Towards Efficient Natural Language Generation

Junxian He

CMU-LTI-22-012

Apr 25, 2022

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

Thesis Committee:

Graham Neubig (co-Chair)	Carnegie Mellon University
Taylor Berg-Kirkpatrick (co-Chair)	University of California, San Diego
Emma Strubell	Carnegie Mellon University
Alexander Rush	Cornell University

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

Keywords: natural language generation, efficiency, generative models

To my beloved family, and those who helped me reach this stage.

Abstract

Natural language generation (NLG) has seen remarkable success benefiting from the development of deep learning techniques. As large-scale pretraining becomes the de-facto standard in NLP, enormous training data and model parameters consistently lead to state-of-the-art performance on standard NLG tasks. While quite successful, current NLG approaches are inefficient from several aspects, which prohibits their usage in broader and practical settings: (1) they are label-inefficient – conditional neural generation (e.g. machine translation) often requires a large number of annotated samples to train, which limits their applications in low-resource regimes; (2) they are parameter-inefficient – it is common practice to fine-tune a pretrained model to adapt it to the downstream task, however, these models could scale to trillions of parameters (Fedus et al., 2021), which would cause a large memory footprint when serving a large number of tasks; and (3) lastly, we focus on the compute-inefficiency of a trending model class, retrieval-augmented NLG models. They retrieve from an external datastore to assist in generation, the added datastore and retrieval process incurs non-trivial space and time cost due to extra computation.

In this thesis, we aim to provide a deeper understanding of research problems in efficient NLG and utilizing the insights to design better approaches. Specifically, (1) for label-efficiency we study unsupervised and semi-supervised conditional generation that take advantage of the abundant unlabeled text data, and thus mitigate the requirement of numerous annotated samples. The proposed methods are validated on a wide variety of NLG tasks; (2) for parameter-efficiency we propose a unified framework to connect parameter-efficient transfer learning, where only few parameters need to be updated to adapt a large pretrained model to downstream tasks. Our framework provides a new understanding of this direction, as well as instantiating state-of-the-art approaches for parameter-efficient NLG; (3) for compute-efficiency in retrieval-augmented NLG we either design new models or post-adapt the retrieval component to compress the datastore, reduce the retrieval compute, and speed up the inference.

Acknowledgements

In August 2017, my flight landed in Pittsburgh and my PhD journey begins. Today in 2022, I am writing my PhD thesis – five years passed in a flash. Now I have grown to be an independent researcher ready to continue my research career in the future, and I feel very grateful for the help I received along the journey.

This dissertation would not be possible without the tremendous help from my amazing advisors Taylor Berg-Kirkpatrick and Graham Neubig. I am grateful to have two advisors who are both encouraging and passionate about research. We met every week to discuss research jointly, most of my research ideas and works originate from those meetings. I am particularly thankful that they never put pressure on me and allowed me to take a break whenever I want. There were certainly many ups and downs during my PhD, and they cared a lot about whether I was happy or not in addition to my research progress. Their support has been with me through the difficult times, and enabled me to have a happy and satisfying research journey in the past five years. They established the perfect example of mentorship in my mind and I wish I could become a mentor like them in the future. I would also like to thank my other committee members Emma Strubell and Alexander Rush for providing insightful advice on my thesis and future career.

I am grateful for the companionship and support I received from members of NeuLab and BergLab. : Chunting Zhou, Pengcheng Yin, Cindy Wang, Pengfei Liu, Frank Xu, Hao Zhu, Aditi Chaudhary, Lucio Dery, Patrick Fernandes, Zhengbao Jiang, Shuyan Zhou, Kayo Yin, Shruti Rijhwani, Danish Pruthi, Hiroaki Hayashi, John Wieting, Junjie Hu, Paul Michel, Zecong Hu, Mengzhou Xia, Uri Alon, Maria Ryskina, Daniel Spokoyny, Volkan Cirik, Harsh Jhamtani, Kartik Goyal, Nikita Srivatsan, Fatemehsadat Mireshghallah, and Nikolai Vogler. On many Friday afternoons in the first two years, Chunting, Pengcheng, Cindy, and I gathered together, discussed research or chatted randomly, and then we went to explore new Pittsburgh restaurants for dinner; Chunting has been my particularly close friend and collaborator, we discussed research and explored many nice restaurants in Manhattan during our internship in Facebook New York in 2019. The works in Chapter 5 and Chapter 6 of this thesis are from our collaboration as well in the last year. I also spent a good time with her cats. Cindy and I were the same age and came to CMU in the same year, and thus we share experiences, anxieties, and happiness throughout our PhD. Cindy and I co-led the work in Chapter 3 of this thesis. She was able to progress research steadily day after day, and in the meanwhile maintained a very healthy working schedule in the past five years, I learned a lot from her. Pengcheng gave me very helpful advice on course selection, conducting research, paper writing, and Pittsburgh life in my first year; Pengfei is one of the most visionary

young researchers that I have ever met, he helped me form my long-term research visions and advised on how to mentor junior students. Maria was my office mate and accompanied me through the difficult times. I also had great pleasure hanging out and having Friday dinners with Shuyan, Frank, Zhengbao, and Hao. Finally, I want to thank my other friends in and outside LTI for research discussions, collaborations, and break gatherings: Bohan Li, Zhiting Hu, Hanqing Lu, Zhisong Zhang, and Xuezhe Ma.

During my PhD, I have had two wonderful summer internships in Facebook and Salesforce respectively. I am grateful to my awesome mentors: Jiatao Gu, Marc'Aurelio Ranzato, Wojciech Kryscinski, and Bryan McCann. Jiatao is one of the most diligent and dedicated researchers that I met in the past five years, which motivated me to work hard that summer and publish the self-training project presented in Chapter 4 of this thesis. Marc'Aurelio was persistently pushing me to understand self-training, which directly led to the major part of Chapter 4 explaining the results with noise and proposing noisy self-training. At Salesforce, Bryan and Wojciech helped my internship work on controllable summarization. My knowledge of summarization is from this internship experience.

I would also like to thank my old friends who may not pursue a research career but gave me comfort and support that meant a lot to me in the difficult times: Na Liu, Hanxiao He, Chuan Yang, Lihao Yi, Liangzhou Wang, and Xinjie Zhou.

My parents, Jianguo Wang, Chengxiang He, my grandparents, Minjun Ren, Bangqing He, and my stepfather Jian Qiao raise and support me over the years with your unconditional love, I want to thank you all.

Finally, I am deeply grateful to my mom, Jianguo Wang and my wife, Wanzhen He. My mom was fighting cancer since 2017, the year when I left home and started PhD in CMU. She passed away in 2021. She was always encouraging me, and remained strong and optimistic throughout the way. Her attitude in front of cancer inspired me to be brave and optimistic for the life ahead. My wife Wanzhen gave me tremendous support and care, inspired me with her work and lifestyles, and went through our long-distance relationship. This thesis is, undoubtedly, dedicated to you two.

Contents

1 Introduction **1**

1.1 Thesis Overview 3

2 Background and Literature Review **7**

2.1 Neural Text Generation 7

2.2 Efficiencies in NLG 10

2.3 Connection of Different Efficiencies 18

I Label-Efficient Natural Language Generation **21**

3 Unsupervised Sequence to Sequence Transduction **23**

3.1 Introduction 23

3.2 Unsupervised Text Style Transfer 25

3.3 The Deep Latent Sequence Model 26

3.4 Connection to Related Work 30

3.5 Experiments 31

3.6 Summary 39

4 Semi-Supervised NLG with Self-Training **41**

4.1 Introduction 41

4.2 Self-training 43

4.3 A Case Study on Machine Translation 44

4.4 Noise in Self-Training 48

4.5 Experiments 51

4.6 Related Work 58

4.7 Summary 58

5 Prompt consistency for Zero-Shot Task Generalization **61**
5.1 Introduction . . . 61
5.2 Prompt-based Zero-Shot Task Generalization . . . 64
5.3 Prompt Consistency Training . . . 65
5.4 Experiments . . . 69
5.5 Summary . . . 78

II Parameter-Efficient Natural Language Generation **79**

6 A Unified View of Parameter-Efficient Transfer Learning **81**
6.1 Introduction . . . 81
6.2 Preliminaries . . . 83
6.3 Bridging the Gap – A Unified View . . . 85
6.4 Experiments . . . 89
6.5 Summary . . . 97

III Efficient Retrieval-Augmented Natural Language Generation **99**

7 Learning Sparse Prototypes for Text Generation **101**
7.1 Introduction . . . 101
7.2 Background . . . 104
7.3 Method . . . 104
7.4 Experiments . . . 109
7.5 Summary . . . 115

8 Efficient Nearest Neighbor Language Models **117**
8.1 Introduction . . . 117
8.2 Nearest Neighbors Language Model . . . 119
8.3 The Efficiency of NNLM . . . 122
8.4 The Remedies . . . 124
8.5 Putting it All Together . . . 132
8.6 Summary . . . 133

9 Conclusions and Future Directions **135**

Appendix of Chapter 3 **141**
.1 Sentiment Transfer Example Outputs 141
.2 Repetitive Examples of BT+NLL 141

Appendix of Chapter 6 **145**
.3 Full Results on Different Bottleneck Dimensions 145

Appendix of Chapter 7 **147**
.1 Derivations of Variational Inference and ELBO 147
.2 Experimental Details 149
.3 Qualitative Results on Interpolation 150

Bibliography **153**

Chapter 1

Introduction

Language is the main medium of human communication. In artificial intelligence, language is one of the main interfaces of machines to communicate with humans, and thus machines need to be able to understand and generate natural language. In this thesis, we focus on the latter, natural language generation. Natural language generation is one of the most fundamental categories of tasks in NLP, spanning across machine translation (Bahdanau et al., 2015), text summarization (Rush et al., 2015), dialogue generation (Sordoni et al., 2015a), data description (Novikova et al., 2017), etc. With the rapid development of deep learning for NLP these years (Hochreiter and Schmidhuber, 1997; Bahdanau et al., 2015; Vaswani et al., 2017), we have witnessed tremendous progress on these tasks. In particular, large-scale self-supervised pretraining (Peters et al., 2018; Devlin et al., 2019a; Yang et al., 2019) has taken the performance of NLG tasks to a new level (Lewis et al., 2020a; Raffel et al., 2020). More recently, ever-large pretrained language models have shown the potential to tackle all NLP tasks as generation tasks, achieving competitive zero- or few-shot results given appropriate textual prompts (Radford et al., 2019; Brown et al., 2020; Schick and Schütze, 2021c; Du et al., 2021; Liu et al., 2021a; Sanh et al., 2022). Despite the great success achieved, current NLG approaches are *inefficient* in a number of ways, which prohibit their usage in broader settings. In this thesis, we consider the following three aspects of inefficiency.

Label inefficiency: the state-of-the-art natural language generation models are usually deep encoder-decoder or decoder-only neural networks, often powered by the self-attentional transformer architecture (Vaswani et al., 2017). These models are trained on parallel examples with cross-entropy loss in an end-to-end manner. The model training requires a large number of annotated examples to reach reasonable performance. For example, translation systems are often trained with millions of sentence pairs to reach practical performance (Akhbardeh et al.,

2021); popular text summarization benchmarks also consist of hundreds of thousands parallel examples (Hermann et al., 2015; Narayan et al., 2018). However, labeled examples are often scarce resources – abundant annotations only exist for certain domains. Moreover, most datasets nowadays are English-centric while there are over 7000 languages in the world, which means task labels are not easily available for most of the languages. This puts challenges to applying generally-applicable deep NLG models.

Parameter inefficiency: self-supervised pretraining techniques have demonstrated tremendous success on a wide variety of NLP tasks (Peters et al., 2018; Devlin et al., 2019a; Liu et al., 2019a; Yang et al., 2019). Generally, the model is first pretrained on raw text only with self-supervised losses, then the pretrained model is fine-tuned on downstream tasks with labeled data. Such a pipeline has become the de-facto standard nowadays to create state-of-the-art NLG systems. In this direction, researchers are pursuing more and more powerful pretrained models, which in fact lead to more parameters in most cases – ever-larger language models consist of hundreds of millions to trillions of parameters (Brown et al., 2020; Fedus et al., 2021; Rae et al., 2021). This way, each separate fine-tuning process obtains a different copy of the giant model, causing inefficient parameter utilization at both fine-tuning and inference time. Such parameter inefficiency would incur a significant memory footprint when serving numerous tasks.

Compute inefficiency in retrieval-augmented models: NLG tasks often require certain notion of knowledge from the model to make the correct prediction. For example, competitive machine translation systems need to know basic lexicon or ngram mappings between the two languages, or certain terminology translations in some domains; Question answering or unconditional language generation needs to generate the correct facts and avoid factual hallucination, for example, the model is required to correctly predict the answer for "`Who is the President of the United States?`" Fully parametric neural networks learn such knowledge and store them into model parameters, which is not only parameter-intensive but also challenging (Petroni et al., 2019). Recently, retrieval-augmented generation models are proposed to greatly boost performance on a variety of tasks without extra training through retrieving from an extra datastore (Khandelwal et al., 2020, 2021; Izacard and Grave, 2021). This may even be achieved with over 10$\times$ fewer model parameters (Borgeaud et al., 2021). However, the datastore and retrieval mechanism inevitably incur non-trivial overhead on space and time cost due to extra compute from the retrieval process. For example, nearest neighbor machine translation (Khandelwal et al., 2021) yields datastores of size over 1000GB and slow down the generation by two orders of magnitude. Such compute

inefficiency greatly limits practical deployment of retrieval-augmented models.

1.1 Thesis Overview

In this thesis, we put forward a series of methods to improve the efficiency of natural language generation, so that better NLG systems could be created without significantly increasing the resource requirement. Specifically, we first describe how we utilize unlabeled examples to help improve unsupervised or semi-supervised text generation (Part I), then we propose a unified framework for parameter-efficient transfer learning (PETL) and an accompanied state-of-the-art PETL method (Part II). PETL approaches aim to fine-tune a small portion of parameters of a frozen large model to achieve comparable performance to full fine-tuning, thus improving parameter-efficiency. Lastly, we focus on lightening retrieval-augmented methods in terms of both space and time, through reducing the datastore sizes and speeding up the retrieval process (Part III). A detailed overview of each part is as follows:

Part I Label-Efficient Natural Language Generation: while label efficiency could be improved from multiple perspectives as detailed more in Chapter 2, this thesis mainly focuses on utilizing unlabeled examples to improve performance with a limited number of annotations. This is practical since unlabeled samples are often available and could be easily collected online. Specifically, we first study a completely unsupervised setting where we try to learn from unpaired text data alone to perform sequence to sequence generation. We propose a probabilistic latent sequence model to perform unsupervised sequence-to-sequence transduction (Chapter 3). Our model provides a probabilistic view to unify previous practices such as back-translation and adversarial training, and achieves the state-of-the-art results in several text style transfer benchmarks. Next, we move to a more realistic setting where a few annotated examples and abundant unannotated examples are available in a semi-supervised manner. Semi-supervised learning aims to utilize those unlabeled examples to improve the vanilla supervised baseline systems. Our work explores the effect of self-training (Scudder, 1965) in modern neural sequence generation tasks (Chapter 4). Self-training trains a model with pseudo labels predicted by the model itself on unlabeled inputs. We found that self-training is very effective on neural machine translation and summarization tasks. Notably, we identify that the implicitly added noise, dropout, plays an important role in the success of self-training as a form of consistency regularization, thus we propose noisy self-training that injects noise into inputs as well. Noisy self-training significantly boosts the performance of vanilla supervised systems. Next, we extend the consistency regularization

of self-training from example-level to prompt-level in a more appealing setting, zero-shot task generalization (Chapter 5). Different from traditional dataset or domain generalization, zero-shot task generalization performs tasks that are never seen during model training. This is made possible by advances in prompt approaches (Liu et al., 2021a) that break the boundary of different tasks, and all NLP tasks are unified into the same sequence generation format. Specifically, we propose a novel swarm distillation loss, which distills the predictions of different prompts to each other to optimize prompt consistency. Our approach successfully utilizes unlabeled examples and outperforms the vanilla baseline on 9 out of 11 NLP datasets.

Part II Parameter-Efficient Natural Language Generation: This thesis focuses on parameter-efficient transfer learning (PETL) to improve parameter efficiency. Generally, parameter-efficient tuning updates a small number of (extra) parameters of a big frozen, pretrained model and pursues a large portion of parameter sharing among multiple tasks. This way, each task only needs to store a small task-specific module. Parameter-efficient tuning is demonstrated to be able to match classic, full fine-tuning results. This is possible because of pretraining techniques (Peters et al., 2018; Devlin et al., 2019a) thus PETL approaches. One pioneer work of this line is adapters (Houlsby et al., 2019) that insert a low-rank module inside a transformer model (Vaswani et al., 2017), another representative work includes prefix tuning (Li and Liang, 2021a) and prompt tuning (Lester et al., 2021). We propose a unified framework for previous state-of-the-art parameter-efficient tuning methods to understand their connections and important ingredients for their success (Chapter 6). Remarkably, we show that several PETL methods including prefix tuning could be viewed as a form of adapters, and different methods could be instantiated by varying design choices of the adapter, along our defined design dimensions. Through this unified framework, we are able to connect existing methods, identify their superior or inferior designs, and summarize the best set of design choices to derive a new state-of-the-art adapter variant. Our proposed approach greatly outperforms previous work by tuning fewer parameters.

Part III Efficient Retrieval-Augmented Natural Language Generation: This thesis focuses on improving the compute efficiency of retrieval-augmented generation models. Specifically, in Chapter 7, we study the latent prototype language model, in which text is generated from a prototype that is sampled from the training data. This sampling step was following a uniform distribution (Guu et al., 2018), thus the entire training dataset needs to be maintained and retrieved from at test time, even though much information in the training dataset is redundant. Instead, we propose to sampling prototypes from a latent, sparse distribution on the training examples,

and sparsity is encouraged through a Dirichlet prior. This way, our method learns the sparse distribution to remove redundancy, so that only salient training examples need to be saved and retrieved from during inference. Our approach is able to reduce the datastore size by 1000x with better performance.

