoding algorithms based on beam search. They promise to harness the power of syntax while more easily generalizing from limited data and side-stepping the need for slow heuristic search.

To this end, we propose syntax-aware input and output models to perform quasi-synchronous translation, in which one loosely conditions on a source language syntax tree while producing a target language syntax tree. We use bidirectional TreeGRUs to embed the source tree as they have been shown to be outperform several other models as translation encoders (Chen et al., 2017a). We expect that a syntax-aware encoder will allow the model to better understand long-distance dependencies in the source sentence, more easily understanding the input sentence's meaning. In the previous chapter we saw that our dependency-based models underform on language modelling tasks. As such, we use RNNGs (Dyer et al., 2016), as our target-side syntactic language model. RNNGs have been shown to be strong language models, capable of capturing the grammaticality of sentences, ensuring the validity of our translation output. We experiment with syntax on one side, the other, or both.

While our quasi-synchronous tree-to-tree translation system represents a novel approach to neural MT there has been much prior work on incorporating various types of syntax into NMT. Most work focuses primarily on using syntax to do source-language encodings either using dependencies (Wu et al., 2017b; Chen et al., 2017b; Bastings et al., 2017) or phrase-structure trees (Eriguchi et al., 2016; Chen et al., 2017a). Target-side syntax represents a much further departure from baseline models. Some work has focused on the less ambitious task of generating linearized trees (Aharoni and Goldberg, 2017; Nadejde et al., 2017a), but more recent work has explored fully neural dependency (Wu et al., 2017a) and phrase-structure models (Eriguchi et al., 2017). This chapter takes the best of both worlds, combining the approaches of Eriguchi et al. (2016) and Eriguchi et al. (2017) to do end-to-end tree-to-tree NMT.

This chapter examines the effects of syntactic knowledge on machine translation between English and several typologically different languages, including Arabic, Chinese, French, and German.

MODEL

Our model is based on the attention model (Bahdanau et al., 2015), which consists of three parts: an encoder, a decoder, and an attention mechanism. We replace the baseline RNN-

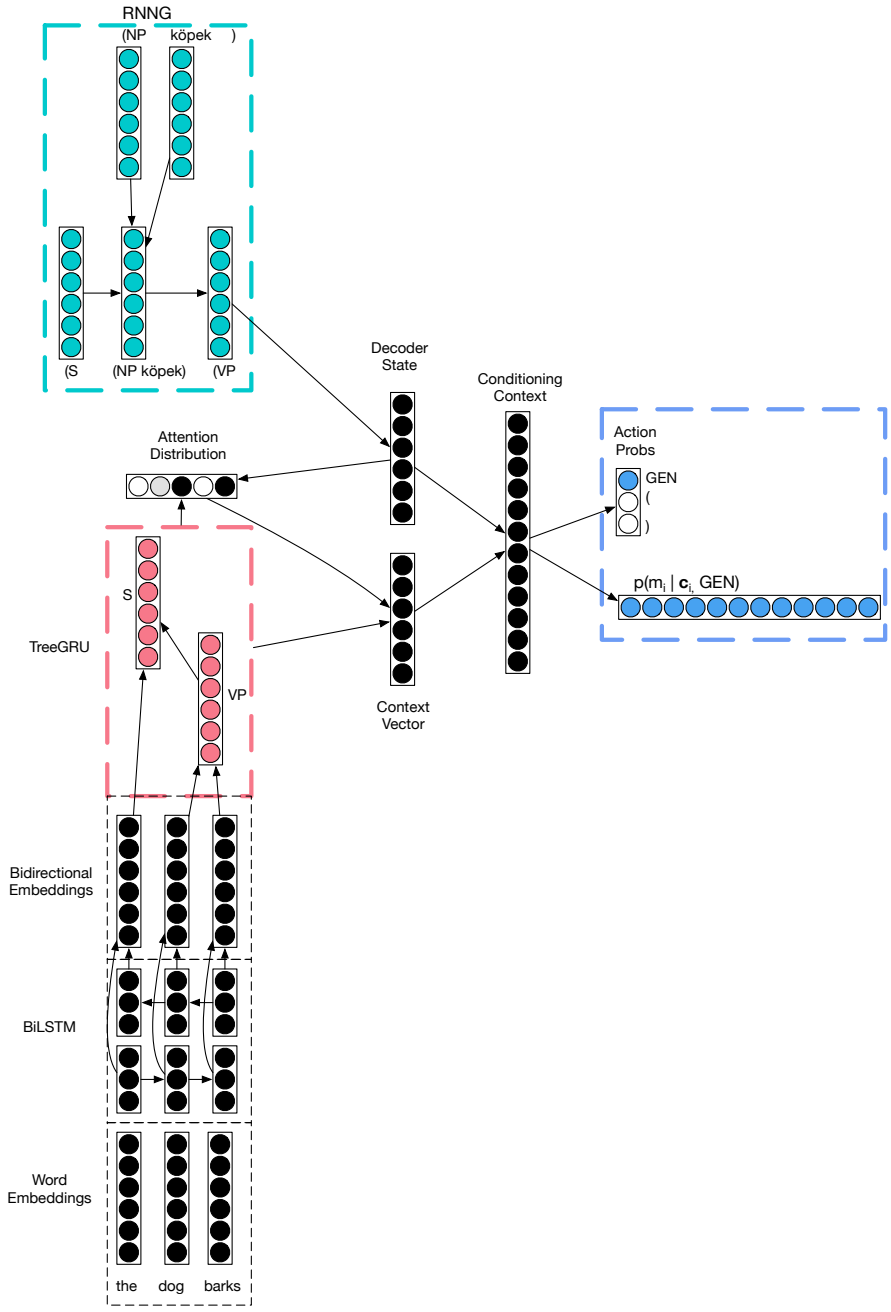


Figure 17: Our syntax-aware machine translation system runs a BiLSTM over input word vectors to create a set of context-aware word embeddings, followed by a TreeGRU to get a final set of embeddings for each node in the source-side tree. Simultaneously a stack LSTM is used to embed the partial target syntax tree, and produce the decoder's hidden state. The decoder's hidden state is compared to each context-aware word embedding to get an attention score. The attention scores are then passed through a softmax to get an attention distribution. The context-aware embeddings are then dotted with the attention distribution to get a context vector. The context vector and decoder state are then combined to create the full conditioning context, used to predict the next action in the construction of the target side syntax tree. Here the model predicts the GEN action, indicating that it wishes to generate a terminal. The model re-uses the single conditioning context vector to predict a single terminal word. Departures from the baseline encoder are marked in salmon. The RNNG that encodes the partially built target-language trees is marked in teal. The factored RNNG action prediction output layer is marked in light blue. Together the teal and light blue components represent the departures from the baseline decoder.

based encoder and decoder with syntax-based alternatives. Below we outline each piece of this system in turn.

Syntactically-Aware Encoder Models

The baseline encoder model uses a BiLSTM to encode the source sentence. Words are embedded in a vector space (using either raw word embeddings or our morphological model from Chapter 3), and then passed through both a forward and backwards LSTM. This produces a pair of hidden states for each word, one from the forward LSTM and one from the backward LSTM, which are then combined to create a single encoded vector representation for each word.

The syntax-aware encoder uses a syntax tree rather than a linear ordering to combine terminal embeddings. Furthermore, it produces an embedding for every node in the syntax tree, rather than just the terminals. The final embedding of each node is a combination of two parts: an inside embedding, which encodes all the information about the node and its descendants, and an outside embedding, which encodes the rest of the syntax tree.

The inside embeddings are created bottom-up, starting at the leaves. Each leaf of the syntax tree is a terminal, and since it has no descendants it suffices to have the inside embedding simply be the encoding of the terminal (either the raw word embedding or the output of the BiLSTM over terminals). For each other node in the tree, its inside embedding is computed using a GRU whose inputs are the embedding of the node's label (e.g. NP, VBZ, etc.) and the embeddings of its children.

The outside embeddings are created top-down, starting at the root. The root node's outside embedding is created using a non-linear transform of its inside embedding. For each other node, its outside embedding is a GRU whose inputs are its own inside vector and its parent's outside vector.

Finally, the inside and outside embeddings are summed to produce a single vector for each node in the syntax tree. All of these embeddings, including those representing non-terminals, are available for the attention model to attend to, allowing the decoder to focus on a whole constituent, rather than just terminals.

This encoder model is based on the work of Chen et al. (2017a), but is extended to use non-terminals labels in both the inside and outside embeddings.

Syntactically-Aware Decoder Models

The baseline decoder model uses a single (foward) LSTM that additionally conditions on the *context vector* output by the attention mechanism. At each time step the model conditions on the hidden state of the LSTM (thus indirectly conditioning on all previously generated output words) and the context vector in order to predict (a distribution over) the next output word.

The syntax-aware decoder model uses an RNNG to construct a parse tree of the target sentence in a top-down fashion. Instead of directly predicting individual words, the RNNG decoder constructions each sentence as a series of actions.

In the original RNNG framework there are three types of actions: The SHIFT(w) action takes a word w and pushes it on to the stack. The NT(n) action takes a non-terminal label n and pushes it on to the stack. The REDUCE action pops items off the stack up until (and including) the first non-terminal label it encounters. It then combines all these elements into one subtree and pushes the result back on to the stack.

In this work, we modify the above framework to better fit our use case of labelled binary trees. To this end, we make the REDUCE action implicit. Whenever the top-most open non-terminal has two completed children we automatically perform the REDUCE action. This means that each SHIFT action may be followed by any number of implicit REDUCE actions. This also means that the REDUCE action is never explicitly used, except for in the case of sentences containing just one word.