In Chapter 8 we study another class of semi-parametric language model, the nearest neighbor language model (Khandelwal et al., 2020), which retrieves the nearest contexts from a corpus (e.g. the training corpus) to help prediction at test time. Such a retrieval process explicitly utilizes the knowledge present in the corpus instead of compacting them into model parameters. Our work proposes and explores three techniques, adaptive retrieval, datastore pruning, and dimension reduction to reduce the datastore size and speed up this process by 6.6x.

To summarize, this thesis focuses on developing efficient and practical natural language generation techniques, from the perspectives of label-efficiency, parameter-efficiency, and compute-efficiency. Before diving into the details of each work, we start with a survey on related work in the next chapter.

Chapter 2

Background and Literature Review

In this chapter, we first briefly review neural text generation in general. Then we emphasize related work to this thesis, specifically on efficient NLG from the perspectives of label-efficiency (§ 2.2.1), parameter-efficiency (§ 2.2.2), and compute-efficiency in retrieval augmented models (§ 2.2.3). In this survey, we also try to draw connections between related work and contributions in this thesis.

2.1 Neural Text Generation

Text generation refers to any task that can be formalized as generating texts as the output, such as generating translations, summaries, or dialogue responses. Language generation is a difficult task since it involves deciding what content to generate, how to organize the content, and even requires understanding abilities. Traditionally, NLG systems are complicated and consist of several stages explicitly: content determination to decide the information in the text, document structuring to organize the information, aggregation to merge similar sentences, lexical choice to put words, and realization to create the actual text (Reiter and Dale, 1997). With the prosperity of deep learning, neural text generation that uses deep neural networks as the backbone dominates the advances in this direction. Importantly, neural text generation does not need multiple stages of generation as a pipeline, but instead models the generation process as a black box implicitly – neural text generation is able to produce the text output in an end-to-end manner. Today, text generation has a broader role in natural language generation given that all NLP tasks may be formatted as a text generation task through prompts (Brown et al., 2020; Liu et al., 2021a). For example, a review sentiment classification task may be reformatted as generating text responses to question `Is the review good or bad?` Techniques proposed in this thesis emphasize *sequence*

generation models in general, and also solve traditional classification problems by treating them as text generation tasks, for example in Chapter 5. Below, we cover representative applications of neural text generation as well as the standard approaches.

2.1.1 Applications

Text generation applications can be classified in terms of their inputs. We follow Li et al. (2021a) to summarize text generation applications into the following categories.

- **Unconditional text generation**, which corresponds to no inputs (or random noise). Language models that predict the next token given context belong to this category, and serve as the most fundamental building blocks of neural text generation. Example benchmarks include Marcus et al. (1993); Chelba et al. (2013); Merity et al. (2017).
- **Attribute-to-text generation**, which corresponds to discrete attributes as the inputs, such as topic or sentiment labels. For example, CTRL (Keskar et al., 2019) is a general framework that is able to generate text from multiple different attributes.
- **Data-to-text generation**, which corresponds to structured data as the inputs, such as tabular data or knowledge graphs. Existing benchmarks cover facts description (Gardent et al., 2017), restaurant introduction (Novikova et al., 2017), basketball game stats summarization (Wiseman et al., 2017), and Wikipedia table description (Parikh et al., 2020).
- **Multimedia-to-text generation**, which corresponds to multimedia inputs. Example tasks include image caption (Chen et al., 2015), video caption (Zhou et al., 2018), or speech recognition (Panayotov et al., 2015).
- **Text-to-text generation**, which corresponds to text sequences as the inputs. This is probably the most common application that we see today, which consists of machine translation (Bahdanau et al., 2015), text summarization (Rush et al., 2015), chatbots (Adiwardana et al., 2020), etc.

In this thesis, we mainly perform experiments on language models and text-to-text generation.

2.1.2 Standard Approaches

Denote the input as $\mathbf{x}$ and the output text as $\mathbf{y}$, neural text generation models $p_\theta(\mathbf{y}|\mathbf{x})$, where θ is the parameters to be learned through gradient descent.

Training and Losses: Most neural text generation models are trained via backpropagation to maximize the log-likelihood $\log p_\theta(\mathbf{y}|\mathbf{x})$, where network architectures are the main variations between different methods. Other alternatives include energy-based models to train with an energy loss (Hinton, 2002; LeCun et al., 2006; Rosenfeld et al., 2001), latent variable models that add random variables into the model which can be further categorized into variational autoencoders (Kingma and Welling, 2014; Bowman et al., 2016) and generative adversarial networks (Goodfellow et al., 2014; Yu et al., 2017), or optimizing parameters to maximize a reward function through reinforcement learning (Ranzato et al., 2015; Paulus et al., 2017). This thesis focuses on methods trained with maximum likelihood estimation, and the works presented include both deep latent-variable models and normal neural models. Next, we briefly introduce the standard neural architectures and the state-of-the-art pretraining techniques for generation.

Architectures: Due to the sequential nature of language, early neural text generation methods mostly employ recurrent neural networks (RNN), such as long-short term memory network (LSTM, Hochreiter and Schmidhuber (1997)) or gated attention unit (GRU, Cho et al. (2014)). Convolutional neural networks (CNN) are also used despite being less common (Kalchbrenner et al., 2016). Conditional generation models typically use an encoder-decoder architecture. Later, the attention mechanism is introduced between encoder and decoder to greatly improve machine translation performance and soon becomes popular in other tasks (Bahdanau et al., 2015). In recent years, state-of-the-art NLG has been gradually dominated by self-attentional networks, Transformers (Vaswani et al., 2017). While most of these models are autoregressive, there are other non-autoregressive architectures that aims for smaller latency at inference time (Gu et al., 2018a; Ma et al., 2019).

Pretraining: Self-supervised pretraining, which trains models on large amount of raw text, has achieved great success and sets new state-of-the-art records in many NLP tasks (Peters et al., 2018; Devlin et al., 2019a; Liu et al., 2019a; Yang et al., 2019; Clark et al., 2020). The de-facto learning paradigm nowadays is to first pretrain a model on raw text, then fine-tune the pretrained model on downstream tasks. Current pretrained models are mostly based on the transformer architecture. For text generation specifically, the representative pretraining includes standard decoder-only language model pretraining such as GPT2 (Radford et al., 2019), GPT3 (Brown et al., 2020); and encoder-decoder sequence-to-sequence pretraining such as BART (Lewis et al., 2020a), MASS (Song et al., 2019), T5 (Raffel et al., 2020), and their multilingual versions for machine translation (Liu et al., 2020a; Xue et al., 2021).

2.2 Efficiencies in NLG

Efficiency in NLG can be considered from several different perspectives, related to the human annotations required, the time, space, or memory spent on training or inference, the computation for training or inference, the carbon footprint, the financial cost, etc. Efficiency is increasingly becoming a serious issue nowadays where the AI community focuses on obtaining "state-of-the-art" results with big data and models. The underlying techniques are unfortunately often inefficient. Given these circumstances, there are calls for Green AI – "AI research that is more environmentally friendly and inclusive" (Schwartz et al., 2020).

There are different types of efficiency. For example, *compute-efficient* approaches aim to save FLOPS at training time (Dai et al., 2020; Yao et al., 2021); *time-efficient* methods pursue better time complexity of the model, such as reducing time complexity of self-attention from quadratic to linear (Wang et al., 2020b; Choromanski et al., 2020; Ma et al., 2021), or non-autoregressive models for faster text generation (Gu et al., 2018a; Ma et al., 2019); and *energy-efficient* methods try to minimize carbon footprint. In this thesis and the literature review below, we mainly focus on label efficiency that tackles limited annotations, parameter efficiency that saves tuned parameters, and compute efficiency in retrieval-augmented NLG that reduces the inference overhead of retrieval.

2.2.1 Label Efficiency

Label-efficient approaches in NLG can be broadly classified into four categories to obtain performance gains with limited annotations: (1) transfer learning that resorts to annotations from other domains; (2) few-shot methods that can achieve competitive performance only fine-tuned with the given annotations; (3) data augmentation; and (4) un/semi-supervised methods to utilize unlabeled data from the same domain. While in this thesis we make efforts mainly from the last perspective, below we introduce related work on each of these directions.

Transfer Learning

Transfer learning is a method that boosts the performance on the target task or domain through training on source tasks or domains. Here we focus on the setting where the target task has no or limited number of annotated examples. In transfer learning, the model can be trained on the source and target data together in a multitask manner, or trained on source data first in a pretraining stage and then applied to the target task, dubbed sequential transfer learning.

Sequential transfer learning is the most popular form and achieved the biggest success so far. Traditionally, researchers pretrained word embeddings such as word2vec (Mikolov et al., 2013) or Glove (Pennington et al., 2014) that can be used to initialize the embedding layers in neural networks. Such embeddings are also extended to represent sentences or documents (Le and Mikolov, 2014; Conneau et al., 2017) which are helpful for sequence to sequence generation tasks. More recently, contextualized word embeddings are introduced which mark the dawn of the large-scale pretraining era (McCann et al., 2017; Peters et al., 2018; Devlin et al., 2019a). Particularly, transformer-based pretraining has demonstrated remarkable progress to improve NLG performance under low-data regime in general, such as the few-shot NLG applications in Peng et al. (2020); Chen et al. (2020c). One representative transfer learning example for low resource settings in NLP is cross-lingual transfer learning since most languages in the world lack sufficient annotations. Such issues are very common for cross-lingual applications like machine translation. To address this problem, pretraining goes to multilingual – for example, representation from different languages trained separately may be aligned afterwards (Ruder et al., 2019; Schuster et al., 2019); or different languages could share the same subword vocabulary (Sennrich et al., 2016c) and the same model is pretrained on all the data (Artetxe and Schwenk, 2019; Lample and Conneau, 2019; Pires et al., 2019; Wu and Dredze, 2019; Liu et al., 2020b; Xue et al., 2021).

In addition to sequential transfer learning, multitask learning has also been widely used to take advantage of source annotations, for example, multilingual machine translation that trains a universal machine translation model among multiple languages is an effective way to deal with low-resource language translation (Dong et al., 2015; Firat et al., 2016; Ha et al., 2016; Johnson et al., 2017a; Aharoni et al., 2019).

Few-shot Methods

Few-shot methods perform tasks without or with as few as tens of annotations. Large pretrained language models greatly improve label efficiency, textual prompts or instructions further advance label efficiency and leads to competitive few-shot performance. This is because most tasks can be reformulated as cloze questions through prompts (e.g. by simple asking "`the correct answer is ___`"), and such effect of textual prompts arguably comes from exploring the implicit task supervision from the raw text data during pretraining. Pioneer examples on this line are the GPT language models (Radford et al., 2019; Brown et al., 2020) that prepend annotated examples to the input as the prompt to perform few-shot generalization without fine-tuning, dubbed *in-context learning*. Such few-shot performance typically improves as the model size

grows, for instance, the largest GPT-3 model has 175 billion parameters. A more efficient way to utilize textual prompts is to design textual templates to re-format the input and output of the NLP task and tune the model on a few examples. It achieves competitive few-shot performance while the model is significantly smaller than GPT-3 language models (Schick and Schütze, 2021a,c; Gao et al., 2021a; Liu et al., 2021b). Le Scao and Rush (2021) further quantitatively show that prompting is often worth 100s of data points. Remarkably, prompting methods breaks the format boundary between NLP tasks, so that all NLP tasks could be treated in a unified format. Therefore, models trained on completely different tasks may be generalized to other tasks in a zero-shot fashion (Sanh et al., 2022; Wei et al., 2022). There are other attempts as well to tackle the few-shot challenge by tuning fewer parameters (Mahabadi et al., 2022).

Data Augmentation

Data augmentation is a class of approaches that increases training data diversity through automatically constructing pseudo paired input-output examples. Here we review some representative works on this line that are related to this thesis, and we refer readers to Feng et al. (2021) for a more comprehensive survey on this direction. Token-level perturbation is one of the simplest approaches to create new examples from existing ones, for example, Wei and Zou (2019) apply random insertion, deletion, swap, or synonym replacement operations; Şahin and Steedman (2018) swaps words relying on the dependency trees; On the sequence level, new sentences could be added by paraphrasing existing ones (Kumar et al., 2019; Longpre et al., 2019), or iterative masking and filling with language models (Ng et al., 2020). Inspired by MixUp (Zhang et al., 2017) for images that interpolate input and outputs to form new examples, in NLP researchers mix embeddings or hidden states instead (Chen et al., 2020a); Wang et al. (2018); Guo et al. (2020a) extend MixUp to sequence generations. Most of these approaches augment the dataset directly from existing annotated data, while there are other related works which construct pseudo paired data from unlabeled examples. For instance, the well-known back-translation technique that translates monolingual target data back to the source language has achieved great success in machine translation (Sennrich et al., 2016a); self-training (Scudder, 1965) instead predicts pseudo labels from the model itself given unlabeled inputs, which are mostly used in classification tasks. More recently, pretrained language models are leveraged to generate synthetic paired examples (Yang et al., 2020; Schick and Schütze, 2021b); in machine translation, cross-lingual paired examples could be retrieved automatically from the wild through aligned multilingual embeddings (Tran et al., 2020). As a simple and effective technique to overcome data scarce,

language augmentation features easy-to-use tools publicly available (Dhole et al., 2021).[1]

Un/Semi-Supervised Learning

Un/Semi-Supervised learning directly leverages unlabeled examples during training to boost performance. For example, Xie et al. (2020a) propose consistency loss to maximize consistency between two different views (e.g. paraphrases) of the same unlabeled input, which demonstrates promises on classification tasks. Such consistency training can be also understood as a form of data augmentation, named unsupervised data augmentation by the authors. Additionally, variational autoencoders (Kingma and Welling, 2014) can perform semi-supervised learning by modeling the targets as latent variables (Kingma et al., 2014). It has been successfully applied to sentence compression (Miao and Blunsom, 2016) and code generation tasks (Yin et al., 2018). Remarkably, back-translation combined with auto-encoding training has bootstrapped models from scratch and enabled unsupervised machine translation (Artetxe et al., 2017; Lample et al., 2018a), opening the door to fully unsupervised sequence to sequence generation.

Our work in Part I aim to leverage unlabeled examples in an effective way, and are the most related to the line of un/semi-supervised learning, while we emphasize on understanding and further proposing better methods based on our understanding. For example, Chapter 3 attempts to provide a probabilistic formulation for existing unsupervised machine translation techniques, the resulted latent-sequence generative model leads to improved results; Chapter 4 studies self-training in the state-of-the-art text generation models and performs analysis to understand the principle behind self-training, we then propose a novel, noisy variant of self-training to achieve significant gains; Chapter 5 extends consistency training to fully unsupervised, state-of-the-art task-generalization methods via prompting methods, where we propose to optimize prompt consistency instead of traditional example consistency. We note that the four sub-directions described in this section are largely orthogonal and could be potentially combined to complement each other.

2.2.2 Parameter Efficiency

As pretrained models grow larger and larger from hundreds of millions of parameters to trillions of parameters (Devlin et al., 2019a; Brown et al., 2020; Fedus et al., 2021; Chowdhery et al., 2022), the model size becomes a factor of concern, and could critically affect the model's practicality. Particularly, pretrained models are often fine-tuned on downstream tasks to achieve the best

[1] https://github.com/GEM-benchmark/NL-Augmenter, https://github.com/makcedward/nlpaug.

performance, thus multiple tasks would result in multiple model copies, which increasingly cause worries with ever-large pretrained language models. Below, we briefly summarize two direction to tackle this problem.

Model Compression

Model compression pursues to compress a trained large model to a smaller one. The most common technique is *distillation* (Hinton et al., 2015; Kim and Rush, 2016; Sanh et al., 2019; Jiao et al., 2020) that use the output distribution of a teacher model to train a student model. As a distillation variant, Xu et al. (2020a) replace submodules in a large model with smaller ones and train them to mimic the original output distribution.

On the other hand, *pruning* aims to remove redundant parameters from a trained model. Pruning can be further categorized as unstructured pruning and structured pruning, where the former zero out individual weights while the latter removes "blocks" from the neural network. Unstructured pruning is able to remove over 90% of the parameters, yet it is not well-supported by current hardware and could not really achieve an inference speed-up (Chen et al., 2020b; Sanh et al., 2020; Huang et al., 2021). In contrast, structured pruning is preferred practically. In terms of the transformer architecture specifically, Fan et al. (2020); Sajjad et al. (2020) drops entire transformer layers; Voita et al. (2019); Michel et al. (2019) removes heads from multi-head attention; Prasanna et al. (2020); Chen et al. (2021) selectively drop the feedforward block in transformers.

Another popular technique to reduce model sizes is quantization, which reduces the float precision of model parameters. Quantization can be subdivided into quantization-aware training (QAT) and post-training quantization (PTQ). QAT produces zero gradients by rounding floating-point numbers to fixed-point numbers, and thus adopts the straight-through estimator (Bengio et al., 2013) to approximate gradients typically during training (Choi et al., 2018; Gong et al., 2019; Esser et al., 2020). PTQ is based on the fact that most deep learning models can be safely quantized to 8-bit without re-training (Li et al., 2021b), while there are efforts pushing for 4-bit quantization (Nagel et al., 2020).