At each time step the decoder gives a distribution over subsequent actions $p(a_t|a_1 \dots a_{t-1})$ factored as

$$\begin{cases} p(a_t = \textsc{shift}|a_1 \dots a_{t-1}) \cdot p(w|a_1 \dots a_{t-1}, a_t = \textsc{shift}) & \text{if } a_t = \textsc{shift} \\ p(a_t = \textsc{nt}|a_1 \dots a_{t-1}) \cdot p(n|a_1 \dots a_{t-1}, a_t = \textsc{nt}) & \text{if } a_t = \textsc{nt} \\ p(a_t = \textsc{reduce}|a_1 \dots a_{t-1}) & \text{if } a_t = \textsc{reduce} \end{cases}$$

The distribution over the three action types, $p(a_t \mid a_1 \dots a_{t-1})$ is computed as $\text{softmax}(W \cdot s + V \cdot c + b)$, where s is an embedding of the decoder's current state, as shown below, c is the context vector from the attention mechanism, and W, V, and b are learned parameters. The distributions $p(w \mid \cdot)$ and $p(n \mid \cdot)$ are implemented similarly as a softmax over the relevant vocabulary following a linear transform.

The state vector is the sum of three component vectors: 1. The output of an LSTM over previous actions 2. The output of an LSTM over previously generated terminals 3. The output of an LSTM over the elements on the stack, from oldest to newest.

When an action a is executed, any required stack manipulations are performed and it is added to the end of the action LSTM. If a is a SHIFT action, its word is added to the end of the terminal LSTM. For REDUCE actions (be they implicit or explicit), elements up to and including the most recent non-terminal are popped from the stack. They are then combined via a BiLSTM. In each direction, the LSTM is seeded with the embedding of the most recent non-terminal n, and then passed the vectors representing each child in term. The output of the forward and backward LSTMs are summed, and the result is pushed back onto the stack. For NT(n) actions, the only stack manipulation is to push the embedding of the non-terminal n onto the stack. For SHIFT(w) actions, the embedding of the word w is pushed onto the stack. Then implicit REDUCE actions are triggered until the stack no longer has two complete constituents on top.

BATCHING

Modern graphics cards have the ability to perform the same computation in parallel on many different inputs. Neural networks benefit greatly from leveraging this capability during training (Oh and Jung, 2004). For example, say a network needs to compute a matrix-vector product Wx_i for each input example i. Given a batch of b input examples, one may stack the vectors $x_1 \dots x_b$ into one matrix X and then compute one easily parallelizable matrix-matrix product WX.

While optimally performing this batching on a given computation graph is NP-hard (Neubig et al., 2017b), several neural network toolkits have used heuristics to automatically optimize computation graphs (Neubig et al., 2017b; Bradbury and Fu, 2018). While these tools offer low-effort approximate solutions practitioners typically achieve substantial speed-ups with hand-engineered batching schemes specific to their particular computation graphs (Junczys-Dowmunt et al., 2018, e.g.). This is usually done by combining the largest groups of like computation nodes, while still respecting the topological ordering of the graph.

When using a simple RNN the topology of the computation graph is a simple left-to-right flow from timestep 1 to timestep T. Since the computations to advance from timestep t to timestep $t+1$ use the same operators and parameter matrices across all input examples,

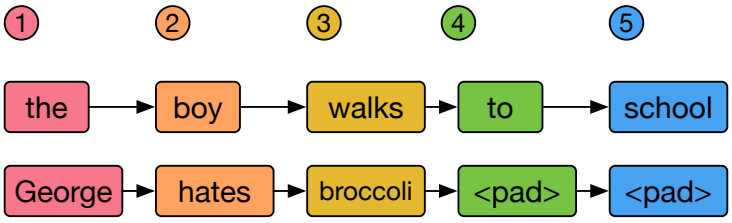


Figure 18: Batching in a typical RNN.

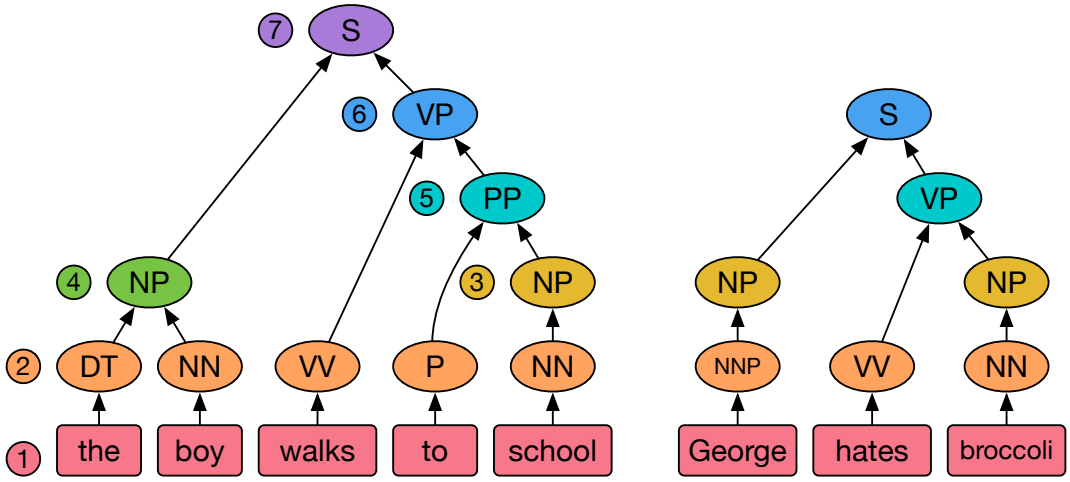


Figure 19: Bottom-up encoder batching.

it is relatively straight forward to stack the b input vectors x_i into one input matrix X and perform even relatively complicated computations (e.g. the computations required to advance an LSTM by one step) in parallel for all b examples (Figure 18).

Given a minibatch of b input sentences, each of which has a maximum length of N, computation for a simple RNN proceeds as follows. First, any sentences with length less than N are padded with a special <PAD> token. Next the RNN's initial state vector s_0 (of dimension d is duplicated b times to form a $b \times d$ matrix. Then, instead of performing computations on each of the b state vectors individually, the system can perform the computations to advance from step t to $t+1$ (e.g. $s_{t+1} = \tanh(Ws_t + b)$) on the entire matrix at once, resulting in a matrix whose ith row represents the RNN state of the ith of the b inputs after $t+1$ words. Finally, a mask is typically applied, multiplying outputs corresponding to <PAD> tokens by 0, thereby resetting them and indicating that they are not used downstream.

Syntax trees, however, lack this nice topological structure since each input sentence will have a different tree structure. For example, one sentence may have its first two words directly combining into a constituent (Figure 19, left), but another sentence may have the second and third word combine, with the first word attaching higher up in the tree (Fig-

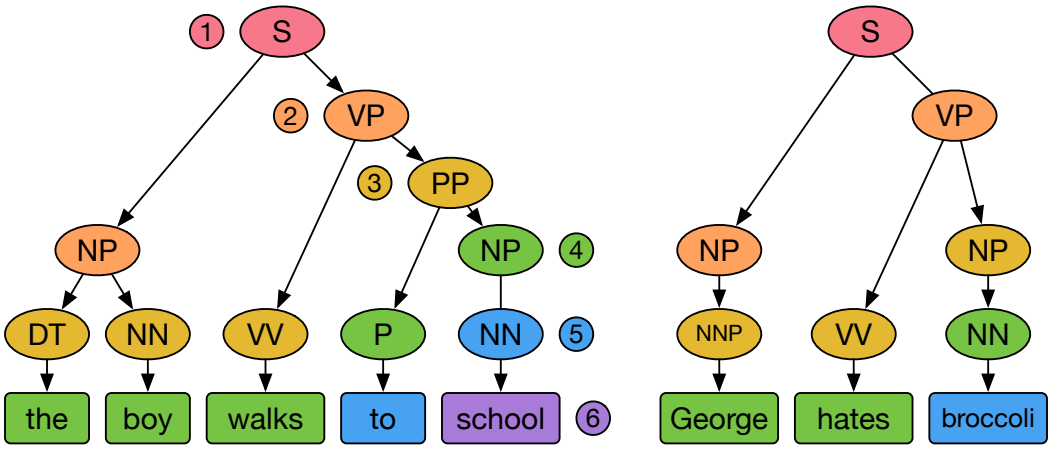


Figure 20: Top-down encoder batching.

ure 19, right). As such, naïve batching schemes do not work well for syntax trees, leading us to seek an alternative.

In our syntax-based encoder, however, nodes are not encoded linearly. In general it is not the case that the ith word in each of the b input sentences will attach to the same place in the overall tree structure. Nevertheless there does exist substructure in the input syntax trees that we can exploit to perform batching. For one, if we topologically sort input trees such that nodes always succeed their children (Figure 19) then the bottom-up pass of our tree encoder model can be efficiently batched. In fact, not only can we batch multiple input examples together, but we can additionally do a surprising amount of batching within a single input tree! Given $b \geqslant 1$ trees all the pre-terminals, for example, can be batched together, their states computed in parallel given the terminals beneath them.

Figure 19 shows how this scheme works on a batch of two input sentences. Notice that the computation of all eight pre-terminal nodes can happen in parallel, with batching happening both intra- ane inter-sententially. Similarly, all the binary nodes at level ③ (for example) can be computed together.

Formally, we define the distance-to-leaf (DTL) of a node to be 0 if that node is a terminal and one more than the maximum of it's childrens' DTLs otherwise.

$$\text{DTL}(n) = \begin{cases} 0 & \text{if } n \text{ is a terminal} \\ \max_{c \in \text{children}(n)} \text{DTL}(c) + 1 & \text{otherwise} \end{cases}$$

Now we can sort nodes primarily by their distance-to-leaves and then secondarily by their arity. Then (in the bottom-up pass), we can batch all tree nodes that share these two properties.

For the top-down pass we can perform a similar trick, this time sorting by distance-to-root instead of distance-to-leaf (Figure 20). Note that for the top-down pass we needn't

	Train			Dev			Test		
Lang. Pair	Sents	Src Words	Trg Words	Sents	Src Words	Trg Words	Sents	Src Words	Trg Words
Ar–En	214K	14.7M	14.9M	4711	315K	325K	5950	393K	397K
De–En	168K	10.1M	11.4M	4148	250K	281K	4489	261K	289K
Fr–En	192K	13.5M	13.1M	4317	304K	292K	4864	327K	314K
Zh–En	200K	13.8M	13.8M	4558	299K	313K	5251	345K	344K

Table 10: Statistics about the WIT³ corpus used in our translation experiments.

sub-sort by arity since each node's top-down embedding is a function of only its parent's top-down embedding and the node's own bottom-up embedding.