Parameter-efficient tuning

Different from model compression which mostly compresses a trained model, parameter-efficient tuning focuses on *tuning few parameters* (e.g. as few as 0.1% of all the parameters) of pretrained language models without losing performance on downstream tasks. We note that parameter-

efficient tuning does not reduce the model size at inference time for a single task, but mainly improves efficiency when the model serves multiple tasks. For example, it greatly reduces the disk space footprint when the same pretrained language model is fine-tuned on hundreds of downstream tasks, which facilitates model management. Also, a large portion of parameters sharing between tasks would potentially save running memory when deployed to serve multiple tasks simultaneously. This is a relatively new direction in NLP led by (Houlsby et al., 2019) which adds a light module, adapter, to a pretrained transformer model and only updates adapters during training; Hu et al. (2022) similarly adds low-rank matrices as the module to be learned; prefix tuning Li and Liang (2021b); Lester et al. (2021) prepends continuous prefixes as parameters to the input (or hidden states) as learnable parameters; Ben Zaken et al. (2021) propose to only update the bias of pretrained language models.

Our work in Part II study parameter-efficient tuning – we establish a unified framework to connect existing, state-of-the-art methods such as adapters and prefix tuning, and we further utilize the unified framework to instantiate better parameter-efficient tuning methods.

2.2.3 Compute Efficiency in Retrieval-Augmented NLG

Retrieval-Augmented NLG

Models need to acquire certain understanding abilities of the input and context and information to complete tasks, where such "ability and information" can be referred to as *knowledge*.[2] Most neural networks nowadays are fully parametric and store knowledge into parameters, as *internal knowledge*. This is done through gradient descent. Learning all required knowledge in such an internal form is not only challenging but also inefficient – training is expensive to learn a large amount of knowledge, for example, from the entire Wikipedia. Also, training is necessary to acquire new knowledge or update knowledge. On the other hand, knowledge may be provided in an external form directly through retrieval techniques, as *external knowledge*. This way, model does not need to be trained on large knowledge sources like Wikipedia but instead directly retrieves from it; adding new or updating stale knowledge also becomes much easier through replacing the datastore without training, leading to knowledge efficiency. Formally, retrieval-augmented NLG models the distribution $p(\mathbf{y}|\mathbf{x}|\mathbf{K})$:

$$p(\mathbf{y}|\mathbf{x}|\mathbf{K}) = \mathbb{E}_{\mathcal{S}\sim q(\mathcal{S}|\mathbf{x},\mathbf{K})} p(\mathbf{y}|\mathbf{x},\mathcal{S}), \tag{2.1}$$

[2]Sometimes people use "knowledge" to refer to factual knowledge specifically, while here we are referring a broader definition.

where $\mathbf{K}$ is the external knowledge datastore, and $\mathcal{S}$ is the subset of entries retrieved from it. $q(\mathcal{S}|\mathbf{x}, \mathbf{K})$ is usually referred to as the *retriever* while $p(\mathbf{y}|\mathbf{x}, \mathcal{S})$ is referred to as the *reader*. In many cases, the reader $p(\mathbf{y}|\mathbf{x}, \mathcal{S})$ takes one knowledge entry where $|\mathcal{S}| = 1$ and multiple retrieved items are aggregated through the expectation, yet $p(\mathbf{y}|\mathbf{x}, \mathcal{S})$ could deal with multiple retrieved items directly as well. Retrieval-augmented NLG has demonstrated state-of-the-art results on language modeling (Khandelwal et al., 2020), machine translation (Khandelwal et al., 2021), and open-domain question answering (Izacard and Grave, 2021). In Part III of this thesis, we focus on compute-efficiency of retrieval-augmented language models.

Knowledge Sources: Classic knowledge sources are structured such as knowledge bases (KBs) and knowledge graphs (KGs). In KBs, for example, "knowledge" consists of triples (subject, predicate, object), which are also referred to as "factual triplets". KB is an important source to store structured factual knowledge and several large KBs are public (Auer et al., 2007; Bollacker et al., 2008). KB-enhanced text generation has been widely used in question answering (Fu and Feng, 2018; Févry et al., 2020; Verga et al., 2021), dialogue applications (Madotto et al., 2018; Wang et al., 2020a; Wu et al., 2020), and language modeling (Liu et al., 2022). Knowledge graphs further organize the factual triplets into graphs that are more informative, and demonstrated successes to assist in text generation as well (Bauer et al., 2018; Liu et al., 2019b; Tuan et al., 2019; Zhang et al., 2020a; Zhao et al., 2020b; Ji et al., 2020; Xu et al., 2020b). While KBs or KGs are interpretable and relatively accurate compared to unstructured sources, they are also difficult to construct: large, complete KBs are not available for many domains and languages, and automatic construction of KBs from raw text is still challenging which produces very noisy outputs. Moreover, KBs or KGs only represent a limited form of knowledge that could be represented by simple triplets, while there is far more knowledge omitted from such sources, for example, the content of the inauguration speech of Barack Obama cannot be covered by triplets, not to mention the general notion of "knowledge" that we described in the beginning of § 2.2.3 is beyond factual knowledge. To overcome these challenges, researchers turn to unstructured text as the knowledge sources, which are easily available and do not limit the scope of knowledge. For example, Chen et al. (2017) retrieves from Wikipedia articles directly for question answering as a pioneer attempt on this direction, then a series of follow-up work leverages Wikipedia to achieve state-of-the-art performance on open-domain question answering (Lee et al., 2019; Karpukhin et al., 2020; Izacard and Grave, 2021); Similarly, Dinan et al. (2019); Kim et al. (2020); Zhao et al. (2020a) utilizes Wikipedia for dialogue; Guu et al. (2020) incorporate Wikipedia articles into mask language model pretraining; Lewis et al. (2020b) employ this framework for various

knowledge-intensive generation tasks; Petroni et al. (2021) release a benchmark for Wikipedia-augmented generation tasks; Gu et al. (2018b); Guu et al. (2018); Khandelwal et al. (2020, 2021); Wang et al. (2022) retrieves from training data to improve language modeling, machine translation, and summarization; Grave et al. (2016); Wu et al. (2022) retrieve from test context to extend the context length; Borgeaud et al. (2021) scale up the datastore sizes and train a language model that retrieves from trillions of tokens on the web. Inspired by how humans perform tasks in daily life through using search engines, there are recent efforts teaching models to use browsers (e.g. scroll web pages, click a link, etc.) through reinforcement learning (Nakano et al., 2021). Our work in Part III follow this trend and take unstructured text as the knowledge source due to its friendly availability and generality. Specifically, we retrieve from the domain-specific training data to improve language modeling.

Retriever: The retriever plays a critical role in retrieval-augmented generation, and is often considered the main bottleneck currently for the final performance. The retrievers can be broadly categorized as sparse retrievers such as TF-IDF and BM25 (Robertson and Zaragoza, 2009), and dense retrievers that rely on dense embeddings. Heuristic word match is typically used when the knowledge source is KB or KG (Fu and Feng, 2018; Madotto et al., 2018; Zhao et al., 2020b; Liu et al., 2022), while dense retrieval is widely used and sets the state-of-the-art performance to retrieve from text (Lewis et al., 2020b; Izacard and Grave, 2021; Lee et al., 2021). Dense retriever mostly adopts a bi-encoder structure – all the items in the datastore are encoded as dense vectors with a datastore encoder, the query is encoded as a query embedding, then similarity search is performed to retrieve the most similar datastore vectors to the query vector. The best dense retrievers are often trained (or fine-tuned) through supervised data (Chen et al., 2017; Reimers and Gurevych, 2019a; Li et al., 2020; Gao and Callan, 2021; Gao et al., 2021b), which could underperform unsupervised sparse retrieval approaches in new domains, and robust dense retrieval is still an open research direction (Izacard et al., 2021). Recently, some researchers take a detour from the bi-encoder structure and instead study *generative retrieval*, where a neural decoder directly generates the retrieved items without similarity search on a high-dimensional space (De Cao et al., 2020; Tay et al., 2022).

Reader: The reader, $p(\mathbf{y}|\mathbf{x}, \mathcal{S})$, parameterizes how we fuse the knowledge into the neural network. Traditionally, researchers often encode the retrieved items separately and fuse them in through engineering the original model architectures (Chen et al., 2017; Madotto et al., 2018), the emergence of powerful pretrained language models mitigates the necessity for special treatment

of the fusion architecture, as many recent works directly concatenate the retrieved item to the input to demonstrate effectiveness (Guu et al., 2020; Lewis et al., 2020b; Wang et al., 2022).

Efficiency Issue

One limitation of retrieval-augmented text generation is that it adds extra overhead to maintain a large datastore at inference time and retrieves from it. This process causes a non-trivial amount of space and time cost. For example, Khandelwal et al. (2020) produces datastore of size over 1000 GB from a billion-token dataset; Khandelwal et al. (2021) observe two orders of magnitude slowdown during translation at inference time; Similarity, state-of-the-art retrieval-augmented question answering can only handle less than 10 questions per second (Lee et al., 2021). This efficiency problem greatly influences the practical usage of the retrieval-augmented method, and has recently attracted attention from the field: (Meng et al., 2021; Martins et al., 2022) try to improve the inference speed of nearest neighbor machine translation, and Min et al. (2021) describe a space-efficient question answering competition and the participant systems, where they are required to operate under small space limits. Our work in Part III focuses on this aspect and aims for a lighter retrieval-augmented generation. We focus on the nearest neighbor language model class (Chapter 8) and prototype-based language model class (Chapter 7) respectively, and propose novel methods to reduce the datastore size and speed up the retrieval process.

2.3 Connection of Different Efficiencies

Different types of efficiency are not independent, instead, they are closely related to each other, and tradeoff is often present between them: larger models are typically more label-efficient, compute-efficient, and converge faster (Kaplan et al., 2020); parameter-efficient tuning needs more training iterations and computation; Retrieval-augmented methods incur retrieval overhead, yet may end up saving many parameters since the required knowledge is not needed to be stored into parameters (Borgeaud et al., 2021). Importantly, these tradeoffs mean different costs and returns for each method, for example, label-efficient methods pay extra training (i.e. computation, time) to potentially save human annotation; parameter-efficient methods pay training computation to save inference time or space.

Given these tradeoffs, one may wonder what would be the optimal method to choose in practice – it depends on the main resource limitation to specific researchers and practitioners. Human annotation, computation complexity, and space or memory usage could all end up being

the balance of financial cost, time, and carbon footprint to the environment (Strubell et al., 2019).

Part I

Label-Efficient Natural Language Generation

Chapter 3

Unsupervised Sequence to Sequence Transduction

In this chapter, we introduce a probabilistic framework for unsupervised sequence to sequence transduction. The presented approach does not need any parallel data to train, and is thus sample-efficient. Some example sequence to sequence transduction tasks in this chapter include sentiment transfer, word decipherment, and related language translation. This work is presented in:

- Junxian He*, Xinyi Wang* (equal contribution), Graham Neubig, and Taylor Berg-Kirkpatrick. A Probabilistic Formulation of Unsupervised Text Style Transfer. In *International Conference on Learning Representations (ICLR)*, 2020

3.1 Introduction

Text sequence transduction systems convert a given text sequence from one domain to another. These techniques can be applied to a wide range of natural language processing applications such as machine translation (Bahdanau et al., 2015), summarization (Rush et al., 2015), and dialogue response generation (Zhao et al., 2017). In many cases, however, parallel corpora for the task at hand are scarce. Therefore, *unsupervised* sequence transduction methods that require only non-parallel data are appealing and have been receiving growing attention (Bannard and Callison-Burch, 2005; Ravi and Knight, 2011; Mizukami et al., 2015; Shen et al., 2017; Lample et al., 2018b, 2019). This trend is most pronounced in the space of text *style transfer* tasks where parallel data is particularly challenging to obtain (Hu et al., 2017; Shen et al., 2017; Yang et al., 2018). Style transfer has historically referred to sequence transduction problems that modify

superficial properties of text – i.e. style rather than content.[1] We focus on a standard suite of style transfer tasks, including formality transfer (Rao and Tetreault, 2018), author imitation (Xu et al., 2012), word decipherment (Shen et al., 2017), sentiment transfer (Shen et al., 2017), and related language translation (Pourdamghani and Knight, 2017). General unsupervised translation has not typically been considered style transfer, but for the purpose of comparison we also conduct evaluation on this task (Lample et al., 2018a).

Recent work on unsupervised text style transfer mostly employs non-generative or non-probabilistic modeling approaches. For example, Shen et al. (2017) and Yang et al. (2018) design adversarial discriminators to shape their unsupervised objective – an approach that can be effective, but often introduces training instability. Other work focuses on directly designing unsupervised training objectives by incorporating intuitive loss terms (e.g. backtranslation loss), and demonstrates state-of-the-art performance on unsupervised machine translation (Lample et al., 2018b; Artetxe et al., 2019) and style transfer (Lample et al., 2019). However, the space of possible unsupervised objectives is extremely large and the underlying modeling assumptions defined by each objective can only be reasoned about indirectly. As a result, the process of designing such systems is often heuristic.

In contrast, probabilistic models (e.g. the noisy channel model (Shannon, 1948)) define assumptions about data more explicitly and allow us to reason about these assumptions during system design. Further, the corresponding objectives are determined naturally by principles of probabilistic inference, reducing the need for empirical search directly in the space of possible objectives. That said, classical probabilistic models for unsupervised sequence transduction (e.g. the HMM or semi-HMM) typically enforce overly strong independence assumptions about data to make exact inference tractable (Knight et al., 2006; Ravi and Knight, 2011; Pourdamghani and Knight, 2017). This has restricted their development and caused their performance to lag behind unsupervised neural objectives on complex tasks. Luckily, in recent years, powerful variational approximation techniques have made it more practical to train probabilistic models without strong independence assumptions (Miao and Blunsom, 2016; Yin et al., 2018). Inspired by this, we take a new approach to unsupervised style transfer.

We directly define a generative probabilistic model that treats a non-parallel corpus in two domains as a partially observed parallel corpus. Our model makes few independence assumptions and its true posterior is intractable. However, we show that by using amortized variational inference (Kingma and Welling, 2014), a principled probabilistic technique, a natural unsupervised

[1]Notably, some tasks we evaluate on do change content to some degree, such as sentiment transfer, but for conciseness we use the term "style transfer" nonetheless.

objective falls out of our modeling approach that has many connections with past work, yet is different from all past work in specific ways. In experiments across a suite of unsupervised text style transfer tasks, we find that the natural objective of our model actually outperforms all manually defined unsupervised objectives from past work, supporting the notion that probabilistic principles can be a useful guide even in deep neural systems. Further, in the case of unsupervised machine translation, our model matches the current state-of-the-art non-generative approach.

3.2 Unsupervised Text Style Transfer

We first overview text style transfer, which aims to transfer a text (typically a single sentence or a short paragraph – for simplicity we refer to simply "sentences" below) from one domain to another while preserving underlying content. For example, formality transfer (Rao and Tetreault, 2018) is the task of transforming the tone of text from informal to formal without changing its content. Other examples include sentiment transfer (Shen et al., 2017), word decipherment (Knight et al., 2006), and author imitation (Xu et al., 2012). If parallel examples were available from each domain (i.e. the training data is a bitext consisting of pairs of sentences from each domain), supervised techniques could be used to perform style transfer (e.g. attentional Seq2Seq (Bahdanau et al., 2015) and Transformer (Vaswani et al., 2017)). However, for most style transfer problems, only non-parallel corpora (one corpus from each domain) can be easily collected. Thus, work on style transfer typically focuses on the more difficult unsupervised setting where systems must learn from non-parallel data alone.

The model we propose treats an observed non-parallel text corpus as a partially observed parallel corpus. Thus, we introduce notation for both observed text inputs and those that we will treat as latent variables. Specifically, we let $X = \{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ represent observed data from domain $\mathcal{D}_1$, while we let $Y = \{y^{(m+1)}, y^{(m+2)}, \cdots, y^{(n)}\}$ represent observed data from domain $\mathcal{D}_2$. Corresponding indices represent parallel sentences. Thus, none of the observed sentences share indices. In our model, we introduce latent sentences to complete the parallel corpus. Specifically, $\bar{X} = \{\bar{x}^{(m+1)}, \bar{x}^{(m+2)}, \cdots, \bar{x}^{(n)}\}$ represents the set of latent parallel sentences in $\mathcal{D}_1$, while $\bar{Y} = \{\bar{y}^{(1)}, \bar{y}^{(2)}, \cdots, \bar{y}^{(m)}\}$ represents the set of latent parallel sentences in $\mathcal{D}_2$. Then the goal of unsupervised text transduction is to infer these latent variables conditioned the observed non-parallel corpora; that is, to learn $p(\bar{y}|x)$ and $p(\bar{x}|y)$.

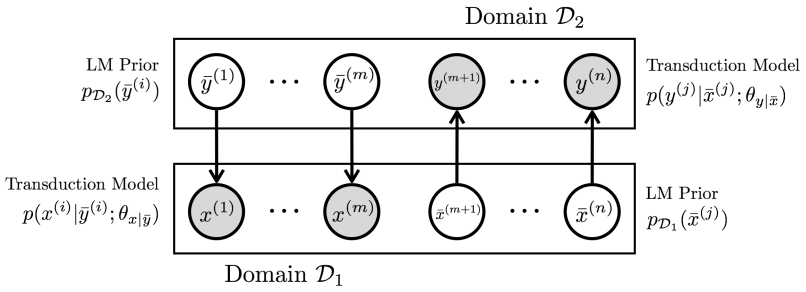


Figure 3.1: Proposed graphical model for style transfer via bitext completion. Shaded circles denote the observed variables and unshaded circles denote the latents. The generator is parameterized as an encoder-decoder architecture and the prior on the latent variable is a pretrained language model.

3.3 The Deep Latent Sequence Model

First we present our generative model of bitext, which we refer to as a deep latent sequence model. We then describe unsupervised learning and inference techniques for this model class.

3.3.1 Model Structure

Directly modeling $p(\bar{y}|x)$ and $p(\bar{x}|y)$ in the unsupervised setting is difficult because we never directly observe parallel data. Instead, we propose a generative model of the complete data that defines a joint likelihood, $p(X, \bar{X}, Y, \bar{Y})$. In order to perform text transduction, the unobserved halves can be treated as latent variables: they will be marginalized out during learning and inferred via posterior inference at test time.

Our model assumes that each observed sentence is generated from an unobserved parallel sentence in the opposite domain, as depicted in Figure 1. Specifically, each sentence $x^{(i)}$ in domain $\mathcal{D}_1$ is generated as follows: First, a latent sentence $\bar{y}^{(i)}$ in domain $\mathcal{D}_2$ is sampled from a prior, $p_{\mathcal{D}_2}(\bar{y}^{(i)})$. Then, $x^{(i)}$ is sampled conditioned on $\bar{y}^{(i)}$ from a transduction model, $p(x^{(i)}|\bar{y}^{(i)})$. Similarly, each observed sentence $y^{(j)}$ in domain $\mathcal{D}_2$ is generated conditioned on a latent sentence, $\bar{x}^{(j)}$, in domain $\mathcal{D}_1$ via the opposite transduction model, $p(y^{(j)}|\bar{x}^{(j)})$, and prior, $p_{\mathcal{D}_1}(\bar{x}^{(j)})$. We let $\theta_{x|\bar{y}}$ and $\theta_{y|\bar{x}}$ represent the parameters of the two transduction distributions respectively. We assume the prior distributions are pretrained on the observed data in their respective domains and therefore omit their parameters for simplicity of notation. Together, this gives the following joint likelihood:

$$p(X, \bar{X}, Y, \bar{Y}; \theta_{x|\bar{y}}, \theta_{y|\bar{x}}) = \left(\prod_{i=1}^{m} p\big(x^{(i)}|\bar{y}^{(i)}; \theta_{x|\bar{y}}\big) p_{\mathcal{D}_2}\big(\bar{y}^{(i)}\big)\right) \left(\prod_{j=m+1}^{n} p\big(y^{(j)}|\bar{x}^{(j)}; \theta_{y|\bar{x}}\big) p_{\mathcal{D}_1}\big(\bar{x}^{(j)}\big)\right) \tag{3.1}$$

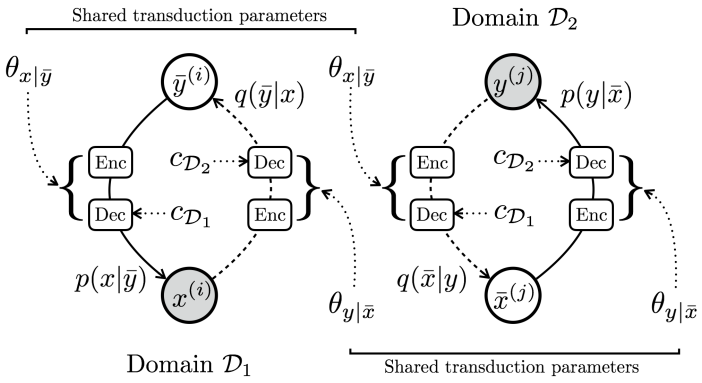


Figure 3.2: Depiction of amortized variational approximation. Distributions $q(\bar{y}|x)$ and $q(\bar{x}|y)$ represent inference networks that approximate the model's true posterior. Critically, parameters are shared between the generative model and inference networks to tie the learning problems for both domains.

The log marginal likelihood of the data, which we will approximate during training, is:

$$\log p(X, Y; \theta_{x|\bar{y}}, \theta_{y|\bar{x}}) = \log \sum_{\bar{X}} \sum_{\bar{Y}} p(X, \bar{X}, Y, \bar{Y}; \theta_{x|\bar{y}}, \theta_{y|\bar{x}}) \tag{3.2}$$

Note that if the two transduction models share no parameters, the training problems for each observed domain are independent. Critically, we introduce parameter sharing through our variational inference procedure, which we describe in more detail in § 3.3.2.

Architecture: Since we would like to be able to model a variety of transfer tasks, we choose a parameterization for our transduction distributions that makes no independence assumptions. Specifically, we employ an encoder-decoder architecture based on the standard attentional Seq2Seq model which has been shown to be successful across various tasks (Bahdanau et al., 2015; Rush et al., 2015). Similarly, our prior distributions for each domain are parameterized as recurrent language models which, again, make no independence assumptions. In contrast, traditional unsupervised generative sequence models typically make strong independence assumptions to enable exact inference (e.g. the HMM makes a Markov assumption on the latent sequence and emissions are one-to-one). Our model is more flexible, but exact inference via dynamic programming will be intractable. We address this problem in the next section.

3.3.2 Learning

Ideally, learning should directly optimize the log data likelihood, which is the marginal of our model shown in Eq. 3.2. However, due to our model's neural parameterization which does not

factorize, computing the data likelihood cannot be accomplished using dynamic programming as can be done with simpler models like the HMM. To overcome the intractability of computing the true data likelihood, we adopt amortized variational inference (Kingma and Welling, 2014) in order to derive a surrogate objective for learning, the evidence lower bound (ELBO) on log marginal likelihood[2] :

$$
\begin{aligned}
&\log p(X, Y; \theta_{x|\bar{y}}, \theta_{y|\bar{x}}) \\
&\geq \mathcal{L}_{\text{ELBO}}(X, Y; \theta_{x|\bar{y}}, \theta_{y|\bar{x}}, \phi_{\bar{x}|y}, \phi_{\bar{y}|x}) \\
&= \sum\nolimits_i \Big[\mathbb{E}_{q(\bar{y}|x^{(i)};\phi_{\bar{y}|x})}[\log p(x^{(i)}|\bar{y}; \theta_{x|\bar{y}})] - D_{\text{KL}}\big(q(\bar{y}|x^{(i)}; \phi_{\bar{y}|x})||p_{\mathcal{D}_2}(\bar{y})\big)\Big] \\
&+ \sum\nolimits_j \underbrace{\Big[\mathbb{E}_{q(\bar{x}|y^{(j)};\phi_{\bar{x}|y})}[\log p(y^{(j)}|\bar{x}; \theta_{y|\bar{x}})]}_{\text{Reconstruction likelihood}} \underbrace{- D_{\text{KL}}\big(q(\bar{x}|y^{(j)}; \phi_{\bar{x}|y})||p_{\mathcal{D}_1}(\bar{x})\big)\Big]}_{\text{KL regularizer}}
\end{aligned}
\tag{3.3}
$$

The surrogate objective introduces $q(\bar{y}|x^{(i)}; \phi_{\bar{y}|x})$ and $q(\bar{x}|y^{(j)}; \phi_{\bar{x}|y})$, which represent two separate inference network distributions that approximate the model's true posteriors, $p(\bar{y}|x^{(i)}; \theta_{x|\bar{y}})$ and $p(\bar{x}|y^{(j)}; \theta_{y|\bar{x}})$, respectively. Learning operates by jointly optimizing the lower bound over both variational and model parameters. Once trained, the variational posterior distributions can be used directly for style transfer. The KL terms in Eq. 3.3, that appear naturally in the ELBO objective, can be intuitively viewed as regularizers that use the language model priors to bias the induced sentences towards the desired domains. Amortized variational techniques have been most commonly applied to continuous latent variables, as in the case of the variational autoencoder (VAE) (Kingma and Welling, 2014). Here, we use this approach for inference over discrete sequences, which has been shown to be effective in related work on a semi-supervised task (Miao and Blunsom, 2016).

Inference Network and Parameter Sharing: Note that the approximate posterior on one domain aims to learn the *reverse* style transfer distribution, which is exactly the goal of the generative distribution in the opposite domain. For example, the inference network $q(\bar{y}|x^{(i)}; \phi_{\bar{y}|x})$ and the generative distribution $p(y|\bar{x}^{(i)}; \theta_{y|\bar{x}})$ both aim to transform $\mathcal{D}_1$ to $\mathcal{D}_2$. Therefore, we use the same architecture for each inference network as used in the transduction models, and tie their parameters: $\phi_{\bar{x}|y} = \theta_{x|\bar{y}}, \phi_{\bar{y}|x} = \theta_{y|\bar{x}}$. This means we learn only two encoder-decoders overall – which are parameterized by $\theta_{x|\bar{y}}$ and $\theta_{y|\bar{x}}$ respectively – to represent two directions of transfer. In

[2]Note that in practice, we add a weight λ (the same to both domains) to the KL term in ELBO since the regularization strength from the pretrained language model varies depending on the datasets, training data size, or language model structures. Such reweighting has proven necessary in previous work that is trained with ELBO (Bowman et al., 2016; Miao and Blunsom, 2016; Yin et al., 2018).

addition to reducing the number of learnable parameters, this parameter tying couples the learning problems for both domains and allows us to jointly learn from the full data. Moreover, inspired by recent work that builds a universal Seq2Seq model to translate between different language pairs (Johnson et al., 2017b), we introduce further parameter tying between the two directions of transduction: the same encoder is employed for both x and y, and a domain embedding c is provided to the same decoder to specify the transfer direction, as shown in Figure 3.2. Ablation analysis in § 3.5.5 suggests that parameter sharing is important to achieve good performance.

Approximating Gradients of ELBO: The reconstruction and KL terms in Eq. 3.3 still involve intractable expectations due to the marginalization over the latent sequence, thus we need to approximate their gradients. Gumbel-softmax (Jang et al., 2017) and REINFORCE (Williams, 1987) are often used as stochastic gradient estimators in the discrete case. Since the latent text variables have an extremely large domain, we find that REINFORCE-based gradient estimates result in high variance. Thus, we use the Gumbel-softmax straight-through estimator to backpropagate gradients from the KL terms.[3] However, we find that approximating gradients of the reconstruction loss is much more challenging – both the Gumbel-softmax estimator and REINFORCE are unable to outperform a simple *stop-gradient* method that does not back-propagate the gradient of the latent sequence to the inference network. This confirms a similar observation in previous work on unsupervised machine translation (Lample et al., 2018b). Therefore, we use greedy decoding without recording gradients to approximate the reconstruction term.[4] Note that the inference networks still receive gradients from the prior through the KL term, and their parameters are shared with the decoders which do receive gradients from reconstruction. We consider this to be the best empirical compromise at the moment.

Initialization. Good initialization is often necessary for successful optimization of unsupervised learning objectives. In preliminary experiments, we find that the encoder-decoder structure has difficulty generating realistic sentences during the initial stages of training, which usually results in a disastrous local optimum. This is mainly because the encoder-decoder is initialized randomly and there is no direct training signal to specify the desired latent sequence in the unsupervised setting. Therefore, we apply a self-reconstruction loss $\mathcal{L}_{\text{rec}}$ at the initial epochs of training. We denote the output the encoder as $e(\cdot)$ and the decoder distribution as p_{dec}, then

$$\mathcal{L}_{\text{rec}} = -\alpha \cdot \sum\nolimits_i [p_{\text{dec}}(e(x^{(i)}), c_x)] - \alpha \cdot \sum\nolimits_j [p_{\text{dec}}(e(y^{(j)}), c_y)], \tag{3.4}$$

[3] We use one sample to approximate the expectations.

[4] We compare greedy and sampling decoding in § 3.5.5.

c_x and c_y are the domain embeddings for x and y respectively, α decays from 1.0 to 0.0 linearly in the first k epochs. k is a tunable parameter and usually less than 3 in all our experiments.

3.4 Connection to Related Work

Our probabilistic formulation can be connected with recent advances in unsupervised text transduction methods. For example, back translation loss (Sennrich et al., 2016b) plays an important role in recent unsupervised machine translation (Artetxe et al., 2018; Lample et al., 2018b; Artetxe et al., 2019) and unsupervised style transfer systems (Lample et al., 2019). In order to incorporate back translation loss the source language x is translated to the target language y to form a pseudo-parallel corpus, then a translation model from y to x can be learned on this pseudo bitext just as in supervised setting. While back translation was often explained as a data augmentation technique, in our probabilistic formulation it appears naturally with the ELBO objective as the reconstruction loss term.

Some previous work has incorporated a pretrained language models into neural semi-supervised or unsupervised objectives. He et al. (2016) uses the log likelihood of a pretrained language model as the reward to update a supervised machine translation system with policy gradient. Artetxe et al. (2019) utilize a similar idea for unsupervised machine translation. Yang et al. (2018) employed a similar approach, but interpret the LM as an adversary, training the generator to fool the LM. We show how our ELBO objective is connected with these more heuristic LM regularizers by expanding the KL loss term (assume x is observed):

$$D_{\mathrm{KL}}(q(\bar{y}|x)||p_{\mathcal{D}_2}(\bar{y})) = -H_q - \mathbb{E}_q[\log p_{\mathcal{D}_2}(\bar{y})], \tag{3.5}$$

Note that the loss used in previous work does not include the negative entropy term, $-H_q$. Our objective results in this additional "regularizer", the negative entropy of the transduction distribution, $-H_q$. Intuitively, $-H_q$ helps avoid a peaked transduction distribution, preventing the transduction from constantly generating similar sentences to satisfy the language model. In experiments we will show that this additional regularization is important and helps bypass bad local optima and improve performance. These important differences with past work suggest that a probabilistic view of the unsupervised sequence transduction may provide helpful guidance in determining effective training objectives.

3.5 Experiments

We test our model on five style transfer tasks: sentiment transfer, word substitution decipherment, formality transfer, author imitation, and related language translation. For completeness, we also evaluate on the task of general unsupervised machine translation using standard benchmarks.

We compare with the unsupervised machine translation model (UNMT) which recently demonstrated state-of-the-art performance on transfer tasks such as sentiment and gender transfer (Lample et al., 2019).[5] To validate the effect of the negative entropy term in the KL loss term Eq. 3.5, we remove it and train the model with a back-translation loss plus a language model negative log likelihood loss (which we denote as BT+NLL) as an ablation baseline. For each task, we also include strong baseline numbers from related work if available. For our method we select the model with the best validation ELBO, and for UNMT or BT+NLL we select the model with the best back-translation loss.

3.5.1 Model Configurations

We adopt the following attentional encoder-decoder architecture for UNMT, BT+NLL, and our method across all the experiments:

- We use word embeddings of size 128.
- We use 1 layer LSTM with hidden size of 512 as both the encoder and decoder.
- We apply dropout to the readout states before softmax with a rate of 0.3.
- Following Lample et al. (2019), we add a max pooling operation over the encoder hidden states before feeding it to the decoder. Intuitively the pooling window size would control how much information is preserved during transduction. A window size of 1 is equivalent to standard attention mechanism, and a large window size corresponds to no attention. See Appendix 3.5.2 for how to select the window size.
- There is a noise function for UNMT baseline in its denoising autoencoder loss (Lample et al., 2018a, 2019). We use the default noise function and noise hyperparameters in Lample et al. (2018a). For BT+NLL and our method we found that adding the extra noise into the self-reconstruction loss (Eq. 3.4) often hurts the performance, because we already have a

[5]The model they used is slightly different from the original model of Lample et al. (2018b) in certain details – e.g. the addition of a pooling layer after attention. We re-implement their model in our codebase for fair comparison and verify that our re-implementation achieves performance competitive with the original paper.

language model to avoid the local optimum of direct-copy generation.

3.5.2 Hyperparameter Tuning.

We vary pooling windows size as $\{1, 5\}$, the decaying patience hyperparameter k for self-reconstruction loss (Eq. 3.4) as $\{1, 2, 3\}$. For the baselines UNMT and BT+NLL, we also try the option of not annealing the self-reconstruction loss at all as in the unsupervised machine translation task (Lample et al., 2018b). We vary the weight λ for the NLL term (BT+NLL) or the KL term (our method) as $\{0.001, 0.01, 0.03, 0.05, 0.1\}$.

3.5.3 Datasets and Experiment Setup

Word Substitution Decipherment. Word decipherment aims to uncover the plain text behind a corpus that was enciphered via word substitution where word in the vocabulary is mapped to a unique type in a cipher dictionary (Dou and Knight, 2013; Shen et al., 2017; Yang et al., 2018). In our formulation, the model is presented with a non-parallel corpus of English plaintext and the ciphertext. We use the data in (Yang et al., 2018) which provides 200K sentences from each domain. While previous work (Shen et al., 2017; Yang et al., 2018) controls the difficulty of this task by varying the percentage of words that are ciphered, we directly evaluate on the most difficult version of this task – 100% of the words are enciphered (i.e. no vocabulary sharing in the two domains). We select the model with the best unsupervised reconstruction loss, and evaluate with BLEU score on the test set which contains 100K parallel sentences. Results are shown in Table 3.2.

Sentiment Transfer. Sentiment transfer is a task of paraphrasing a sentence with a different sentiment while preserving the original content. Evaluation of sentiment transfer is difficult and is still an open research problem (Mir et al., 2019). Evaluation focuses on three aspects: attribute control, content preservation, and fluency. A successful system needs to perform well with respect to all three aspects. We follow prior work by using three automatic metrics (Yang et al., 2018; Lample et al., 2019): classification accuracy, self-BLEU (BLEU of the output with the original sentence as the reference), and the perplexity (PPL) of each system's output under an external language model. We pretrain a convolutional classifier (Kim, 2014) to assess classification accuracy, and use an LSTM language model pretrained on each domain to compute the PPL of system outputs.

We use the Yelp reviews dataset collected by Shen et al. (2017) which contains 250K negative sentences and 380K positive sentences. We also use a small test set that has 1000 human-annotated

Table 3.1: Results on the sentiment transfer, author imitation, and formality transfer. We list the PPL of pretrained LMs on the test sets of both domains. We only report Self-BLEU on the sentiment task to compare with existing work.