Note that while the experiments in this chapter focus exclusively on binarized trees, our batching approach is more general. It can be applied to trees of any mixed arity.

Our approach has several advantages over the type of batching used for simple RNNs. First, note that no padding is required in either the top-down or bottom-up passes. Second, we are able to batch computations both across multiple input trees and within individual input trees. Third, note that to encode a sentence of length n an RNN must perform n steps while our model need only perform dA steps, where d is the depth of the tree and A is the number of unique node arities within the tree. Since on average $d \approx \log(n)$ this is a significant benefit.

EXPERIMENTS

We compare systems with and without our syntax-enhanced encoder and decoder across four language pairs: Arabic–English, German–English, French–English, and Chinese–English. We use the WIT³ corpus of TED talks [1], statistics about which can be found in Table 10. We use the Berkeley Parser (Kummerfeld et al., 2012, 2013) (with its `--binarize` flag) to parse both sides of the corpora, discarding any sentences that fail to parse. We use the pre-trained models packaged with the Berkeley Parser, which are trained on per-language treebanks created by automatically annotationg news articles and manually correcting the results.

Models are implemented using xnmt (Neubig et al., 2018) and DyNet (Neubig et al., 2017a). They are trained using Adam (Kingma and Ba, 2015) with a learning rate of 0.001, dropout with $p = 0.2$ and minibatches of 64 sentences, and use 512-dimensional hidden states and 128-dimensional words embeddings. Models are trained end-to-end to optimize maximum likelihood of the gold trees produced by the Berkeley Parser. Each time perfor-

1 https://wit3.fbk.eu/

		Ar–En			De–En			Fr–En			Zh–En		
Enc.	Dec.	BLEU	MTR	Len	BLEU	MTR	Len	BLEU	MTR	Len	BLEU	MTR	Len
str	str	**30.23**	**31.32**	**99.26**	27.02	28.59	**96.10**	**36.33**	**34.61**	**97.04**	**16.76**	**23.00**	91.93
str	tree	21.34	25.05	83.29	15.18	25.57	81.79	28.75	29.75	87.28	11.23	17.88	79.35
tree	str	26.87	28.77	97.51	**27.24**	**29.03**	94.61	32.09	31.73	95.69	15.09	21.72	**96.54**
tree	tree	21.14	24.72	91.52	11.60	13.05	93.67	25.72	27.27	91.24	11.77	18.56	85.60

Table 11: Machine translation ablation results on our dev set, ablating use of the syntactic encoder (Enc.) and syntactic decoder (Dec.).

mance on the held-out dev set fails to improve three epochs in a row the learning rate is halved. This process is repeated until the learning rate would be halved a fourth time. The final models are chosen via early stopping, optimizing performance on the dev set. During decoding we use standard beam search with a beam width of 5, a maximum length of 500, and no length normalization, unless otherwise noted. System outputs are evaluated using BLEU (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2014), and length ratio. Preliminary results on our held out dev sets can be found in Table 11. The results (and their shortcomings) inspire the investigations and solutions described in the remainder of this chapter.

SYNTAX AND LENGTH EFFECTS

Table 11 shows that our string-to-string baseline beats our syntax-informed models by a substantial margin in three of the four languages, and is only narrowly edged out by a tree-to-string system in German. The length ratios in Table 11 show one big reason why: the syntax-based systems tend to severely undergenerate terminals, leading to a hard-hitting BLEU length penalty.

We see that all the systems undergenerate to some extent, and that the systems that output trees undergenerate to a much worse extent. Neural MT models undergenerating is a known phenomenon (Murray and Chiang, 2018), and in fact Stahlberg and Byrne (2019) experiment with three popular NMT architectures and find that for more than half of the time the best output by model score is the empty string. While all standard NMT systems undergenerate, our syntax-based models exhibit this effect to a greater extent. We investigate several techniques to combat this undergeneration effect. First, we try standard length normalization models, including both additive and multiplicative normalization schemes. Second, we demonstrate the effects of changing the maximum legnth parameter during

decoding, as well as the effect of capping the depth allowed in generated trees. Third, we use a modified search algorithm when decoding using RNNGs.

Additive and Multiplicative Normalization

Two simple and widely used techniques for correcting length bias in translation output are additive and multiplicative normalization.

Additive normalization (also known as a "word bonus" or "word penalty") simply adds a score bonus proportional to the length of the output: $s' = s + cL$, where s is the original score of a given sentence, L is the sentence's length, s' is the normalized score, and c is a constant that controls the strength of the normalization (Neubig, 2016).

Multiplicative normalization modifies the score by dividing it by the length of the sentence raised to some power: $s' = \frac{s}{L^c}$ (Wu et al., 2016). One popular choice is to use $c = 1$ (Koehn and Knowles, 2017), which corresponds to comparing hypotheses based on their average per-word score.

While both of these approaches have similar overall effects, Murray and Chiang (2018) show that additive normalization has a small empirical edge in addition to being more theoretically justifiable.

Length and Depth Constraints

Since certain quirks of our trained models result in repetition of open non-terminals we examine a very simple and pragmatic way to alleviate length issues: limiting the sentence length and/or depth of the output trees. For sentence length limits we institute a hard cap on the number of actions an output tree can contain. For tree depth we limit the number of non-terminals that can be open at any given time.

Results using length and depth caps can be seen in Figure 21. We find that both of these values have large impact on translation quality, with poor settings performing extremely badly and only slight changes resulting in swings of half a BLEU or more. Good settings of either parameter can bring the system's length ratio very close to one, but the optimal setting of the depth parameter results in nearly +2 BLEU over the optimal setting of the maximum length.

Though these two settings both accomplish the same task, controlling the length ratio, one might naturally ask if these two approaches could be combined to yield further im-

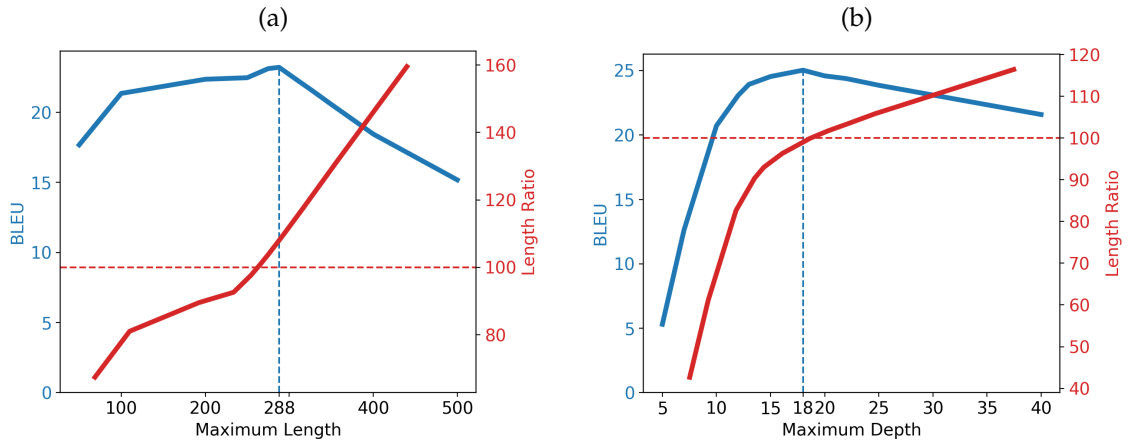


Figure 21: The effect of changing (a) maximum length and (b) maximum depth at decoding time in our German–English string-to-tree system. Note: 99.7% of training trees have depth $\leqslant 40$.

provements. Unfortunately, combining the two techniques caps out at 25.02 BLEU, yielding no further gains over using either one alone.

In addition to simply capping the maximum length at some arbitrary (if empirically chosen) depth, we also experiment with predicting the maximum depth. We experiment with three such models. The first simply uses the training data to fit a logistic curve that predicts tree depth given source length. Other types of curves, including linear, polynomial, and exponential curves were considered, but the logistic curve was used because it had the best r^2. The second model bins the training sentences by their source length and computes the mean and standard deviation of the tree depths within each bin. Finally, we fit a linear model to the means and a logistic model to the standard deviations, allowing us to predict depths for source sentences of unseen lengths. We then set the maximum depth to be $\mu + k\sigma$ for $k \in \{1, 2, 3\}$. Finally, we trained two neural networks that look at the whole source sentence and attempt to predict the target tree's depth. The first network ("NN MSE") is trained to minimize squared error. The second network ("NN Asym")'s loss function is quadratic if the predicted depth is too small but only linear if the predicted depth is too large. This loss function encodes the intuition that too small a limit is very bad, since the reference tree may become unreachable. Too large a limit, on the other hand, is much less severe a problem, just allowing the model to overgenerate a bit.

Results of these approaches are found in Table 12. We find that our second neural model is the best of the lot, but even that one is no stronger than the stubbornly strong baseline of using a fixed maximum depth.

Model	BLEU	METEOR	Length
Logistic	24.62	27.33	96.79
Gauss ($\sigma = 1$)	24.56	**27.83**	102.98
Gauss ($\sigma = 2$)	23.78	27.71	106.21
Gauss ($\sigma = 3$)	23.00	27.64	109.86
NN/MSE	24.48	27.20	96.13
NN/Asym.	**25.02**	27.68	**100.73**

Table 12: Comparison of different models of tree depth given source sentence on our German–English string-to-tree system.

Better Beam Search for Syntactic Output

Generative models are notoriously hard to decode with, and RNNGs are no different. First, we would ideally like to find the best translation string, marginalizing out the tree structure: $\hat{e} = \arg\max_e \sum_{t \in \mathcal{T}(e)} p(t, e \mid s)$, where $\mathcal{T}(e)$ gives the set of all possible tree structures over a sentence e. Since the number of possible tree structures is exponential in sentence length this computation is intractable, we note that most sentences have one parse tree that is much more likely that the others and thus approximate $\sum_{t \in \mathcal{T}(e)} p(t, e \mid s) \approx \max_{t \in \mathcal{T}(e)} p(t, e \mid s)$.

Second, when decoding with an RNNG the NT actions almost always have probabilities much higher than the SHIFT actions, simply because there are several orders of magnitude more possible SHIFT actions (one per terminal type) than NT actions (one per non-terminal type) (Fried et al., 2017). This leads to the model vastly over-producing non-terminals and under-producing terminals. In fact, in preliminary experiments for this chapter, we found many English output "sentences" where the system had simply repeatedly output NT(S) over and over until the maximum decoding length was reached.

Fried et al. (2017)'s solution to this issue on their parsing task is Word-Synchronous Beam Search. Normal action-level beam search bins hypotheses by the number of *actions* taken so far, and keeps only the top k from each bin. Word-Synchronous Beam Search instead bins hypotheses by the number of *terminals*, and again keeps only the top k.

Algorithmically, they label each bin $(|W|, |A_w|)$, where W is the set of terminals generated thusfar and A_w is the set of non-terminal actions taken since the most recent SHIFT action. The algorithm begins with the empty hypothesis in bin $(0, 0)$. Just like in standard beam search, the algorithm expands each of the (up to) k hypotheses in a bin with the top k continuations each, producing k^2 new candidates. In standard beam search the candidate continuations based on hypotheses from bin i are put into bin $i + 1$. In Word-Synchronous

beam search, when expanding hypotheses in bin (i, j), expansions ending with a SHIFT action are placed into bin $(i+1, 0)$, reflecting the fact that have one more terminal than before, and zero NT actions since the most recent shift. Expansions from bin (i, j) that end in an NT action, however, are placed into bin $(i, j+1)$, because they have the same number of terminals as before but one more NT action since the most recent shift. This means that in each bin we are comparing hypotheses with a strictly equal number of SHIFT actions, negating the advantage the NTS have in terms of local probability.

Algorithm 2 Word-Level Beam Search (Fried et al., 2017)

```
1:  procedure WordLevelBeamSearch(initial_state, beam_size)
2:      # A hypothesis is (score, state, word, parent)
3:      hyps = [(0, initial_state, <s>, NONE]
4:      complete_hyps = ∅
5:      for length in range(max_length) do
6:          if len(complete_hyps) ⩾ beam_size then
7:              complete_hyps = sorted(complete_hyps, reverse = TRUE)[: beam_size]
8:              hyps = filter(hyps, λ hyp : hyp.score ⩾ complete_hyps[−1].score)
9:              if len(hyps) == 0 then
10:                 break
11:         new_shift_hyps = []
12:         for num_struct_actions in range(max_length - length + 1) do
13:             new_nt_hyps = []
14:             fasttrack = (len(num_shift_hyps) == 0)
15:             for hyp in hyps do
16:                 new_state = translator.add_word(hyp.state, hyp.word)
17:                 if new_state.is_complete() then
18:                     complete_hyps.append(hyp)
19:                     continue
20:                 candidates = translator.k_best(new_state, beam_size)
21:                 if fasttrack then
22:                     candidates += translator.best_shifts(new_state, beam_size)
23:                 for (word, score) in candidates do
24:                     new_hyp = (hyp.score + score, new_state, word, hyp)
25:                     if word.is_shift() then
26:                         new_shift_hyps.append(new_hyp)
27:                     else
28:                         new_nt_hyps.append(new_hyp)
29:             hyps = sorted(new_nt_hyps, reverse = TRUE)[: beam_size]
30:         hyps = sorted(new_shift_hyps, reverse = TRUE)[: beam_size]
31:     return sorted(complete_hyps)[−1]
```

One other technique employed by Fried et al. (2017) is "fast-tracking" the top k SHIFT continuations from bin (i, j) to bin $(i+1, j)$. This ensures that even if all of the top k overall actions are NT actions at least some SHIFT actions are explored and that infinite chains of nt actions are impossible.

Algorithm 3 Bag-Level Beam Search (Mabona et al., 2019)

```
procedure BagLevelBeamSearch(initial_state, beam_size)
    # A hypothesis is (score, state, word, parent)
    bins[0,0] = [(0, initial_state, <s>, NONE]
    complete_hyps = ∅
    for num_nts in range((max_length + 1) / 2) do
        for num_shifts in range(num_nts + 2) do
            bin = bins[num_shifts, num_nts]
            if len(complete_hyps) ⩾ beam_size then
                complete_hyps = sorted(complete_hyps, reverse = TRUE)[: beam_size]
                bin = filter(bin, λ hyp : hyp.score ⩾ complete_hyps[−1].score)
            bin = sorted(bin, reverse = TRUE)[: beam_size]
            for hyp in bin do
                new_state = translator.add_word(hyp.state, hyp.word)
                if new_state.is_complete() then
                    complete_hyps.append(hyp)
                    continue
                candidates = translator.k_best(new_state, beam_size)
                for (word, score) in candidates do
                    new_hyp = (hyp.score + score, new_state, word, hyp)
                    if word.is_shift() then
                        bins[num_shifts + 1, num_nts].append(new_hyp)
                    else
                        bins[num_shifts, num_nts + 1].append(new_hyp)
    return sorted(complete_hyps)[−1]
```

More recently Mabona et al. (2019) discuss a left-branching bias in the word-level decoding algorithm described above and propose Bag-Level Beam Search as a solution. This is of particular concern when decoding into English, a language with a tendancy towards right-branching structures (Van Riemsdijk and Williams, 1986). Where Word-Synchronous Beam Searched binned hypotheses by the number of SHIFT actions and the number of NT actions since the most recent SHIFT, Bag-Level Beam Search simply bins them by the number of SHIFT actions and the number of NT actions.

A visual comparison of word- and bag-level beam search is shown in Figure 22.

We note that both Fried et al. (2017) and Mabona et al. (2019) examine the problem of decoding with RNNGs in the context of parsing. As such, they are able to know the length of the input string a priori, and avoid dealing with length effects.

To test the effectiveness of the methods we focus on our German–English string-to-tree system and report development BLEU scores using additive length normalization, mul-

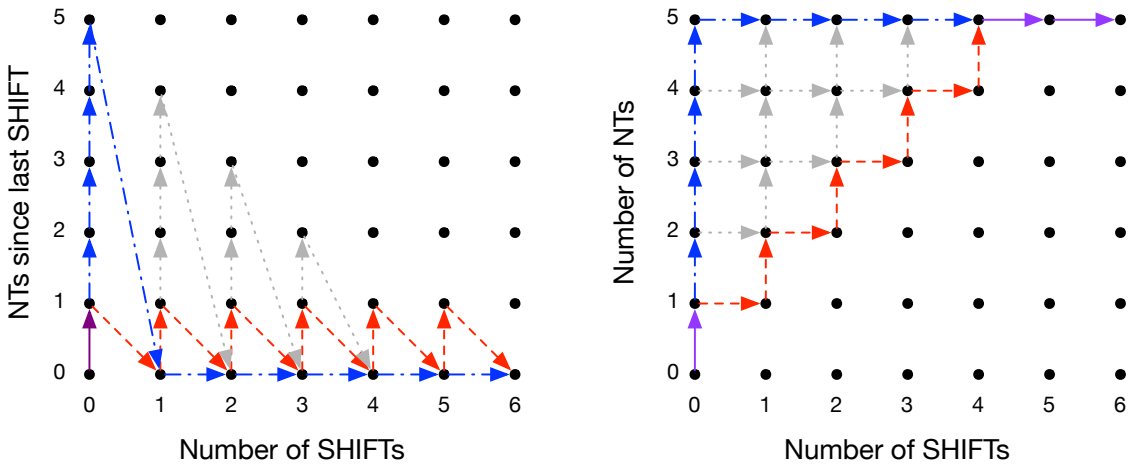


Figure 22: A comparison of word-level (left) and bag-level (right) beam search algorithms. Transitions taken by a left-branching tree structure are shown in blue dot-dashed lines. Transitions taken by a right-branching tree structure are shown in red dashed lines. Transitions belonging to both are shown in solid purple lines. Other possible transitions are shown in grey dotted lines. Figure inspired by Mabona et al. (2019).

Technique	BLEU	METEOR	Length Ratio
(None)	15.18	25.57	159.52
Max Length	23.21	25.93	100.77
Max Depth	25.03	27.63	**99.93**
Additive Length Norm	25.69	27.71	92.53
Multiplicative LN	16.12	27.67	159.15
Word-Level Search	22.95	26.11	83.30
Word-Level Search + ALN	25.74	27.92	99.83
Bag-Level Search	8.47	15.68	42.63
Bag-Level Search + ALN	**25.94**	**27.98**	98.48

Table 13: Effects of techniques aimed at fixing the length ratio of our German–English string-to-tree system.

tiplicative length normalization, word-level beam search, and bag-level beam search [2] in Table 13.

These results show that the search procedure used has a drastic effect on the quality of the resulting translations. Stahlberg and Byrne (2019) show that nearly all translation models are woefully biased towards short translations and that it is only through interactions with imperfect search that reasonable translations can be produced. Adding target side structure increases the complexity of decoding and exacerbates the effects of short-sighted decoding, necessitating the use of specialized inference algorithms. We note that Bag-Level

2 Due to the $\mathcal{O}(n^2)$ complexity of bag-level beam search, we use a maximum length of 100 for bag-level search experiments.

		Ar–En			De–En			Fr–En			Zh–En		
Enc.	Dec.	BLEU	MTR	Len	BLEU	MTR	Len	BLEU	MTR	Len	BLEU	MTR	Len
str	str	**30.37**	**31.47**	100.23	27.50	28.67	99.79	36.85	34.93	**100.01**	**17.54**	**23.70**	99.37
str	tree	25.88	28.64	**100.08**	25.94	27.98	98.48	32.13	32.20	**100.01**	14.63	21.39	**99.59**
tree	str	28.67	30.42	99.79	**28.63**	**30.24**	**100.04**	**37.23**	**35.06**	99.91	16.96	23.33	99.26
tree	tree	24.79	27.90	100.33	25.01	27.66	99.44	30.58	31.59	100.92	14.14	21.02	101.51

Table 14: Translation results on the dev set using additive length normalization and, where relevant, bag-level decoding.