Task	Model	Acc.	BLEU	Self-BLEU	PPL$_{\mathcal{D}_1}$	PPL$_{\mathcal{D}_2}$
Sentiment	Test Set	-	-	-	31.97	21.87
	Shen et al. (2017)	79.50	6.80	12.40	50.40	52.70
	Hu et al. (2017)	87.70	-	**65.60**	115.60	239.80
	Yang et al. (2018)	83.30	13.40	38.60	30.30	42.10
	UNMT	87.17	16.99	44.88	26.53	35.72
	BT+NLL	**88.36**	12.36	31.48	8.75	12.82
	Ours	87.90	**18.67**	48.38	27.75	35.61
Author Imitation	Test Set	-	-	-	132.95	85.25
	UNMT	80.23	7.13	-	40.11	39.38
	BT+NLL	76.98	10.80	-	61.70	65.51
	Ours	**81.43**	**10.81**	-	49.62	44.86
Formality	Test Set	-	-	-	71.30	135.50
	UNMT	78.06	16.11	-	26.70	10.38
	BT+NLL	**82.43**	8.57	-	6.57	8.21
	Ours	80.46	**18.54**	-	22.65	17.23

parallel sentences introduced in Li et al. (2018). We denote the positive sentiment as domain $\mathcal{D}_1$ and the negative sentiment as domain $\mathcal{D}_2$. We use Self-BLEU and BLEU to represent the BLEU score of the output against the original sentence and the reference respectively. Results are shown in Table 3.1.

Formality Transfer. Next, we consider a harder task of modifying the formality of a sequence. We use the GYAFC dataset (Rao and Tetreault, 2018), which contains formal and informal sentences from two different domains. In this chapter, we use the Entertainment and Music domain, which has about 52K training sentences, 5K development sentences, and 2.5K test sentences. This dataset actually contains parallel data between formal and informal sentences, which we use only for evaluation. We follow the evaluation of sentiment transfer task and test models on three axes. Since the test set is a parallel corpus, we only compute reference BLEU and ignore self-BLEU. We use $\mathcal{D}_1$ to denote formal text, and $\mathcal{D}_2$ to denote informal text. Results are shown in Table 3.1.

Author Imitation. Author imitation is the task of paraphrasing a sentence to match another

Table 3.2: BLEU for decipherment, related language translation (Sr-Bs), and general unsupervised translation (En-De).

Model	Decipher	Sr-Bs	Bs-Sr	En-De	De-En
Shen et al. (2017)	50.8	-	-	-	-
Yang et al. (2018)	49.3	31.0	33.4	-	-
UNMT	76.4	31.4	33.4	26.5	32.2
BT+NLL	78.0	29.6	31.4	-	-
Ours	**78.4**	**36.2**	**38.3**	26.9	32.0

author's style. The dataset we use is a collection of Shakespeare's plays translated line by line into modern English. It was collected by Xu et al. (2012)[6] and used in prior work on supervised style transfer (Jhamtani et al., 2017). This is a parallel corpus and thus we follow the setting in the formality transfer task. We use $\mathcal{D}_1$ to denote modern English, and $\mathcal{D}_2$ to denote Shakespeare-style English. Results are shown in Table 3.1.

Related Language Translation. Next, we test our method on a challenging related language translation task (Pourdamghani and Knight, 2017; Yang et al., 2018). This task is a natural test bed for unsupervised sequence transduction since the goal is to preserve the meaning of the source sentence while rewriting it into the target language. For our experiments, we choose Bosnian (bs) and Serbian (sr) as the related language pairs. We follow Yang et al. (2018) to report BLEU-1 score on this task since BLEU-4 score is close to zero. Results are shown in Table 3.2.

Unsupervised MT. In order to draw connections with a related work on general unsupervised machine translation, we also evaluate on the WMT'16 German English translation task. This task is substantially more difficult than the style transfer tasks considered so far. We compare with the state-of-the-art UNMT system using the existing implementation from the XLM codebase,[7] and implement our approach in the same framework with XLM initialization for fair comparison. We train both systems on 5M non-parallel sentences from each language. Results are shown in Table 3.2.

In Tables 3.1 we also list the PPL of the test set under the external LM for both the source and target domain. PPL of system outputs should be compared to PPL of the test set itself because extremely low PPL often indicates that the generated sentences are short or trivial.

[6] https://github.com/tokestermw/tensorflow-shakespeare
[7] https://github.com/facebookresearch/XLM

3.5.4 Results

Tables 3.1 and 3.2 demonstrate some general trends. First, UNMT is able to outperform other prior methods in unsupervised text style transfer, such as (Yang et al., 2018; Hu et al., 2017; Shen et al., 2017). The performance improvements of UNMT indicate that flexible and powerful architectures are crucial (prior methods generally do not have an attention mechanism). Second, our model achieves comparable classification accuracy to UNMT but outperforms it in all style transfer tasks in terms of the reference-BLEU, which is the most important metric since it directly measures the quality of the final generations against gold parallel data. This indicates that our method is both effective and consistent across many different tasks. Finally, the BT+NLL baseline is sometimes quite competitive, which indicates that the addition of a language model alone can be beneficial. However, our method consistently outperforms the simple BT+NLL method, which indicates the effectiveness of the additional entropy regularizer in Eq. 3.5 that is the byproduct of our probabilistic formulation.

Next, we examine the PPL of the system outputs under pretrained domain LMs, which should be evaluated in comparison with the PPL of the test set itself. For both the sentiment transfer and the formality transfer tasks in Table 3.1, BT+NLL achieves extremely low PPL, lower than the PPL of the test corpus in the target domain. After a close examination of the output, we find that it contains many repeated and overly simple outputs. For example, the system generates many examples of "I love this place" when transferring negative to positive sentiment (see Appendix .1 for examples). It is not surprising that such a trivial output has low perplexity, high accuracy, and low BLEU score. On the other hand, our system obtains reasonably competitive PPL, and our approach achieves the highest accuracy and higher BLEU score than the UNMT baseline.

3.5.5 Further Ablations and Analysis

Parameter Sharing. We also conducted an experiment on the word substitution decipherment task, where we remove parameter sharing (as explained in § 3.3.2) between two directions of transduction distributions, and optimize two encoder-decoder instead. We found that the model only obtained an extremely low BLEU score and failed to generate any meaningful outputs.

Performance vs. Domain Divergence. Figure 3.3 plots the relative improvement of our method over UNMT with respect to accuracy of a naive Bayes' classifier trained to predict the domain of test sentences. Tasks with high classification accuracy likely have more divergent domains. We can see that for decipherment and en-de translation, where the domains have different vocabularies

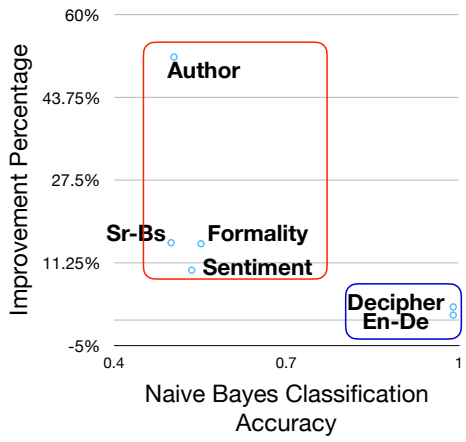


Figure 3.3: Improvement over UNMT vs. classification accuracy.

and thus are easily distinguished, our method yields a smaller gain over UNMT. This likely indicates that the (discrimination) regularization effect of the LM priors is less important or necessary when the two domains are very different.

Why does the proposed model outperform UNMT? Finally, we examine in detail the output of our model and UNMT for the author imitation task. We pick this task because the reference outputs for the test set are provided, aiding analysis. Examples shown in Table 3.3 demonstrate that UNMT tends to make overly large changes to the source so that the original meaning is lost, while our method is better at preserving the content of the source sentence. Next, we quantitatively examine the outputs from UNMT and our method by comparing the F1 measure of words bucketed by their syntactic tags. We use the open-sourced compare-mt tool (Neubig et al., 2019), and the results are shown in Figure 3.4. Our system has outperforms UNMT in all word categories. In particular, our system is much better at generating nouns, which likely leads to better content preservation.

Greedy vs. Sample-based Gradient Approximation. In our experiments, we use greedy decoding from the inference network to approximate the expectation required by ELBO, which is a biased estimator. The main purpose of this approach is to reduce the variance of the gradient estimator during training, especially in the early stages when the variance of sample-based approaches is quite high. As an ablation experiment on the sentiment transfer task we compare greedy and sample-based gradient approximations in terms of both train and test ELBO, as well as task performance corresponding to best test ELBO. After the model is fully trained, we find that the sample-based approximation has low variance. With a single sample, the standard deviation of

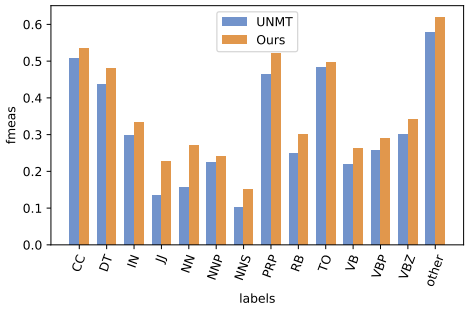


Figure 3.4: Word F1 score by POS tag.

Table 3.3: Examples for author imitation task

Methods	Shakespeare to Modern
Source	Not to his father's .
Reference	Not to his father's house .
UNMT	Not to his brother .
Ours	Not to his father's house .
Source	Send thy man away .
Reference	Send your man away .
UNMT	Send an excellent word .
Ours	Send your man away .
Source	Why should you fall into so deep an O ?
Reference	Why should you fall into so deep a moan ?
UNMT	Why should you carry so nicely , but have your legs ?
Ours	Why should you fall into so deep a sin ?

Table 3.4: Comparison of gradient approximation on the sentiment transfer task.

Method	**train ELBO↑**	**test ELBO↑**	**Acc.**	**BLEU**$_r$	**BLEU**$_s$	**PPL**$_{\mathcal{D}_1}$	**PPL**$_{\mathcal{D}_2}$
Sample-based	-3.51	-3.79	87.90	13.34	33.19	24.55	25.67
Greedy	-2.05	-2.07	87.90	18.67	48.38	27.75	35.61

Table 3.5: Comparison of gradient propagation method on the sentiment transfer task.

Method	**train ELBO↑**	**test ELBO↑**	**Acc.**	**BLEU**$_r$	**BLEU**$_s$	**PPL**$_{\mathcal{D}_1}$	**PPL**$_{\mathcal{D}_2}$
Gumbel Softmax	-2.96	-2.98	81.30	16.17	40.47	22.70	23.88
REINFORCE	-6.07	-6.48	95.10	4.08	9.74	6.31	4.08
Stop Gradient	-2.05	-2.07	87.90	18.67	48.38	27.75	35.61

the EBLO is less than 0.3 across 10 different test repetitions. All final reported ELBO values are all computed with this approach, regardless of whether the greedy approximation was used during training. The reported ELBO values are the evidence lower bound per word. Results are shown in Table 3.4, where the sampling-based training underperforms on both ELBO and task evaluations.

3.5.6 Comparison of gradient propagation methods

As noted above, to stabilize the training process, we stop gradients from propagating to the inference network from the reconstruction loss. Does this approach indeed better optimize the actual probabilistic objective (i.e. ELBO) or only indirectly lead to improved task evaluations? In this section we use sentiment transfer as an example task to compare different methods for propagating gradients and evaluate both ELBO and task evaluations.

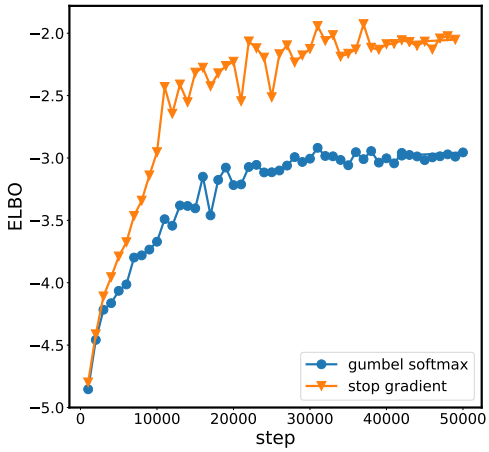


Figure 3.5: ELBO on the validation set v.s. the number training steps.

Specifically, we compare three different methods:

- **Stop Gradient:** The gradients from reconstruction loss are not propagated to the inference network. This is the method we use in all previous experiments.
- **Gumbel Softmax (Jang et al., 2017):** Gradients from the reconstruction loss are propagated to the inference network with the straight-through Gumbel estimator.
- **REINFORCE (Williams, 1987):** Gradients from reconstruction loss are propagated to the inference network with ELBO as a reward function. This method has been used in previous work for semi-supervised sequence generation (Miao and Blunsom, 2016; Yin et al., 2018), but often suffers from instability issues.

We report the train and test ELBO along with task evaluations in Table 3.5, and plot the learning curves on validation set in Figure 3.5.[8] While being much simpler, we show that the stop-gradient trick produces superior *ELBO* over Gumbel Softmax and REINFORCE. This result suggests that stopping gradient helps better optimize the likelihood objective under our probabilistic formulation in comparison with other optimization techniques that propagate gradients, which is counter-intuitive. A likely explanation is that as a gradient estimator, while clearly biased, stop-gradient has substantially reduced variance. In comparison with other techniques that offer reduced bias but extremely high variance when applied to our model class (which involves discrete sequences as latent variables), stop-gradient actually leads to better optimization of our objective because it achieves better balance of bias and variance overall.

[8]We remove REINFORCE from this figure since it is very difficult to stabilize training and obtain reasonable results (e.g. the ELBO value is much worse than others in Table 3.5)

3.6 Summary

We propose a probabilistic generative formulation that unites past work on unsupervised text style transfer. We show that this probabilistic formulation provides a different way to reason about unsupervised objectives in this domain. Our model leads to substantial improvements on five text style transfer tasks, yielding bigger gains when the styles considered are more difficult to distinguish.

Chapter 4

Semi-Supervised NLG with Self-Training

The previous chapter explores label-efficiency of NLG in a fully unsupervised setting where no annotated data is available. In practice, however, we often have a handful of annotated data and abundant unannotated data. In this chapter, we study semi-supervised sequence generation where we have a handful of annotated data and abundant annotated data, and we would like to improve conditional generation performance with the extra unlabeled data. Specifically, this chapter will focus on self-training, a simple technique to train a model on its own predictions. This work is presented in:

- Junxian He*, Jiatao Gu* (equal contribution), Jiajun Shen, and Marc'Aurelio Ranzato. Revisiting Self-Training for Neural Sequence Generation. In *International Conference on Learning Representations (ICLR)*, 2020

4.1 Introduction

Deep neural networks often require large amounts of labeled data to achieve good performance. However, acquiring labels is a costly process, which motivates research on methods that can effectively utilize unlabeled data to improve performance. Towards this goal, semi-supervised learning (Chapelle et al., 2010) methods that take advantage of both labeled and unlabeled data are a natural starting point. In the context of sequence generation problems, semi-supervised approaches have been shown to work well in some cases. For example, back-translation (Sennrich et al., 2016b) makes use of the monolingual data on the target side to improve machine translation systems, latent variable models (Kingma et al., 2014) are employed to incorporate unlabeled source data to facilitate sentence compression (Miao and Blunsom, 2016) or code generation (Yin

et al., 2018).

In this work, we revisit a much older and simpler semi-supervised method, self-training (ST, Scudder (1965)), where a base model trained with labeled data acts as a "teacher" to label the unannotated data, which is then used to augment the original small training set. Then, a "student" model is trained with this new training set to yield the final model. Originally designed for classification problems, common wisdom suggests that this method may be effective only when a good fraction of the predictions on unlabeled samples are correct, otherwise mistakes are going to be reinforced (Zhu and Goldberg, 2009). In the field of natural language processing, some early work have successfully applied self-training to word sense disambiguation (Yarowsky, 1995) and parsing (McClosky et al., 2006; Reichart and Rappoport, 2007; Huang and Harper, 2009).

However, self-training has not been studied extensively when the target output is natural language. This is partially because in language generation applications (e.g. machine translation) hypotheses are often very far from the ground-truth target, especially in low-resource settings. It is natural to ask whether self-training can be useful at all in this case. While Ueffing (2006) and Zhang and Zong (2016) explored self-training in statistical and neural machine translation, only relatively limited gains were reported and, to the best of our knowledge, it is still unclear what makes self-training work. Moreover, Zhang and Zong (2016) did not update the decoder parameters when using pseudo parallel data noting that "*synthetic target parts may negatively influence the decoder model of NMT*".

In this work, we aim to answer two questions: (1) How does self-training perform in sequence generation tasks like machine translation and text summarization? Are "bad" pseudo targets indeed catastrophic for self-training? (2) If self-training helps improving the baseline, what contributes to its success? What are the important ingredients to make it work?

Towards this end, we first evaluate self-training on a small-scale machine translation task and empirically observe significant performance gains over the supervised baseline (§ 4.3.2), then we perform a comprehensive ablation analysis to understand the key factors that contribute to its success (§ 4.3.3). We find that the decoding method to generate pseudo targets accounts for part of the improvement, but more importantly, the perturbation of hidden states – dropout (Hinton et al., 2012) – turns out to be a crucial ingredient to prevent self-training from falling into the same local optimum as the base model, and this is responsible for most of the gains. To understand the role of such noise in self-training, we use a toy experiment to analyze how noise effectively propagates labels to nearby inputs, sometimes helping correct incorrect predictions (§ 4.4.1). Motivated by this analysis, we propose to inject additional noise by perturbing also the input. Comprehensive

Algorithm 1 Classic Self-training

1: Train a base model $f_{\boldsymbol{\theta}}$ on $L = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{l}$
2: **repeat**
3: Apply $f_{\boldsymbol{\theta}}$ to the unlabeled instances U
4: Select a subset $S \subset \{(\boldsymbol{x}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))|\boldsymbol{x} \in U\}$
5: Train a new model $f_{\boldsymbol{\theta}}$ on $S \cup L$
6: **until** convergence or maximum iterations are reached

experiments on machine translation and text summarization tasks demonstrate the effectiveness of noisy self-training.

4.2 Self-training

Formally, in conditional sequence generation tasks like machine translation, we have a parallel dataset $L = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{l}$ and a large unlabeled dataset $U = \{\boldsymbol{x}_j\}_{j=l+1}^{l+u}$, where $|U| > |L|$ in most cases. As shown in Algorithm 1, classic self-training starts from a base model trained with parallel data L, and iteratively applies the current model to obtain predictions on unlabeled instances U, then it incorporates a subset of the *pseudo parallel* data S to update the current model.

There are two key factors: (1) Selection of the subset S. S is usually selected based on some confidence scores (e.g. log probability) (Yarowsky, 1995) but it is also possible for S to be the whole pseudo parallel data (Zhu and Goldberg, 2009). (2) Combination of real and pseudo parallel data. A new model is often trained on the two datasets jointly as in back-translation, but this introduces an additional hyper-parameter to weigh the importance of the parallel data relative to the pseudo data (Edunov et al., 2018). Another way is to treat them separately – first we train the model on pseudo parallel data S, and then fine-tune it on real data L. In our preliminary experiments, we find that the *separate* training strategy with the *whole* pseudo parallel dataset (i.e. $S = \{(\boldsymbol{x}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))|\boldsymbol{x} \in U\}$) produces better or equal performance for neural sequence generation while being simpler. Therefore, in the remainder of this chapter we use this simpler setting. We include quantitative comparison regarding joint training, separate training, and pseudo-parallel data filtering in Appendix 4.5.7, where separate training is able to match (or surpass) the performance of joint training.

In self-training, the unsupervised loss $\mathcal{L}_U$ from unlabeled instances is defined as:

$$\mathcal{L}_U = -\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\mathbb{E}_{\boldsymbol{y}\sim p_{\boldsymbol{\theta}^*}(\boldsymbol{y}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}), \tag{4.1}$$

where $p(\boldsymbol{x})$ is the empirical data distribution approximated with samples from S, $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$ is the conditional distribution defined by the model. $\boldsymbol{\theta}^*$ is the parameter from the last iteration (initially it is set as the parameter of the supervised baseline), and fixed within the current iteration. Eq. 4.1 reveals the connection between self-training and entropy regularization (Grandvalet and Bengio, 2004). In the context of classification, self-training can be understood from the view of entropy regularization (Lee, 2013), which favors a low-density separation between classes, a commonly assumed prior for semi-supervised learning (Chapelle and Zien, 2005).

4.3 A Case Study on Machine Translation

To examine the effectiveness of self-training on neural sequence generation, we start by analyzing a machine translation task. We then perform ablation analysis to understand the contributing factors of the performance gains.

4.3.1 Setup

We work with the standard WMT 2014 English-German dataset consisting of about 3.9 million training sentence pairs after filtering long and imbalanced pairs. Sentences are encoded using 40K byte-pair codes (Sennrich et al., 2016d). As a preliminary experiment, we randomly sample 100K sentences from the training set to train the model and use the remaining English sentences as the unlabeled monolingual data. For convenience, we refer to this dataset as WMT100K. Such synthetic setting allows us to have high-quality unlabeled data to verify the performance of self-training. We train with the Base Transformer architecture (Vaswani et al., 2017) and dropout rate at 0.3. Full training and optimization parameters can be found in § 4.5.1. All experiments throughout this chapter including the transformer implementation are based on the fairseq toolkit (Ott et al., 2019), and all results are in terms of case-sensitive tokenized BLEU (Papineni et al., 2002). We use beam search decoding (beam size 5) to create the pseudo targets and to report BLEU on test set.

4.3.2 Observations

In Figure 4.1, we use green bars to show the result of applying self-training for three iterations. We include both (1) *pseudo-training (PT)*: the first step of self-training where we train a new model (from scratch) using *only* the pseudo parallel data generated by the current model, and

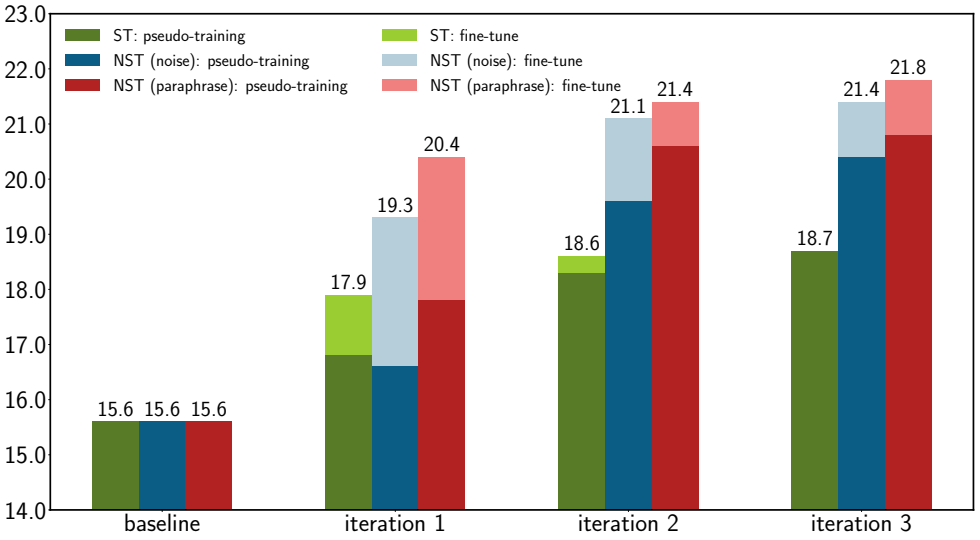


Figure 4.1: BLEU on WMT100K dataset from the supervised baseline and different self-training variants. We plot the results over 3 iterations. "ST" denotes self-training while "NST" denotes noisy self training.

(2) *fine-tuning (FT)*: the fine-tuned system using real parallel data based on the pretrained model from the PT step. Surprisingly, we find that the pseudo-training step at the first iteration is able to improve BLEU even if the model is only trained on its own predictions, and fine-tuning further boosts the performance. The test BLEU keeps improving over the first three iterations, until convergence to outperform the initial baseline by 3 BLEU points.

This behaviour is unexpected because no new information seems to be injected during this iterative process – target sentences of the monolingual data are from the base model's predictions, thus translation errors are likely to remain, if not magnified. This is different from back-translation where new knowledge may originate from an additional backward translation model and real monolingual targets may help the decoder generate more fluent sentences.

One straightforward hypothesis is that the added pseudo-parallel data might implicitly change the training trajectory towards a (somehow) better local optimum, given that we train a new model *from scratch* at each iteration. To rule out this hypothesis, we perform an ablation experiment and initialize $\boldsymbol{\theta}$ from the last iteration (i.e. $\boldsymbol{\theta}^*$). Formally, based on Eq. 4.1 we have:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_U|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} = -\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{y}\sim p_{\boldsymbol{\theta}^*}(\boldsymbol{y}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}\right] = 0, \tag{4.2}$$

because the conditional log likelihood is maximized when $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$ matches the underlying data distribution $p_{\boldsymbol{\theta}^*}(\boldsymbol{y}|\boldsymbol{x})$. Therefore, the parameter $\boldsymbol{\theta}$ should not (at least not significantly) change if we initialize it with $\boldsymbol{\theta}^*$ from the last iteration.

Table 4.1: Test tokenized BLEU on WMT100K. Self-training results are from the first iteration. "Scratch" denotes that the system is initialized randomly and trained from scratch, while "baseline" means it is initialized with the baseline model.

Methods	PT	FT
baseline	–	15.6
ST (scratch)	16.8	17.9
ST (baseline)	16.5	17.5

Table 4.2: Ablation study on WMT100K data. For ST and noisy ST, we initialize the model with the baseline and results are from one single iteration. Dropout is varied only in the PT step, while dropout is always applied in FT step. Different decoding methods refer to the strategy used to create the pseudo target. At test time we use beam search decoding for all models.

Methods	PT	FT
baseline	–	15.6
baseline (w/o dropout)	–	5.2
ST (beam search, w/ dropout)	16.5	17.5
ST (sampling, w/ dropout)	16.1	17.0
ST (beam search, w/o dropout)	15.8	16.3
ST (sampling, w/o dropout)	15.5	16.0
Noisy ST (beam search, w/o dropout)	15.8	17.9
Noisy ST (beam search, w/ dropout)	**16.6**	**19.3**

Table 4.1 shows the comparison results of these two initialization schemes at the first iteration. Surprisingly, continuing training from the baseline model also yields an improvement of 1.9 BLEU points, comparable to initializing from random. While stochastic optimization introduces randomness in the training process, it is startling that continuing training gives such a non-trivial improvement. Next, we investigate the underlying reasons for this.

4.3.3 The Secret Behind Self-Training

To understand why continuing training contradicts Eq. 4.2 and improves translation performance, we examine possible discrepancies between our assumptions and the actual implementation, and formulate two new hypotheses:

H1. Decoding Strategy. According to this hypothesis, the gains come from the use of beam search for decoding unlabeled data. Since our focus is a sequence generation task, we decode $\boldsymbol{y}$ with beam search to approximate the expectation in $\mathbb{E}_{\boldsymbol{y}\sim p_{\boldsymbol{\theta}^*}(\boldsymbol{y}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$, yielding a biased estimate, while sampling decoding would result in an unbiased Monte Carlo estimator. The results in Table 4.2 demonstrate that the performance drops by 0.5 BLEU when we change the decoding strategy to sampling, which implies that beam search does contribute a bit to the performance gains. This phenomenon makes sense intuitively since beam search tends to generate higher-quality pseudo targets than sampling, and the subsequent cross-entropy training might benefit from implicitly learning the decoding process. However, the decoding strategy hypothesis does not fully explain it, as we still observe a gain of 1.4 BLEU points over the baseline from sampling decoding with dropout.

H2. Dropout (Hinton et al., 2012). Eq. 4.1 and Eq. 4.2 implicitly ignore a (seemingly) small difference between the model used to produce the pseudo targets and the model used for training: at test/decoding time the model does not use dropout while at training time dropout noise is injected in the model hidden states. At training time, the model is forced to produce the same (pseudo) targets given the same set of inputs and the same parameter set but various noisy versions of the hidden states. The conjecture is that the additional expectation over dropout noise renders Eq. 4.2 false. To verify this, we remove dropout in the pseudo training step[1]. The results in Table 4.2 indicate that without dropout the performance of beam search decoding drops by 1.2 BLEU, just 0.7 BLEU higher than the baseline. Moreover, the pseudo-training performance of sampling without dropout is almost the same as the baseline, which finally agrees with our intuitions from Eq. 4.2.

In summary, Table 4.2 suggests that beam-search decoding contributes only partially to the performance gains, while the implicit perturbation – dropout – accounts for most of it. However, it is still mysterious why such perturbation results in such large performance gains. If dropout is meant to avoid overfitting and fit the target distribution better in the pseudo-training step, why

[1]During finetuning, we still use dropout.

Table 4.3: Results on the toy sum dataset. For ST and noisy ST, smoothness (↓) and symmetric (↓) results are from the pseudo-training step, while test errors (↓) are from fine-tuning, all at the first iteration.

Methods	smoothness	symmetric	error
baseline	9.1	9.8	7.6
ST	8.2	9.0	6.2
noisy ST	**7.3**	**8.2**	**4.5**

does it bring advantages over the baseline given that the target distribution is from the baseline model itself ? This is the subject of the investigation in the next section.

4.4 Noise in Self-Training

4.4.1 The Role of Noise

One hypothesis as to why noise (perturbation) is beneficial for self-training, is that it enforces local smoothness for this task, that is, semantically similar inputs are mapped to the same or similar targets. Since the assumption that similar input should ideally produce similar target largely holds for most tasks in practice, this smoothing effect of pseudo-training step may provide a favorable regularization for the subsequent fine-tuning step. Unlike standard regularization in supervised training which is local to the real parallel data, self-training smooths the data space covered by the additional and much larger monolingual data.

To verify this hypothesis more easily, we work with the toy task of summing two integers in the range 0 to 99. We concatenate the two integers and view them as a sequence of digits, the sum is also predicted at the digit level, thus this is still a sequence to sequence task. There are 10000 possible data points in the entire space, and we randomly sample 250 instances for training,[2] 100 for validation, 5000 for test, and 4000 as the unlabeled data. Test errors are computed as the absolute difference between the predicted integer and the ground-truth integer. We use an LSTM model to tackle this task. We perform self-training for one iteration on this toy sum dataset and initialize the model with the base model to rule out differences due to the initialization. Setup

[2]We choose 250 instances since we find that 500 training samples already yields perfect performance on this task. However, we want to mimic real seq2seq tasks where the supervised models are often far from perfect.

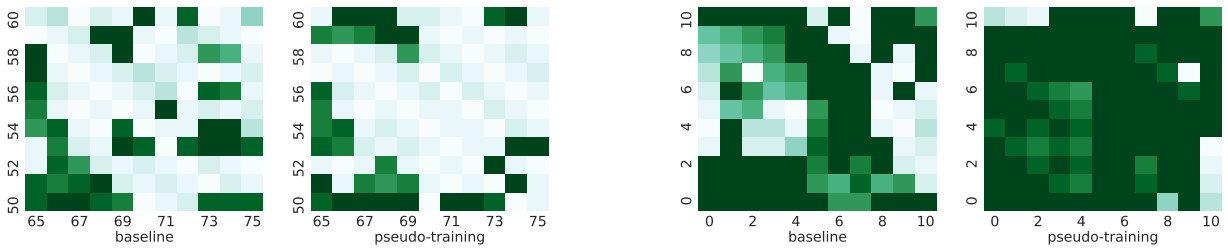


Figure 4.2: Two examples of error heat map on the toy sum dataset that shows the effect of smoothness. The left panel of each composition is from the baseline, and the right one is from the pseudo-training step at the first iteration. x and y axes represent the two input integers. Deeper color represent larger errors.

details are in § 4.5.1.

For any integer pair (x_1, x_2), we measure local smoothness as the standard deviation of the predictions in a 3×3 neighborhood of (x_1, x_2). These values are averaged over all the 10000 points to obtain the overall smoothness. We compare smoothness between baseline and ST pseudo-training in Table 4.3. To demonstrate the effect of smoothing on the fine-tuning step, we also report test errors after fine-tuning. We observe that ST pseudo-training attains better smoothness, which helps reducing test errors in the subsequent fine-tuning step.

One natural question is whether we could further improve performance by encouraging even lower smoothness value, although there is a clear trade-off, as a totally smooth model that outputs a constant value is also a bad predictor. One way to decrease smoothness is by increasing the dropout probability in the pseudo-training step, but a large dropout (like 0.5) makes the model too unstable and slow at converging. Therefore, we consider a simple model-agnostic perturbation process – perturbing the input, which we refer to as *noisy self-training* (noisy ST).

4.4.2 Noisy Self-Training

If we perturb the input during the pseudo-training step, then Eq. 4.1 would be modified to:

$$\mathcal{L}_U = -\mathbb{E}_{\boldsymbol{x}' \sim g(\boldsymbol{x}), \boldsymbol{x} \sim p(\boldsymbol{x})} \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}^*}(\boldsymbol{y}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}'), \tag{4.3}$$

where $g(\boldsymbol{x})$ is a perturbation function. Note that we apply both input perturbation and dropout in the pseudo-training step for noisy ST throughout the chapter, but include ablation analysis in § 4.4.3. We first validate noisy ST in the toy sum task. We shuffle the two integers in the input as the perturbation function. Such perturbation is suitable for this task since it would help the

model learn the commutative law as well. To check that, we also measure the symmetry of the output space. Specifically, for any point (x_1, x_2), we compute $|f(x_1, x_2) - f(x_2, x_1)|$ and average it over all the points. Both smoothness and symmetry values are reported in Table 4.3. While we do not explicitly perturb the input at nearby integers, the shuffling perturbation greatly improves the smoothness metric as well. Furthermore, predictions are more symmetric and test errors are reduced.

In order to illustrate the effect of smoothness, in Figure 4.2 we show two examples of error heat map.[3] When a point with large error is surrounded by points with small errors, the labels might propagate due to smoothing and its error is likely to become smaller, resulting in a "self-correcting" behaviour, as demonstrated in the left example of Figure 4.2. However, the prediction of some points might become worse due to the opposite phenomenon too, as shown in the right example of Figure 4.2. Therefore, the smoothing effect by itself does not guarantee a performance gain in the pseudo-training step, but fine-tuning benefits from it and seems to consistently improve the baseline in all datasets we experiment with.

4.4.3 Observations on Machine Translation

Next, we apply noisy self-training to the more realistic WMT100 translation task. We try two different perturbation functions: (1) Synthetic noise as used in unsupervised MT (Lample et al., 2018b), where the input tokens are randomly dropped, masked, and shuffled. We use the default noising parameters as in unsupervised MT but study the influence of noise level in § 4.5.4. (2) Paraphrase. We translate the source English sentences to German and translate it back to obtain a paraphrase as the perturbation. Figure 4.1 shows the results over three iterations. Noisy ST (NST) greatly outperforms the supervised baseline by over 6 BLEU points and normal ST by 3 BLEU points, while synthetic noise does not exhibit much difference from paraphrasing. Since synthetic noise is much simpler and more general, in the remaining experiments we use synthetic noise unless otherwise specified.

Next, we report an ablation analysis of noisy ST when removing dropout at the pseudo-training step in Table 4.2. Noisy ST without dropout improves the baseline by 2.3 BLEU points and is comparable to normal ST with dropout. When combined together, noisy ST with dropout produces another 1.4 BLEU improvement, indicating that the two perturbations are complementary.

[3]Error heat map for the entire space can be found in Appendix 4.5.8.

4.5 Experiments

Our experiments below are designed to examine whether the noisy self-training is generally useful across different sequence generation tasks and resource settings. To this end, we conduct experiments on two machine translation datasets and one text summarization dataset to test the effectiveness under both high-resource and low-resource settings.

4.5.1 General Setup

We run noisy self-training for three iterations or until performance converges. The model is trained from scratch in the pseudo-training step at each iteration since we found this strategy to work slightly better empirically. In some settings, we also include back-translation (BT, Sennrich et al., 2016b) as a reference point, since this is probably the most successful semi-supervised learning method for machine translation. However, we want to emphasize that BT is not directly comparable to ST since they use different resources (ST utilizes the unlabeled data on the *source* side while BT leverages *target* monolingual data) and use cases. For example, BT is not very effective when we translate English to extremely low-resource languages where there is almost no in-domain target monolingual data available. We follow the practice in (Edunov et al., 2018) to implement BT where we use unrestricted sampling to translate the target data back to the source. Then, we train the real and pseudo parallel data jointly and tune the upsampling ratio of real parallel data.