		Ar–En			De–En			Fr–En			Zh–En		
Enc.	Dec.	BLEU	MTR	Len	BLEU	MTR	Len	BLEU	MTR	Len	BLEU	MTR	Len
str	str	**29.20**	**30.92**	101.29	29.76	29.89	100.60	**37.47**	**35.30**	100.21	**18.05**	**24.49**	**100.05**
str	tree	24.81	28.24	102.61	27.12	29.12	**100.33**	32.21	32.99	102.59	14.48	22.04	103.01
tree	str	27.76	29.92	**100.39**	**30.12**	**31.24**	101.12	37.33	**35.30**	**100.01**	17.43	24.01	99.50
tree	tree	23.75	27.69	103.05	25.69	28.81	102.08	31.66	32.20	102.08	13.81	21.74	105.46

Table 15: Translation results on the test set using additive length normalization and, where relevant, bag-level decoding.

Search achieves a rather low BLEU score by itself, but it is indeed this low scoring output that achieves the best model scores. Furthermore, Bag-Level Search seems to combine particularly well with length normalization, allowing it to find high-BLEU hypotheses *despite* its relatively fewer search errors.

After applying additive normalization for all systems, and combining it with bag-level search for our systems with syntactic output, we arrive at the final results in Tables 14 and 15.

QUALITATIVE ANALYSIS

In addition to the quantitative analysis of BLEU and METEOR scores, we find qualitatively that there are certain types of sentences on which string-based or tree-based systems over- or under-perform. To that end we perform a manual analysis of the output of our four systems on the German–English task.

First, we find that syntax-based systems produce far fewer UNK tokens than the baseline system. On our development set our baseline string-to-string system produced a total of 7871 UNKS, compared to 2072 for the string-to-tree, 2420 for the tree-to-string, and 1824 for the tree-to-tree system. A large part of this seems to be sequences of multiple UNKS, sometimes repeating a dozen or more times in the translation of a single sentence.

src	Man muss sie sich selbst erschaffen .
ref	You have to make it yourself .
s2s	You have to make it yourself .
s2t	You have to create them .
t2s	You 've got to make them yourself .
t2t	You 've got to make them out of themselves .
src	" Jetzt kannst du auf eine richtige Schule gehen , " sagte er .
ref	" You can go to a real school now , " he said .
s2s	UNK UNK UNK , you can go to a real school , UNK UNK , he said .
s2t	" Now , you can go to a real school . "
t2s	" Now you can go to a really high school , " he said . "
t2t	" Now you can go to a real school , " he said .
src	Stattdessen stehe ich heute hier , als stolze Absolventin des Middlebury College .
ref	Instead , I stand here a proud graduate of Middlebury College .
s2s	Instead , I stand here today , as proud UNK of the UNK College .
s2t	Instead , I am standing here , as a proud UNK of the UNK college .
t2s	So , instead of today , I stand here as a proud UNK of the UNK College .
t2t	Instead , I 'm standing here today as a proud student of college college .

Table 16: Examples of the phenomena identified in our qualtiative analysis. Syntax systems paraphrase more (top). Our string-to-string overproduces UNKs (middle). Our syntax-based decoder confuses continous aspect (bottom).

Second, we find that our tree-based decoder seems to confuse progressive aspect. It frequently uses the continuous aspect when the reference (and string-based decoder systems) do not, and chooses not to use it when the reference does. We hypothesize this is because of the tree-based decoder having more access to nearby adverbs as it builds the target sentence. Since German lacks continous aspect (Mair, 2012), the existence of adverbs such as *heute* ("today") or *jetzt* ("now") is used to disambiguate. The tree-based decoder seems to have learned to over-rely on this type of rule, generating the progressive almost exclusively in the presence of such adverbs.

Third, we find that our syntax-based systems in general have a tendency to paraphrase more than the string-to-string system, which is often more literal. This tendency explains the fact that the gap between our systems is much smaller according to METEOR (which is aware of paraphrasing and morphological variation) than according to BLEU (which relies soley on strict n-gram matching).

Examples of these phenomena on sentences from our German–English development set can be found in Table 16.

DISCUSSION

Comparing the length ratios in these two results tables we can see that the length normalization coefficients chosen to maximize BLEU on our dev sets do indeed generalize quite well, causing the length on the test set to be quite close to 100%.

We see that overall our tree-to-string systems perform about as well as our string-to-string baseline. Surprisingly source-side syntax seems to do best on German–English and French–English, language pairs that have less divergent syntax than Arabic–English or Chinese–English.

Target-side syntax is still underperforming string-to-string baselines, despite the new bag-level decoding algorithm providing gains of +3 BLEU or more. It seems that the underlying models are extremely biased towards very short hypotheses. In our experiments with bag-level decoding and no length normalization, we saw outputs with great model scores, but with very low length ratios and hence very low BLEU scores.

This observation underscores the perverse nature of NMT decoding. Traditional string-to-string models have been shown to favor very short hypotheses over ones prefered by humans or standard metrics (Stahlberg and Byrne, 2019). We observe a similar bias in our syntax-aware models. The difference, however, lies in how the models interact with the decoding algorithm.

String-to-string models are known to only produce good results with small beam sizes (Koehn and Knowles, 2017). Larger beam sizes continue to improve model score, yet BLEU scores nosedive as the search algorithm starts to explore the very short hypotheses to which the MT model assigns the best scores. One reason for this behavior is that the </s> token is unlikely to be in the top few preferred words until the model has output a more or less complete sentence. However since each word strictly decreases a sentence's model score, a full sentence followed by a reasonably likely </s> may well still have a worse model score than a 0-word hypothesis that generates the end of sentence symbol first thing (Stahlberg and Byrne, 2019). It is due to a happy irony that this conspiracy between a bad model and a bad inference algorithm is able to produce reasonable results.

Our syntax-output systems, on the other hand, do not share the benefits of this conspiracy. In our model the end of sentence symbol is not a separate action. Instead the model ends a sentence with a SHIFT like any other. This means that very short hypotheses (e.g. one NT followed by two SHIFTS) are likely to be visible to a standard beam search and indeed are guaranteed to be visible to the bag-level decoding algorithm proposed.

		10%			25%			50%			75%			100%		
		BLEU	MTR	Len	BLEU	MTR	Len	BLEU	MTR	Len	BLEU	MTR	Len	BLEU	MTR	Len
string	string	**7.51**	**13.84**	102.50	**12.05**	**18.70**	**101.05**	**15.27**	**22.00**	99.41	**15.06**	**21.66**	**100.13**	**18.05**	**24.49**	**100.05**
string	tree	6.05	12.61	104.63	9.46	16.62	103.21	12.26	19.75	102.66	13.20	21.00	104.60	14.48	22.04	103.01
tree	string	2.74	8.93	115.26	8.99	16.70	111.37	15.22	21.93	**100.35**	12.27	19.58	99.23	17.43	24.01	99.50
tree	tree	3.69	8.60	**98.15**	7.47	14.16	105.34	9.19	16.23	105.57	11.26	18.67	102.84	13.81	21.74	105.46

Table 17: Results of data ablation on our Chinese–English systems using length normalization and, where relevant, bag-level decoding. Systems were trained on the first {10, 25, 50, 75, 100}% of the training data. Length normalization strength was chosen to optimize dev set BLEU. Results shown are on the held out test set.

With these deficiencies in our models (be they syntax-based or not) and our decoding algorithms the art of decoding with neural models is in a very unstable state. Improvements to our search algorithms counter-intuitively hurt performance. A bad model with a bad decoding algorithm may outperform (according to BLEU, METEOR, or human evaluations) a better model with a better inference procedure.

DATA ABLATION

To further investigate our hypothesis that linguistic information can help sample efficiency we perform a data ablation on our Chinese–English translation systems. We run each of our four systems (string-to-string, string-to-tree, tree-to-string, and tree-to-tree) with varying amounts of training data, from just 10% up to 75% of the whole corpus. We then optimize the length normalization constant on the same dev set and test on the full test as the full systems. Table 17 shows the results.

We find that the string-to-string system still performs best overall across all data sizes and the tree-to-tree system performs worst. The two intermediate systems, however, exhibit some interesting behaviors. We see that the string-to-tree system outperforms the tree-to-string system with low data quantities, but the opposite as true at higher data sizes.

FUTURE DIRECTIONS

Our ultimate goal is to correct the model error inherent to the current generation of translation models such that advances in search techniques once again lead to improved translation quality.

There are several promising avenues toward this end. One is training models to maximize metric scores directly, e.g. with reinforcement learning (Shen et al., 2015; Wu et al.,

2016). Another method is to explicit incorporate a notion of coverage into the model (Tu et al., 2016; Chen et al., 2017a), allowing it to understand that it should translate all of the content in the source sentence and punishing it if it fails to do so.

Once we remove the effect of length bias from MT models we believe the power of RNNGs to capture the processes that underlie human sentence generation and judgements of grammaticality will be able to shine.

Finally, we leave to future work the integration of the syntax-based translation ideas contained in this chapter and the morphology-based advances made in Chapter 3. We predict that such a fully linstically-structured translation system would be very successful on many traditionally difficult language pairs. In addition its improved sample efficiency would improve the state of translation for low-resource languages, the overwhelming majority of which are far more morphologically complex than English.

RELATED WORK

Syntax in Symbolic MT Systems

Shortly after the first statistical translation systems saw success many authors began incorporating syntactic information into SMT. Alshawi et al. (2000) was among the first, using dependency transducers to turn a source-side depdency tree into a target language one. Yamada and Knight (2001) created a tree-to-tree system using phrase-structure trees. They use a multi-step approach wherein nodes in a source-side tree are reordered, then unaligned target words are allowed to be inserted into the tree structure, and finally the remaining terminals are translated using a dictionary.