For all experiments, we optimize with Adam (Kingma and Ba, 2015) using $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 1e-8$. All implementations are based on fairseq (Ott et al., 2019), and we basically use the same learning rate schedule and label smoothing as in fairseq examples to train the transformers.[4] Except for the toy sum dataset which we run on a single GPU and each batch contains 32 examples, all other experiments are run on 8 GPUs with an effective batch size of 33K tokens. All experiments are validated with loss on the validation set. For self-training or noisy self-training, the pseudo-training takes 300K synchronous updates while the fine-tuning step takes 100K steps. We use the downloading and preprocessing scripts in fairseq to obtain the WMT 2014 English-German dataset,[5] which hold out a small fraction of the original training data as the validation set.

The model architecture for the toy sum dataset is a single-layer LSTM with word embedding

[4] https://github.com/pytorch/fairseq/blob/master/examples/translation.
[5] https://github.com/pytorch/fairseq/tree/master/examples/translation.

Table 4.4: Results on two machine translation datasets. For WMT100K, we use the remaining 3.8M English and German sentences from training data as unlabeled data for noisy ST and BT, respectively.

Methods	WMT English-German		FloRes English-Nepali		
	100K (+3.8M mono)	3.9M (+20M mono)	En-Origin	Ne-Origin	Overall
baseline	15.6	28.3	6.7	2.3	4.8
BT	20.5	–	8.2	**4.5**	**6.5**
noisy ST	**21.4**	**29.3**	**8.9**	3.5	**6.5**

size 32, hidden state size 32, and dropout rate 0.3. The model architecture of WMT10K baseline in Figure 4.3a is a single layer LSTM with word embeddings size 256, hidden state size 256, and dropout rate 0.3.

4.5.2 Machine Translation

We test the proposed noisy self-training on a high-resource translation benchmark: WMT14 English-German and a low-resource translation benchmark: FloRes English-Nepali.

- **WMT14 English-German:** In addition to WMT100K, we also report results with all 3.9M training examples. For WMT100K we use the Base Transformer architecture, and the remaining parallel data as the monolingual data. For the full setting, we use the Big Transformer architecture (Vaswani et al., 2017) and randomly sample 20M English sentences from the News Crawl corpus for noisy ST.
- **FloRes English-Nepali:** We evaluate noisy self-training on a low-resource machine translation dataset FloRes (Guzmán et al., 2019) from English (en) to Nepali (ne), where we have 560K training pairs and a very weak supervised system that attains BLEU smaller than 5 points. For this dataset we have 3.6M Nepali monolingual instances in total (for BT) but 68M English Wikipedia sentences.[6] We randomly sample 5M English sentences for noisy ST. We use the same transformer architecture as in (Guzmán et al., 2019).

The overall results are shown in Table 4.4. For almost all cases in both datasets, the noisy ST outperforms the baselines by a large margin ($1 \sim 5$ BLEU scores), and we see that noisy ST still improves the baseline even when this is very weak.

[6] http://www.statmt.org/wmt19/parallel-corpus-filtering.html

Table 4.5: Rouge scores on Gigaword datasets. For the 100K setting we use the remaining 3.7M training data as unlabeled instances for noisy ST and BT. In the 3.8M setting we use 4M unlabeled data for noisy ST. Stared entry (∗) denotes that the system uses a much larger dataset for pretraining.

Methods	**100K (+3.7M mono)**			**640K (+3.2M mono)**			**3.8M (+4M mono)**		
	R1	R2	RL	R1	R2	RL	R1	R2	RL
MASS (Song et al., 2019)*	–	–	–	–	–	–	38.7	19.7	36.0
baseline	30.4	12.4	27.8	35.8	17.0	33.2	37.9	19.0	35.2
BT	32.2	13.8	29.6	**37.3**	**18.4**	**34.6**	–	–	–
noisy ST	**34.1**	**15.6**	**31.4**	36.6	18.2	33.9	**38.6**	**19.5**	**35.9**

Effect of Domain Mismatch. Test sets of the FloRes benchmark were built with mixed original-translationese – some sentences are from English sources and some are from Nepali sources. Intuitively, English monolingual data should be more in-domain with English-origin sentences and Nepali monolingual data should help more for Nepali-origin sentences. To demonstrate this possible domain-mismatch effect, in Table 4.4 we report BLEU on the two different test sets separately.[7] As expected, ST is very effective when the source sentences originate from English.

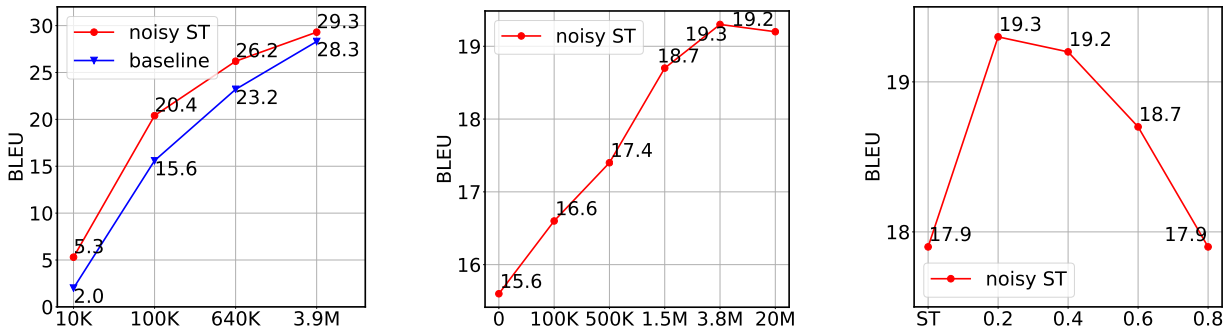
Comparison to Back-Translation. Table 4.4 shows that noisy ST is able to beat BT on WMT100K and on the en-origin test set of FloRes. In contrast, BT is more effective on the ne-origin test set according to BLEU, which is not surprising as the ne-origin test is likely to benefit more from Nepali than English monolingual data.

4.5.3 Text Summarization

We further evaluate noisy self-training on the Gigaword summarization dataset (Rush et al., 2015) that has 3.8M training sentences. We encode the data with 30K byte-pair codes and use the Base Transformer architecture. Similar to the setting of WMT100K, for Gigaword we create two settings where we sample 100K or 640K training examples and use the remaining as unlabeled data to compare with BT. We also consider the setting where all the 3.8M parallel samples are used and we mine in-domain monolingual data by revisiting the original preprocessing procedure[8] and using the ∼4M samples that Rush et al. (2015) disregarded because they had low-quality

[7]Test set split is obtained through personal communication with the authors.

[8]https://github.com/facebookarchive/NAMAS



(a) BLEU v.s. parallel data (b) BLEU v.s. monolingual data (c) BLEU v.s. noise level

Figure 4.3: Analysis of noisy self-training on WMT English-German dataset, demonstrating the effect of parallel data size, monolingual data size, and noise level.

targets. We report ROUGE scores (Lin, 2004) in Table 4.5. Noisy ST consistently outperforms the baseline in all settings, sometimes by a large margin (100K and 640K). It outperforms BT with 100K parallel data but underperforms with 640K parallel data. We conjecture that BT is still effective in this case because the task is still somewhat symmetric as Gigaword mostly contains short sentences and their compressed summaries. Notably, noisy ST in the full setting approaches the performance of state-of-the-art systems which use much larger datasets for pretraining (Song et al., 2019).

4.5.4 Analysis

In this section, we focus on the WMT English-German dataset to examine the effect of three factors on noisy self-training: the size of the parallel dataset, the size of the monolingual dataset, and the noise level. All the noisy ST results are after the fine-tuning step.

Parallel data size. We fix the monolingual data size as 20M from News Crawl dataset, and vary the parallel data size as shown in Figure 4.3a. We use a small LSTM model for 10K, Base Transformer for 100K/640K, and Big Transformer for 3.9M.[9] Noisy ST is repeated for three iterations. We see that in all cases noisy ST is able to improve upon the baseline, while the performance gain is larger for intermediate value of the size of the parallel dataset, as expected.

Monolingual data size. We fix the parallel data size to 100K samples, and use the rest 3.8M English sentences from the parallel data as monolingual data. We sample from this set 100K,

[9]These architectures are selected based on validation loss.

500K, 1.5M, and 3.8M sentences. We also include another point that uses 20M monolingual sentences from a subset of News Crawl dataset. We report performance at the first iteration of noisy ST. Figure 4.3b illustrates that the performance keeps improving as the monolingual data size increases, albeit with diminishing returns.

Noise level. We have shown that noisy ST outperforms ST, but intuitively larger noise must not always be better since at some point it may destroy all the information present in the input. We adopt the WMT100K setting with 100K parallel data and 3.8M monolingual data, and set the word blanking probability in the synthetic noise (Lample et al., 2018b) to 0.2 (default number), 0.4, 0.6, and 0.8. We also include the baseline ST without any synthetic noise. Figure 4.3c demonstrates that performance is quite sensitive to noise level, and that intermediate values work best. It is still unclear how to select the noise level *a priori*, besides the usual hyper-parameter search to maximize BLEU on the validation set.

4.5.5 Noise Process on Parallel Data Only

In this section, we justify whether the proposed noisy self-training process would help the supervised baseline alone without the help of any monolingual data. Similar to the training process on the monolingual data, we first train the model on the noisy source data (pseudo-training), and then fine-tune it on clean parallel data. Different from using monolingual data, there are two variations here in the "pseudo-training" step: we can either train with the fake target predicted by the model as on monolingual data, or train with the real target paired with noisy source. We denote them as "parallel + fake target" and "parallel + real target" respectively, and report the performance on WMT100K in Table 4.6. We use the same synthetic noise as used in previous experiments.

When applying the same noise process to parallel data using fake target, the smoothing effect is not significant since it is restricted into the limited parallel data space, producing marginal improvement over the baseline (+0.4 BLEU). As a comparison, 100K monolingual data produces +1.0 BLEU and the effect is enhanced when we increase the monolingual data to 3.8M, which leads to +3.7 BLEU. Interestingly, pairing the noisy source with real target results in much worse performance than the baseline (-4.3 BLEU), which implies that the use of *fake* target predicted by the model (i.e. distillation) instead of real target is important for the success of noisy self-training, at least in the case where parallel data size is small. Intuitively, the distilled fake target is simpler and relatively easy for the model to fit, but the real target paired with noisy source makes learning

Table 4.6: Results on WMT100K data. All results are from one single iteration. "Parallel + real/fake target" denotes the noise process applied on parallel data but using real/fake target in the "pseudo-training" step. "Mono + fake target" is the normal noisy self-training process described in previous sections.

Methods	PT	FT
parallel baseline	–	15.6
noisy ST, 100K mono + fake target	10.2	16.6
noisy ST, 3.8M mono + fake target	16.6	19.3
noisy ST, 100K parallel + real target	6.7	11.3
noisy ST, 100K parallel + fake target	10.4	16.0

even harder than training with real target and real source, which might lead to a bad starting point for fine-tuning. This issue would be particularly severe when the parallel data size is small, in that case the model would have difficulties to fit real target even with clean source.

4.5.6 Justification of the WMT100K Baseline

We provide more details and evidence to show that our baseline model on WMT100K dataset is trained properly. In all the experiments on WMT100K dataset including baseline and self-training ones, we use Adam optimizer with learning rate 0.0005, which is defaulted in fairseq. We do not use early stop during training but select the best model in terms of the validation loss. We train with 30K update steps for the baseline model and (300K pseudo-training + 100K fine-tuning) update steps for self-training. In both cases we verified that the models are trained sufficiently to fully converge through observing the increase of validation loss. Figure 4.4 shows the validation curve of the baseline model. Note that the model starts to overfit, and we select the model checkpoint at the lowest point. We also varied the learning rate hyperparameter as 0.0002, 0.0005, and 0.001, which produced BLEU score 15.0, 15.6 (reported in the paper), and 15.5 respectively – our baseline model in previous sections obtained the best performance.

4.5.7 Comparison regarding separate training, joint training, and filtering

In the paper we perform self-training with separate pseudo-training and fine-tuning steps and always use all monolingual data. However, there are other variants such as joint training or

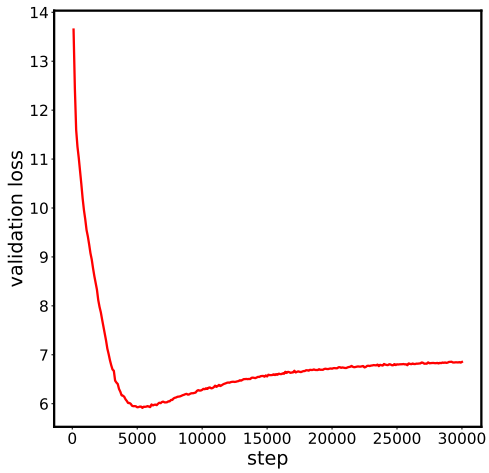


Figure 4.4: Validation loss v.s. number of update steps, for the baseline model on WMT100K dataset.

Table 4.7: Ablation analysis on WMT100K dataset.

Methods	BLEU
baseline	15.6
noisy ST (separate training, all data)	21.8
noisy ST (separate training, filtering)	21.6
noisy ST (joint training, all data)	18.8
noisy ST (joint training, filtering)	20.0

iteratively adding confident examples. Here we compare these variants on WMT100K dataset, noisy self-training uses paraphrase as the perturbation function. For joint training, we tune the upsampling ratio of parallel data just as in back-translation (Edunov et al., 2018). We perform noisy self-training for 3 iterations, and for *filtering* experiments we iteratively use the most confident 2.5M, 3M, and 3.8M monolingual data respectively in these 3 iterations. Table 4.7 shows that the filtering process helps joint training but still underperforms separate-training methods by over 1.5 BLEU points. Within separate training filtering produces comparable results to using all data. Since separate training with all data is the simplest method and produces the best performance, we stick to this version in the paper.

4.5.8 Additional results on the toy sum dataset

We additionally show the error heat maps of the entire data space on the toy sum datasets for the first two iterations. Here the model at pseudo-training step is initialized as the model from last iteration to clearly examine how the decodings change due to injected noise. As shown in Figure 4.5, for each iteration the pseudo-training step smooths the space and fine-tuning step benefits from it and greatly reduces the errors

4.6 Related Work

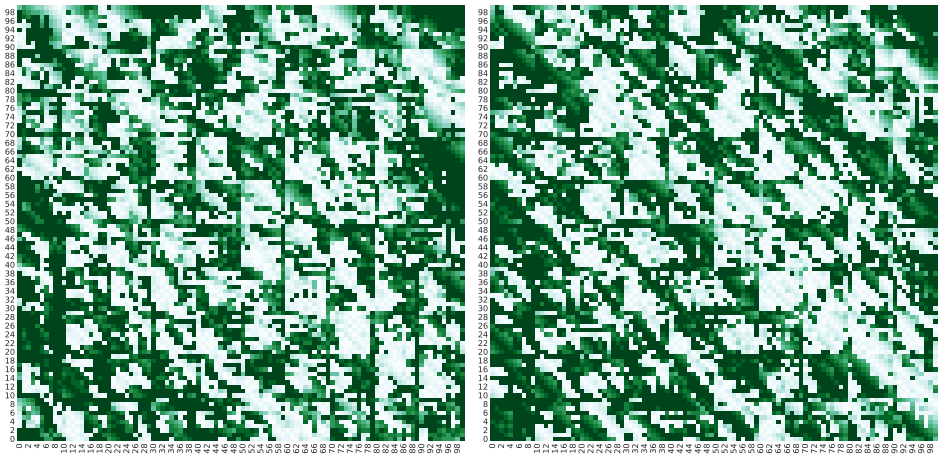
Self-training belongs to a broader class of "pseudo-label" semi-supervised learning approaches. These approaches all learn from pseudo labels assigned to unlabeled data, with different methods on how to assign such labels. For instance, co-training (Blum and Mitchell, 1998) learns models on two independent feature sets of the same data, and assigns confident labels to unlabeled data from one of the models. Co-training reduces modeling bias by taking into account confidence scores from two models. In the same spirit, democratic co-training (Zhou and Goldman, 2004) or tri-training (Zhou and Li, 2005) trains multiple models with different configurations on the same data feature set, and a subset of the models act as teachers for others.

Another line of more recent work perturb the input or feature space of the student's inputs as data augmentation techniques. Self-training with dropout or noisy self-training can be viewed as an instantiation of this. These approaches have been very successful on classification tasks (Rasmus et al., 2015; Miyato et al., 2017; Laine and Aila, 2017; Miyato et al., 2019; Xie et al., 2019) given that a reasonable amount of predictions of unlabeled data (at least the ones with high confidence) are correct, but their effect on language generation tasks is largely unknown and poorly understood because the pseudo language targets are often very different from the ground-truth labels. Recent work on sequence generation employs auxiliary decoders (Clark et al., 2018a) when processing unlabeled data, overall showing rather limited gains.