Later Galley et al. (2004) introduced a string-to-tree system. They use a tree-based word alignments to extract rules and proposed a decoding algorithm wherein source-language substrings are iterative replace with target-language tree fragments. More recently Pust et al. (2015) modified the former approach to use AMR parses (Banarescu et al., 2013) rather than standard phrase structure trees.

The Hiero system of Chiang (2007) generalizes phrase-based translation by introducing the concept of hierarchical phrases. The result can be thought of as a tree-to-tree system using *unlabeled* phrase structure trees. Hanneman et al. (2011) introduce a full tree-to-tree system using phrase-structure trees by learning a probabilistic synchronous context free grammar directly from aligned, parsed, parallel sentences.

Syntax in Neural MT Systems

Perhaps the simplest method explored to shoehorn syntax into the existing encoder-decoder framework is to simply use a linearize a tree structure and then feed it into the system as if it were a string. Li et al. (2017) and (Aharoni and Goldberg, 2017) explore this approach on the source and target sides respectively. Nadejde et al. (2017b) employ a similar approach, introducing words' CCG tags as extra tokens in the source and target sentences.

Chen et al. (2018a) use source-side syntax trees to compute pair-wise distances between words and use this information to inform the attention model. They encourage the attention distribution at time step $t+1$ to be similar, according to this distance metric, to the attention distribution at time step t.

Several authors have introduces variants of source-side syntax into their NMT systems. Wu et al. (2017b) enrich the embeddings of each source word with head- and child-based information gleaned from dependency trees. Bastings et al. (2017) use graph convolutional networks to encode source-side dependency trees. Eriguchi et al. (2016) generalize LSTMs to admit tree structures rather than just linear sequences. They apply their TreeLSTMs to source-language phrase-structure trees and demonstrate improved translation performance. Similarly, Chen et al. (2017a) generalize GRUs to tree structures and run them through the source syntax tree in both top-down and bottom-up fashions, analogous to how BiLSTMs traverse terminals in both directions. They additionally introduce an attention mechanism that incorporates source-side syntax, as well as the notion of coverage, to nice effect.

Target-side syntax has proven more elusive, but Eriguchi et al. (2017) take the first steps in that direction. They show that by training a model to perform both translation and RNNG-based parsing they achieve superior translation models. While the underlying translation system is still string-to-string, they plug the output stream of the translation system into the buffer of the RNNG model, thus yielding a target-language tree. At inference time, however, they are free to drop the parsing task entirely and emit strings from their translation model directly. Wang et al. (2018) learn a CFG from training data and then allow a neural decoder to generate a parse tree using those CFG rules. While they experiment with phrase structure and dependency trees, they actually conclude that simply dividing the words into a balanced binary tree without using any linguistic knowledge actually outperforms using trees produced by parsers.

CONCLUSIONS

This chapter focuses on using syntax-infused neural networks to model *conditional* generation of sentences, a more practical and more difficult goal than the unconditional generation models described in Chapter 4. This task is particularly interesting because not only can we use syntax to govern generation, but we can also use it to modulate how sentences are represented in the source language and how the model draws connections between the two languages. Furthermore, it also follows up on historical work by testing whether the syntax-based improvements added into statistical models continue to have practical value in the age of neural networks.

We introduce a novel batching scheme capable of handling source-side syntax, making syntactic encoders much more palettable.

Our experiments here show mixed support for the thesis statement. On the one hand, we show that source-side syntax can improve translation performance for some langauge pairs. On the other, we find that the decoding problem syntax-based decoders is more difficult than that of tradition string-emitting decoders. We examine several simple methods to encourage adequacy and propose two novel decoding algorithms that outperform beam search by quite some margin, yet a gap remains between our baseline and target-side syntax systems. Since RNNGs are known to encode helpful biases for modelling language (Dyer et al., 2016) we conclude that the harmful biases of teacher forcing and maximum likelihood training offset any gains the extra linguistic knowledge provides.

6

CONCLUSION

SUMMARY OF CONTRIBUTIONS

In this thesis we have explored and incorporated several types of linguistic information into models (both conditional and unconditional) of human language.

First, we investigated a language model that uses knowledge of the morphological processes that turn morphemes, fundamental units of meaning, into words. Our model makes use of the knowledge encoded in morphological analyzers and constructs its output by combining word-, character-, and morpheme-level information. We demonstrated the effectiveness of this new language model on several morphologically rich languages for which traditional language modelling techniques struggle.

Second, we devised two different dependency-based language models. With the success of phrase-structure language models and the relatively large amount of human-annotated dependency data available we hoped to produce similarly strong language models extendable to more language pairs. Instead we discovered that while dependencies are a wonderful tool for analyzing sentences, they seem less able to provide models with the grammaticality judgement style of knowledge required to make great language models. Nonetheless we show that generative neural dependency models improve the performance of a strong discriminative baseline parser.

Third, we introduce phrase-structure syntax into neural machine translation systems on both the source and target sides. We present a new syntax-based decoder that outputs a tree encoded using RNNG actions rather than a linear sequence of words. We use this decoder, along with TreeGRUs, to create the first neural quasi-synchronous tree-to-tree translation system. We ablate this system, producing an empirical comparison of string-to-string, string-to-tree, tree-to-string, and tree-to-tree translation paradigms. In addition, we unveil a novel batching scheme for source-side syntax and propose two new decoding algorithms for tackling the challenging problem of inference with target-side syntax.

CONCLUSIONS

This thesis demonstrates that explicit knowledge of linguistic structure can be used to encode helpful biases to neural models. By imbuing neural language models with knowledge from human experts we can speed their learning and improve their ability to generalize in a matter similar to humans.

Knowledge of word formation morphology allowed us to create a fully open-vocabulary word model capable of elegantly understanding how to successfully conjugate verbs and decline nouns. This capablility even includes generating new forms of never-before-seen tokens such as personal names, a crucial ability in languages with case marking.

We demonstrated that although phrase structure trees improve the performance of unconditional language models (Dyer et al., 2016), the more plentiful dependency trees are unable to offer similar advances. Neural generative models of dependency syntax can, however, improve the accuracy of discriminative dependency parsers. This contrast demonstrates that phrase structure grammars better capture the notion of grammaticality and fluency in a language, while dependency models are better able to analyze the relationship between words. While the former may thus be preferable for language modelling, the latter may be preferred for downstream NLP tasks such as summarization (Barzilay and McKeown, 2005) and sentiment classification (Zou et al., 2015).

We also learned that construction order does not play a large role in the performance of our dependency-based language models. The top-down and bottom-up models presented in Chapter 4 encode very different structural biases, providing the model with very different conditioning contexts at each time step. Nonetheless, the models converge to very similar solutions, demonstrating the power of neural language models to capture arbitrary and complex corelations given sufficient training data.

We also highlight the complicated interaction between neural translation models and inference algorithms. Neural models assign dramatically better model scores to short (or even empty) hypotheses (Stahlberg and Byrne, 2019), a flaw only masked by greedy inference algorithms that commit egregious search errors. This interaction means that even if one improves both the translation model and the decoding algorithm the overall performance, according to standard translation metrics, may actually decrease. We observe this effect when we use RNNGs (Dyer et al., 2016), which have been shown to perform better than RNNs, as our target-language model and decode with an inference algorithm that commits fewer search errors. Nevertheless we observe decreased translation performance

across all four language pairs tested, regardless of whether we use syntax-enhanced encoders or not. This interaction proves that it is critical that we endevour to understand the biases that go into our neural models, be they explicit or implicit, hurtful or helpful.

FUTURE DIRECTIONS

Morphology in Translation

In this thesis we have demonstrated that morphological knowledge and syntactic knowledge can both improve the effectiveness of language models. The integration of both into one language model, be it conditional or unconditional is left to future work. Chahuneau et al. (2013a) showed the power of morphology in symbolic machine translation. We posit that integration of morphological knowledge into our machine translation model would similarly yield a neural system that excels on languages with very different word order than English as well as highly productive morphology such as Turkish or Japanese.

Eliminating Harmful Biases in NMT Models

We have identified a key deficiency in neural models' interaction with beam search that occurs to the models' harmful bias towards short sentences. While the two new decoding procedures introduced in this thesis offer improved decoding with RNNG-based language models, much work remains to stablize generation from neural models. In particular, we believe it will be necessary to depart from teacher forcing (Williams and Zipser, 1989) in favor of training schemes that allow a model more room for exploration.

Syntax-aware Attention Mechanisms

The attention model used in most translation system has no intrinsic notion of word order at all. The model is free to jump around the sentence arbitrarily, translating whatever bits and pieces it sees fit. As humans, however, we know *a priori* that some word orders are more likely than others, even without knowing which languages we're translating between.

For example, we know that if we just translated the nth source word, we are more likely to translate the $n+1$th source word, or perhaps the $n-1$th source word, then a word

distant in the sentence (Markovian prior) (Cohn et al., 2016). We probably think that the more times a word has been translated, the less likely it is to be chosen again in the future (Coverage prior) (Mi et al., 2016; Sankaran et al., 2016). We might suppose that a word that occurs early in the source sentence is more likely to occur early in the target sentence, and similar for words towards the middle or later in the source sentence (Diagonal prior).

We propose that syntactic constituents in the source sentence should be translated as a whole. It seems fine to translate the verb phrase and then the subject noun phrase, or the subject noun phrase and then the verb phrase, but it should be very unlikely to, say, translate half of the subject noun phrase, then the verb phrase, and then the rest of the subject noun phrase. We could encode this idea into an attention mechanism that takes into consideration the sequence of attention distributions and encourages the model to stick to translating the current constituent until completion thereof. While using phrase structure trees from a parser in the attention mechanism is yet untested, the ideas of using a Markovian-style prior in dependency space (Chen et al., 2018b) and the idea of using syntax as a latent variable to inform the attention mechanism (Bradbury and Socher, 2017) has shown the potential of syntax-aware models.

Linguistics without Human Knowledge

In this thesis we use syntactic parsers trained on human-annotated data and morphological analyzers hand-crafted by expert linguists. The creation of the tools may be prohibitively expensive, requiring many hours of labor by trained specialists. While these tools exist for many of the most widely spoken languages they do not exist for smaller languages. Furthermore, smaller languages usually lack large amounts of training data making the helpful biases and increased sample efficiency of linguistically aware models all the more important. As such it would be helpful to understand whether automated tools such as Morfessor (Smit et al., 2014) (for morphology), the system of Han et al. (2019) (for dependency trees), or the system of Maillard et al. (2019) (for phrase structure trees), could be used as drop-in replacements for the analyzers used in this thesis. While we might anticipate some drop in performance on large languages for discarding human knowledge, the ability to reap the benefits of syntax on smaller languages makes this effort a promising direction.

Document-level Language Modelling

In this thesis we injected word- and sentence-level linguitics into language models but humans' linguistic knowledge does not stop there. Recently work has improved the state of intersential discourse parsing (Jia et al., 2018; Morey et al., 2018), opening the door to document-level modelling to further improve the fluency and cohesion of the output of our language models. Document-level machine translation has been tried before (Zhang et al., 2018; Junczys-Dowmunt, 2019) with mixed success, though (to our knowledge) never using high-level parsers or human knowledge of discourse structure.

Other Sources of Human Knowledge

We have shown that it is possible to imbue neural models with biases based on human knowledge. In this thesis we focus on using morphology and syntax, well-studied areas of linguistics with clear applications to human language. There are, however, many other sources of human knowledge that can be introduced to language modelling and/or translation. For example, WordNet (Fellbaum, 2010) is a database of English words, their definitions, synonyms, senses, adn more. This knowledge could be used to speed language learning by encouraging e.g. synonyms to have similar word vectors. Furthermore, humans have invested much time and effort into creating bilingual dictionaries for language learners. While such dictionaries are imperfect they could be incorporated as priors into translation systems (e.g. using the method Arthur et al. (2016) apply to induced lexicons) to again propel learning. Naturally there are any number of other source of human knowledge that could be similarly integrated to improve sample complexity or naturalness of our models.

BIBLIOGRAPHY

1981. *Yeni Yazım Kılavuzu (New Writing Guide)*, 11 edition. Türk Dil Kurumu (Turkish Language Society), Ankara, Turkey.

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. *Proceedings of the Association for Computational Linguistics*.

Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. 2000. Learning dependency translation models as collections of finite-state head transducers. *Computational Linguistics*, 26(1):45–60.

Philip Arthur, Graham Neubig, and Satoshi Nakamura. 2016. Incorporating discrete translation lexicons into neural machine translation. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Machine Learning*.

Paul Baltescu, Phil Blunsom, and Hieu Hoang. 2014. Oxlm: A neural language modelling framework for machine translation. *The Prague Bulletin of Mathematical Linguistics*, 102(1).

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.

Regina Barzilay and Kathleen R McKeown. 2005. Sentence fusion for multidocument news summarization. *Computational Linguistics*, 31(3):297–328.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Julian Besag. 1975. Statistical analysis of non-lattice data. *The statistician*, pages 179–195.

Anders Björkelund, Agnieszka Falenska, Xiang Yu, and Jonas Kuhn. 2017. Ims at the conll 2017 ud shared task: Crfs and perceptrons meet neural networks. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51.

Jan A Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *ICML*, pages 1899–1907.

James Bradbury and Chunli Fu. 2018. Automatic batching as a compiler pass in pytorch.

James Bradbury and Richard Socher. 2017. Towards neural machine translation with latent tree attention. *Proceedings of the Association for Computational Linguistics*.

Eric Brill and Robert C Moore. 2000. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293. Association for Computational Linguistics.

Peter F Brown, Vincent J Della Pietra, Robert L Mercer, Stephen A Della Pietra, and Jennifer C Lai. 1992. An estimate of an upper bound for the entropy of english. *Computational Linguistics*, 18(1):31–40.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the tenth conference on computational natural language learning*, pages 149–164. Association for Computational Linguistics.

Jan Buys and Phil Blunsom. 2015. Generative incremental dependency parsing with neural 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 2: Short Papers)*, volume 2, pages 863–869.

Jan Buys and Phil Blunsom. 2018. Neural syntactic generative models with exact marginalization. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 942–952.

Victor Chahuneau, Eva Schlinger, Noah A Smith, and Chris Dyer. 2013a. Translating into morphologically rich languages with synthetic phrases.

Victor Chahuneau, Noah A Smith, and Chris Dyer. 2013b. Knowledge-rich morphological priors for bayesian language models. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017a. Improved neural machine translation with a syntax-aware encoder and decoder. *Proceedings of the Association for Computational Linguistics*.

Kehai Chen, Rui Wang, Masao Utiyama, Lemao Liu, Akihiro Tamura, Eiichiro Sumita, and Tiejun Zhao. 2017b. Neural machine translation with source dependency representation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2846–2852.

Kehai Chen, Rui Wang, Masao Utiyama, Eiichiro Sumita, and Tiejun Zhao. 2018a. Syntax-directed attention for neural machine translation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Kehai Chen, Rui Wang, Masao Utiyama, Eiichiro Sumita, and Tiejun Zhao. 2018b. Syntax-directed attention for neural machine translation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics.

David Chiang. 2007. Hierarchical phrase-based translation. *computational linguistics*, 33(2):201–228.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Noam Chomsky. 1993a. *Lectures on government and binding: The Pisa lectures*. 9. Walter de Gruyter.

Noam Chomsky. 1993b. A minimalist program for linguistic theory. *The view from Building 20: Essays in linguistics in honor of Sylvain Bromberger*.

Noam Chomsky. 2014. *Aspects of the Theory of Syntax*, volume 11. MIT press.

Noam Chomsky and David W Lightfoot. 2002. *Syntactic structures*. Walter de Gruyter.

Yoeng-Jin Chu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2017. Hierarchical multiscale recurrent neural networks. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. 2016. Incorporating structural alignment biases into an attentional neural translation model. *Proceedings of NAACL-HLT 2016*.

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. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT 2019*.

Timothy Dozat, Peng Qi, and Christopher D Manning. 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. ACL*.

Chris Dyer, Victor Chahuneau, and Noah A Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209.

Chris Dyer, Jonathan Weese, Hendra Setiawan, Adam Lopez, Ferhan Ture, Vladimir Eidelman, Juri Ganitkevitch, Phil Blunsom, and Philip Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In

Proceedings of the ACL 2010 System Demonstrations, pages 7–12. Association for Computational Linguistics.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71:233–240.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. *Proceedings of the Association for Computational Linguistics*.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. *Proceedings of the Association for Computational Linguistics*.

Manaal Faruqui, Yulia Tsvetkov, Graham Neubig, and Chris Dyer. 2016. Morphological inflection generation using character sequence to sequence learning. *Proceedings of the Association for Computational Linguistics*.

Christiane Fellbaum. 2010. Wordnet. In *Theory and applications of ontology: computer applications*, pages 231–243. Springer.

Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George E Dahl. 2018. The importance of generation order in language modeling. *The importance of generation order in language modeling*.

Daniel Fried, Mitchell Stern, and Dan Klein. 2017. Improving neural parsing by disentangling model combination and reranking effects. *Proceedings of the Association for Computational Linguistics*.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule. Technical report, COLUMBIA UNIV NEW YORK DEPT OF COMPUTER SCIENCE.

Kevin Gimpel and Noah A Smith. 2011. Quasi-synchronous phrase dependency grammars for machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 474–485. Association for Computational Linguistics.

Wenjuan Han, Yong Jiang, and Kewei Tu. 2019. Lexicalized neural unsupervised dependency parsing. *Neurocomputing*, 349:105–115.

Greg Hanneman, Michelle Burroughs, and Alon Lavie. 2011. A general-purpose rule extractor for scfg-based machine translation. In *Proceedings of the Fifth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 135–144. Association for Computational Linguistics.

Betty Hart and Todd R Risley. 2003. The early catastrophe. *Education Review (London)*, 17(1):110–118.

Martin Haspelmath and Andrea Sims. 2013. *Understanding morphology*. Routledge.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 95. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Kyuyeon Hwang and Wonyong Sung. 2017. Character-level language modeling with hierarchical recurrent neural networks. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Ray Jackendoff. 1977. X-bar syntax, a study of phrase structure. *Linguistic Inquiry Monograph No 2*.

Yanyan Jia, Yansong Feng, Yuan Ye, Chao Lv, Chongde Shi, and Dongyan Zhao. 2018. Improved discourse parsing with two-step neural transition-based model. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 17(2):11.

Yong Jiang, Wenjuan Han, and Kewei Tu. 2016. Unsupervised neural dependency parsing. In *Proc. EMNLP*.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Marcin Junczys-Dowmunt. 2019. Microsoft translator at wmt 2019: Towards large-scale document-level neural machine translation. *Proceedings of the Fourth Conference on Machine Translation (WMT)*.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bo-

goychev, et al. 2018. Marian: Fast neural machine translation in c++. *Proceedings of the Association for Computational Linguistics*.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709.

Hiroshi Kanayama, Masayasu Muraoka, and Katsumasa Yoshikawa. 2017. A semi-universal pipelined approach to the conll 2017 ud shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 265–273.

Moonyoung Kang, Tim Ng, and Long Nguyen. 2011. Mandarin word-character hybrid-input neural network language model. In *INTERSPEECH*, pages 625–628.

Katharina Kann, Sascha Rothe, and Katja Filippova. 2018. Sentence-level fluency evaluation: References help, but can be spared! *Proceedings of the 22nd Conference on Computational Natural Language Learning (CoNLL 2018)*.

Ronald M Kaplan, Joan Bresnan, et al. 1982. Lexical-functional grammar: A formal system for grammatical representation. *Formal Issues in Lexical-Functional Grammar*, 47:29–130.

Kazuya Kawakami, Chris Dyer, and Phil Blunsom. 2017. Learning to create and reuse words in open-vocabulary neural language modeling. *Proceedings of the Association for Computational Linguistics*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2741–2749.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations (ICLR 2015)*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE.

Philipp Koehn and Hieu Hoang. 2007. Factored translation models. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 868–876.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. *Proceedings of the First Workshop on Neural Machine Translation*.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics.

Mikhail Korobov. 2015. Morphological analyzer and generator for russian and ukrainian languages. In *International Conference on Analysis of Images, Social Networks and Texts*, pages 320–332. Springer.

Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1048–1059, Jeju Island, South Korea.

Jonathan K. Kummerfeld, Daniel Tse, James R. Curran, and Dan Klein. 2013. An empirical examination of challenges in chinese parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 98–103, Sofia, Bulgaria.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A Smith. 2016. What do recurrent neural network grammars learn about syntax? *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. Lstms can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436.

Peter CR Lane and James B Henderson. 1998. Simple synchrony networks: Learning to parse natural language with temporal synchrony variable binding. In *ICANN 98*, pages 615–620. Springer.

Alon Lavie, Alok Parlikar, and Vamshi Ambati. 2008. Syntax-driven learning of subsentential translation equivalents and translation rules from parsed parallel corpora. In *Proceedings of the Second Workshop on Syntax and Structure in Statistical Translation*, pages 87–95. Association for Computational Linguistics.

Junhui Li, Deyi Xiong, Zhaopeng Tu, Muhua Zhu, Min Zhang, and Guodong Zhou. 2017. Modeling source syntax for neural machine translation. *arXiv preprint arXiv:1705.01020*.

Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. 2016. Latent predictor networks for code generation. *Proceedings of the Association for Computational Linguistics*.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*.

Nelson F Liu, Jonathan May, Michael Pust, and Kevin Knight. 2018. Augmenting statistical machine translation with subword translation of out-of-vocabulary words. *arXiv preprint arXiv:1808.05700*.

Minh-Thang Luong and Christopher D Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. *Proceedings of the Association for Computational Linguistics*.

Amandla Mabona, Laura Rimell, Stephen Clark, and Andreas Vlachos. 2019. Neural generative rhetorical structure parsing. *arXiv preprint arXiv:1909.11049*.

Jean Maillard, Stephen Clark, and Dani Yogatama. 2019. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *Natural Language Engineering*, 25(4):433–449.

Christian Mair. 2012. Progressive and continuous aspect. *The Oxford handbook of tense and aspect*, pages 803–827.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Andrey Andreyevich Markov. 1906. Extension of the law of large numbers to dependent quantities. *Izv. Fiz.-Matem. Obsch. Kazan Univ.(2nd Ser)*, 15:135–156.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. *Proceedings of the International Conference on Learning Representations (ICLR 2017)*.

Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. 2016. Coverage embedding models for neural machine translation. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, and Stefan Kombrink. 2012. Subword language modeling with neural networks.

Yasumasa Miyamoto and Kyunghyun Cho. 2016. Gated word-character recurrent language model. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*.

Mathieu Morey, Philippe Muller, and Nicholas Asher. 2018. A dependency perspective on rst discourse parsing and evaluation. *Computational Linguistics*, 44(2):197–235.

Kenton Murray and David Chiang. 2018. Correcting length bias in neural machine translation. *Proceedings of the Third Conference on Machine Translation (WMT)*.

Maria Nadejde, Siva Reddy, Rico Sennrich, Tomasz Dwojak, Marcin Junczys-Dowmunt, Philipp Koehn, and Alexandra Birch. 2017a. Predicting target language ccg supertags improves neural machine translation. In *Proceedings of the Second Conference on Machine Translation*, pages 68–79.

Maria Nadejde, Siva Reddy, Rico Sennrich, Tomasz Dwojak, Marcin Junczys-Dowmunt, Philipp Koehn, and Alexandra Birch. 2017b. Syntax-aware neural machine translation using ccg. *arXiv preprint arXiv:1702.01147*.

Radford M Neal. 1992. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113.

Graham Neubig. 2016. Lexicons and minimum risk training for neural machine translation: Naist-cmu at wat2016. *Proceedings of the 3rd Workshop on Asian Translation*.

Graham Neubig and Kevin Duh. 2013. How much is said in a tweet? a multilingual, information-theoretic perspective.

Graham Neubig and Chris Dyer. 2016. Generalizing and hybridizing count-based and neural language models. *Proceedings of 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. 2017a. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Graham Neubig, Yoav Goldberg, and Chris Dyer. 2017b. On-the-fly operation batching in dynamic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3971–3981.

Graham Neubig, Matthias Sperber, Xinyi Wang, Matthieu Felix, Austin Matthews, Sarguna Padmanabhan, Ye Qi, Devendra Singh Sachan, Philip Arthur, Pierre Godard, et al. 2018. Xnmt: The extensible neural machine translation toolkit. *Proceedings of AMTA 2018*.

Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer.

Joakim Nivre. 2008. Sorting out dependency parsing. In *International Conference on Natural Language Processing*, pages 16–27. Springer.

Joakim Nivre. 2013. Transition-based parsing.

Joakim Nivre, Lars Ahrenberg Željko Agic, et al. 2017. Universal dependencies 2.0. lindat/clarin digital library at the institute of formal and applied linguistics, charles university, prague.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167. Association for Computational Linguistics.

Kemal Oflazer. 1994. Two-level description of turkish morphology. *Literary and linguistic computing*, 9(2):137–148.

Kyoung-Su Oh and Keechul Jung. 2004. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Kaspar Papli. 2017. Morpheme-aware subword segmentation for neural machine translation.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *Proceedings of NAACL-HLT 2018*.

Tommi A Pirinen. 2015. Development and use of computational morphology of finnish in the open source and open science era: Notes on experiences with omorfi development. *SKY Journal of Linguistics*, 28.

Carl Pollard and Ivan A Sag. 1994. *Head-driven phrase structure grammar*. University of Chicago Press.

Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. 2015. Using syntax-based machine translation to parse english into abstract meaning representation. *Proceedings of 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighted finite state transducers with neural context. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Baskaran Sankaran, Haitao Mi, Yaser Al-Onaizan, and Abe Ittycheriah. 2016. Temporal attention model for neural machine translation. *arXiv preprint arXiv:1608.02927*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *Proceedings of Association for Computational Linguistics*.

Qinlan Shen, Daniel Clothiaux, Emily Tagtow, Patrick Littell, and Chris Dyer. 2016. The role of context in neural morphological disambiguation.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2015. Minimum risk training for neural machine translation. *Proceedings of the Association for Computational Linguistics*.

Peter Smit, Sami Virpioja, Stig-Arne Grönroos, and Mikko Kurimo. 2014. Morfessor 2.0: Toolkit for statistical morphological segmentation. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 21–24.

David A Smith and Jason Eisner. 2006. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 23–30. Association for Computational Linguistics.

Felix Stahlberg and Bill Byrne. 2019. On nmt search errors and model errors: Cat got your tongue? *arXiv preprint arXiv:1908.10090*.

Mark Steedman. 1987. Combinatory grammars and parasitic gaps. *Natural Language & Linguistic Theory*, 5(3):403–439.

Milan Straka, Jan Hajic, Jana Straková, and Jan Hajic Jr. 2015. Parsing universal dependency treebanks using neural networks and search-based oracle. In *International Workshop on Treebanks and Linguistic Theories (TLT14)*, pages 208–220.

Milan Straka and Jana Straková. 2017. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.

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)*, pages 1017–1024.

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.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *Proceedings of the Association for Computational Linguistics*.

Lucien Tesnière. 1959. Eléments de linguistique structurale. *Paris, Klincksieck*, 2.

Ivan Titov and James Henderson. 2010. A latent variable model for generative dependency parsing. In *Trends in Parsing Technology*, pages 35–55. Springer.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. *Proceedings of the Association for Computational Linguistics*.

Henk C Van Riemsdijk and Edwin Williams. 1986. *Introduction to the Theory of Grammar*, volume 12. The MIT Press.

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.

David Vilar, Jan-T Peter, and Hermann Ney. 2007. Can we translate letters? In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 33–39. Association for Computational Linguistics.

Xinyi Wang, Hieu Pham, Pengcheng Yin, and Graham Neubig. 2018. A tree-based decoder for neural machine translation. *arXiv preprint arXiv:1808.09374*.

Warren Weaver. 1955. Translation. *Machine translation of languages*, 14:15–23.

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

Shuangzhi Wu, Dongdong Zhang, Nan Yang, Mu Li, and Ming Zhou. 2017a. Sequence-to-dependency neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 698–707.

Shuangzhi Wu, Ming Zhou, and Dongdong Zhang. 2017b. Improved neural machine translation with source syntax. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 2017). Melbourne, Australia*, pages 4179–4185.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 523–530.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Mehmet Ali Yatbaz and Deniz Yuret. 2009. Unsupervised morphological disambiguation using statistical language models. In *Pro. of the NIPS 2009 Workshop on Grammar Induction, Representation of Language and Language Learning, Whistler, Canada*, pages 321–324.

Dani Yogatama, Chris Dyer, Wang Ling, and Phil Blunsom. 2017. Generative and discriminative text classification with recurrent neural networks. *arXiv preprint arXiv:1703.01898*.

Lei Yu, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Tomas Kocisky. 2016. The neural noisy channel. *arXiv preprint arXiv:1611.02554*.

Deniz Yuret and Ferhan Türe. 2006. Learning morphological disambiguation rules for turkish. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 328–334. Association for Computational Linguistics.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, et al. 2017. Conll 2017 shared task: multilingual parsing from raw text to universal dependencies. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19.

Jiacheng Zhang, Huanbo Luan, Maosong Sun, Feifei Zhai, Jingfang Xu, Min Zhang, and Yang Liu. 2018. Improving the transformer translation model with document-level context. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2015. Top-down tree long short-term memory networks. In *Proc. NAACL*.

Huang Zou, Xinhua Tang, Bin Xie, and Bing Liu. 2015. Sentiment classification using machine learning techniques with syntax features. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 175–179. IEEE.