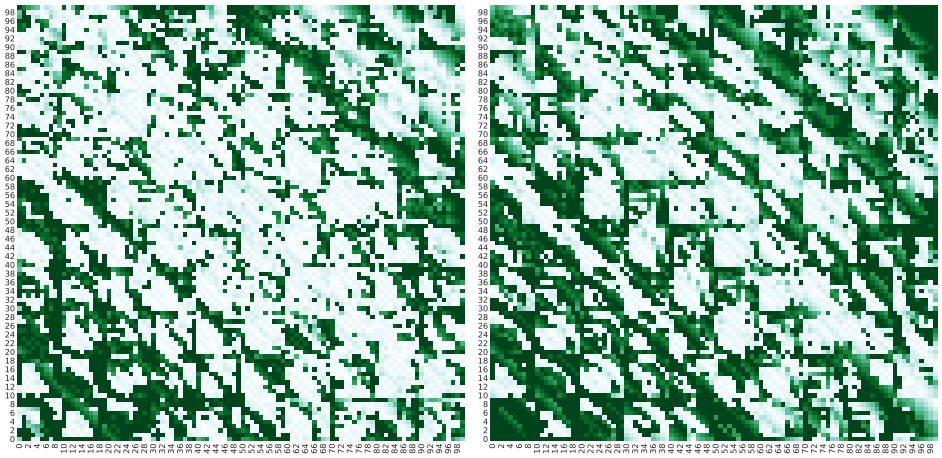
4.7 Summary

In this chapter we revisit self-training for neural sequence generation, and show that it can be an effective method to improve generalization, particularly when labeled data is scarce. Through a comprehensive ablation analysis and synthetic experiments, we identify that noise injected during self-training plays a critical role for its success due to its smoothing effect. To encourage

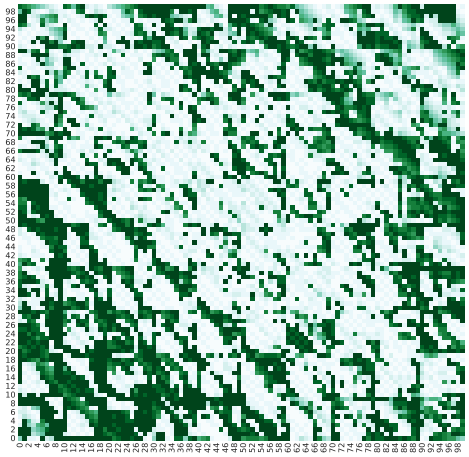
this behaviour, we explicitly perturb the input to obtain a new variant of self-training, dubbed noisy self-training. Experiments on machine translation and text summarization demonstrate the effectiveness of this approach in both low and high resource settings.



(a) Baseline (b) noisy ST (PT, iter=1)



(c) noisy ST (FT, iter=1) (d) noisy ST (PT, iter=2)



(e) noisy ST (FT, iter=2)

Figure 4.5: Error heat maps on the toy sum dataset over the first two iterations. Deeper color represent larger errors.

Chapter 5

Prompt consistency for Zero-Shot Task Generalization

The previous chapter studies noisy self-training as a form of consistency regularization for semi-supervised learning. In this chapter we extend this example-level consistency to prompt-level consistency in an appealing unsupervised setting: zero-shot task generalization. Specifically, a model is trained on annotated data from a set of tasks and is directly applied to other unseen tasks at test time, where we have a few unlabeled examples from the test tasks. Different tasks could have very different data formats traditionally, and thus zero-shot task generalization is considered very challenging. Recently, this becomes possible due to prompting techniques that unify the data format of different tasks as a text generation task. In this chapter, we regularize a consistency loss between synonymous prompts for the same unlabeled example, and achieve gains in an unsupervised manner. This work is presented in:

- Chunting Zhou*, Junxian He* (equal contribution), Xuezhe Ma, Taylor Berg-Kirkpatrick and Graham Neubig. Prompt Consistency for Zero-Shot Task Generalization. Preprint 2022.

5.1 Introduction

While the past decade has demonstrated that pretrained language models (PLMs) are powerful tools for improving generalization from training datasets to test datasets (Devlin et al., 2019b; Liu et al., 2019a; Raffel et al., 2020), more recent work has shown that they can even perform *zero-shot generalization to new tasks* without any annotated examples (Brown et al., 2020; Wei et al., 2022;

Sanh et al., 2022). These systems leverage natural language prompts that specify the task for the model and represent different tasks in a unified format (Liu et al., 2021a). Zero-shot task generalization suggests a path towards generic systems that perform a wide variety of NLP tasks with no annotated examples. However, while enticing conceptually, zero-shot performance often remains relatively low compared to systems trained using even a small amount of task-specific labeled data.

In this chapter, we examine methods to make PLMs better zero-shot learners using unlabeled text. Our work is motivated by consistency training methods that regularize model predictions to be invariant to perturbation (e.g. noise or paraphrasing) of the input examples. Consistency training is widely used in semi-supervised learning literature as an effective technique to utilize unannotated examples (Bachman et al., 2014; Sajjadi et al., 2016; Beyer et al., 2019; Xie et al., 2020a). It is often understood as a type of smoothness regularization or data augmentation (Xie et al., 2020a) and attains strong performance in semi-supervised learning. Instead of example-level consistency, we propose to regularize *prompt consistency*, where a model is regularized to make the same prediction across a diverse set of synonymous task prompts. Prompt consistency regularization makes sense intuitively since PLMs should be robust across synonymous prompts, whereas it is known that model predictions are empirically very sensitive to the wording of the task prompts (Jiang et al., 2020).

Specifically, we design a pairwise distillation loss that encourages consistency between every pair of prompts (Figure 5.1). We refer to our method as *swarm distillation*, and it has the advantage of being fully unsupervised, only requiring unannotated inputs. Notably, unannotated examples are often relatively easy to collect. Drafting several prompts for a task is also far cheaper than annotating labels for each example – in fact, there are already well-designed prompts available for a wide range of NLP tasks (Bach et al., 2022).

Previous work on example-level consistency regularization typically minimizes a consistency loss along with a supervised loss in a semi-supervised setting (Miyato et al., 2019; Xie et al., 2020a). Recently, Elazar et al. (2021) performed experiments optimizing a prompt consistency loss in the context of a relation prediction task, also incorporating a supervised version of the masked language model pretraining objective. In contrast, we (1) optimize a novel prompt consistency loss alone, making our approach completely unsupervised and agnostic to the model's pretraining objective, and (2) experiment on and demonstrate the practicality of such an approach for a broad variety of NLP tasks. Notably, this unsupervised setting poses additional learning challenges: without explicit supervision, the model may suffer from catastrophic forgetting and even exhibit a form of collapse where the model always makes the same predictions for any

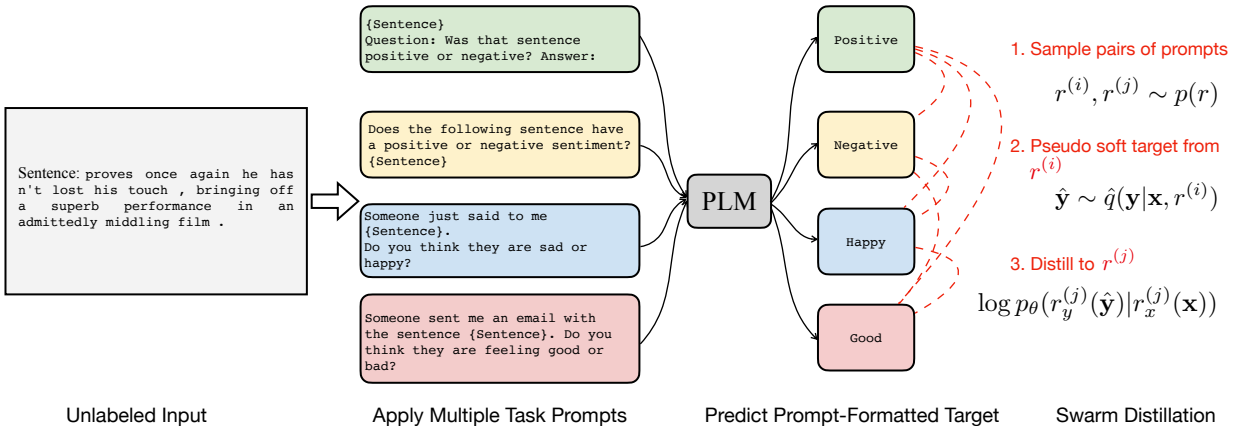


Figure 5.1: An example of the proposed approach in an sentiment classification task. We apply multiple synonymous prompt templates to the unlabeled example, then we regularize the consistency of the predictions from different prompts, through our swarm distillation loss. Note that we are not regularizing the predicted text form $r_y^{(i)}(\mathbf{y})$ to be the same since different prompts have different target templates as shown above – we are actually regularizing the discrete labels $\mathbf{y}$ underneath to be consistent, as detailed in Eq. 5.2.

input. To address this issue, we adopt two simple strategies: (1) we utilize parameter-efficient tuning techniques (Houlsby et al., 2019; He et al., 2022) to only update a small number of extra parameters, naturally mitigating catastrophic forgetting by fixing the original PLM parameters; (2) we propose an unsupervised criterion to select the model checkpoint before it falls into a collapsed local optimum.

In experiments, we build our method on top of a state-of-the-art zero-shot task learner, T0 (Sanh et al., 2022), and validate its performance on 11 datasets from 4 NLP tasks: natural language inference, coreference resolution, word sense disambiguation, and sentence completion. We perform experiments under two secenarios: (1) training the model with unlabeled training data; or (2) tuning the model with unlabeled test inputs directly. In both settings, we show that our swarm distillation method improves the accuracy of the 3B-parameter T0 model on 9 out of 11 datasets by up to 10.6 absolute points. We further scale model size up to 11B parameters, and demonstrate that our approach outperforms the 11B-parameter T0 model on 4 out of 4 datasets. Remarkably, analysis implies that these gains are often possible with only tens of examples, suggesting a small computation overhead.

5.2 Prompt-based Zero-Shot Task Generalization

Given a task where the input is denoted as $\mathbf{x} \in \mathcal{X}$ and the goal is to predict $\mathbf{y} \in \mathcal{Y}$, we focus on the zero-shot task generalization setting: we aim to feed a PLM with $\mathbf{x}$ to predict $\mathbf{y}$, where the PLM is never trained on the specific task to be performed. Zero-shot task generalization goes beyond traditional dataset generalization, as the model must generalize to new functions $f : \mathcal{X} \rightarrow \mathcal{Y}$ as opposed to new input examples, $\mathbf{x}$. Recently, the development of prompting methods has advanced zero-shot task generalization by representing different tasks in a unified format (Liu et al., 2021a), and several prompt-based approaches have attained reasonable zero-shot performance (Brown et al., 2020; Sanh et al., 2022; Wei et al., 2022).

A prompt r consists of an input template r_x, an output template r_y, and metadata to re-format the original $\mathbf{x}$ and $\mathbf{y}$ into new prompt-formatted input and target, $r_x(\mathbf{x})$ and $r_y(\mathbf{y})$. For example, as shown in Figure 5.1, in a sentiment classification task where we must predict positive or negative sentiment of the text, the input includes the field `Sentence` and the target consists of the field `Label`. An input template could be "Does the following sentence have a positive or negative sentiment? `{Sentence}`", and the target template is "`Choices[{label}]`". Here `Choices` is the metadata that is a list containing [`Positive`, `Negative`] to correspond to the digit labels. We note that such metadata is prompt-specific and can differ with different prompts for the same task – for instance, in Figure 5.1 the four prompts actually have three different `Choices` lists; the `Choices` list of the last prompt on the bottom is [`Good`, `Bad`]. In prompt-based approaches the PLM models the conditional probability $q(\mathbf{y}|\mathbf{x}, r)$ through $p_\theta(r_y(\mathbf{y})|r_x(\mathbf{x}))$ where θ denotes the model parameters. In classification tasks where $\mathcal{Y}$ is a finite label set, $q(\mathbf{y}|\mathbf{x}, r)$ is normalized over the possible labels at inference time to predict $\mathbf{y}$:

$$q(\mathbf{y}|\mathbf{x}, r) = \frac{p_\theta(r_y(\mathbf{y})|r_x(\mathbf{x}))}{\sum_{\mathbf{y}' \in \mathcal{Y}} p_\theta(r_y(\mathbf{y}')|r_x(\mathbf{x}))}. \tag{5.1}$$

In generation tasks where $\mathcal{Y}$ is an infinite sequence space, the target template is typically instantiated as the target itself, i.e. $p_\theta(r_y(\mathbf{y})|r_x(\mathbf{x})) = p_\theta(\mathbf{y}|r_x(\mathbf{x}))$, then the output can be directly decoded through sequence decoding approaches. Through designing such prompts for each task, all NLP tasks share the same data format, and models trained on one task may generalize to others.

5.3 Prompt Consistency Training

5.3.1 Problem Definition

In this chapter, we aim to explore unannotated examples to improve prompt-based zero-shot task generalization. Formally, we are given an unlabeled dataset in the task of interest $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, and we assume the dataset has K different prompts, $\{(r_x^{(1)}, r_y^{(1)}), \cdots, (r_x^{(K)}, r_y^{(K)})\}$. Our goal is to utilize these resources and adapt a PLM to predict $r_y(\mathbf{y})$ conditioned on $r_x(\mathbf{x})$. Unlabeled inputs are often available in practice, we consider two such scenarios in the chapter.

First, we consider the case when unannotated examples from a non-test set are available. For many NLP tasks their inputs are plain text such as reviews, documents, or questions and can be easily collected (less so for other NLP tasks, like natural language inference the inputs are paired hypotheses and premises that can be non-trivial to obtain automatically). In this chapter, we test this setting by utilizing the inputs of the training dataset.

Second is the case when unannotated test inputs are available. This is almost always true for any task. We use the test split to mimic the setting. While the limited number of unlabeled examples could potentially limit the effectiveness of some unsupervised learning methods, we show in § 5.4.4 that our method is effective even with tens to hundreds of unlabeled examples.

On the other hand, a diverse set of prompts is not exceedingly difficult to collect practically – drafting prompts for each task is easier than annotating labels for a large number of examples. In fact, the community efforts have pushed out a Public Pool of Prompts (P3)[1] that contains thousands of prompts for hundreds of NLP datasets already (Bach et al., 2022).

5.3.2 The Prompt Consistency Loss

Consistency regularization is a method that creates different views (e.g. paraphrases of text) of the input and regularizes the outputs to be close to each other, and has achieved significant success in semi-supervised learning (Clark et al., 2018b; Xie et al., 2020a,b). While previous methods use an additional module to perturb each example and then optimize example-level consistency, we propose to optimize prompt-level consistency which (1) is conceptually simple, and (2) can mitigate the fact that the predictions of PLMs are typically inconsistent with different prompts for the same task (Jiang et al., 2020; Elazar et al., 2021). Intuitively, we propose to regularize the predictions of different prompts for a given input to be close to each other, using a pairwise

[1] https://github.com/bigscience-workshop/promptsource

distillation loss to draw the predictions from one prompt closer to those from the other. Concretely, we randomly sample a few pairs of prompts and distill the pseudo target $\hat{\mathbf{y}}$ from one prompt $r^{(i)}$ to the other prompt $r^{(j)}$, as illustrated in Figure 5.1. The loss function is defined as:

$$\mathcal{L} = -\mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\mathbb{E}_{r^{(i)},r^{(j)}\sim p(r)} \\ \mathbb{E}_{\hat{\mathbf{y}}\sim\hat{q}(\mathbf{y}|\mathbf{x},r^{(i)})} \log p_\theta(r_y^{(j)}(\hat{\mathbf{y}})|r_x^{(j)}(\mathbf{x})), \tag{5.2}$$

where $p_d(\mathbf{x})$ is the empirical data distribution, $p(r)$ is a uniform distribution over possible prompts in the prompt set, and $\hat{q}(\mathbf{y}|\mathbf{x}, r)$ is the conditional target distribution defined as in Eq. 5.1 but with a stopping gradient operator. We do not propagate gradients to $\hat{q}(\mathbf{y}|\mathbf{x}, r^{(i)})$ following Miyato et al. (2019) and Xie et al. (2020a).[2] Stopping the gradient of one side in a pairwise consistency loss is also shown to help mitigate the collapse issue where all inputs lead to the same predictions (Chen and He, 2021). Different from traditional distillation that distills from a teacher model to a student model (Hinton et al., 2015), or previous consistency training that a single teacher distills to several students (Clark et al., 2018b; Xie et al., 2020a), we perform distillation among a swarm of prompts where each prompt is a teacher and student at the same time, thus we term our method as *swarm distillation*. In our implementation, we approximate the expectation over the paired prompts $(r^{(i)}, r^{(j)})$ with k randomly sampled pairs instead of enumerating all pairs for training efficiency.

Prompt consistency is related to example-level consistency when viewing different prompt-formatted inputs $r_x^{(i)}(\mathbf{x})$ as separated views of the same example, thus our swarm distillation approach shares spirit with previous work on example-level consistency training and can be understood similarly from the perspective of unsupervised data augmentation, smoothness regularization, or label propagation (Xie et al., 2020a). In this chapter, we focus on classification tasks where $\mathcal{Y}$ is a finite label set, while Eq. 5.2 can be directly applied to sequence generation tasks as well with sequence distillation (Kim and Rush, 2016).

Our approach differs from previous consistency training methods which often combine an unsupervised consistency loss with a supervised loss in a semi-supervised setting (Miyato et al., 2019; Clark et al., 2018b; Xie et al., 2020a). Elazar et al. (2021) try to improve prompt consistency for a relation filling task with a pairwise two-sided KL divergence loss, while they also optimize a supervised version of the original PLM objective that turns out to be important. In contrast, our approach minimizes the swarm distillation loss in Eq. 5.2 alone, and therefore is completely unsupervised and agnostic to the pretraining objective. However, this setting also poses challenges in learning, which we discuss next.

[2]Note that $\hat{q}(\mathbf{y}|\mathbf{x}, r)$ still changes as we train the model.

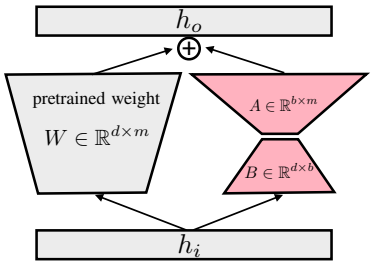


Figure 5.2: A diagram of LoRA in the FFN sublayer. Only the LoRA parameters, A and B, are updated during training while other parameters are fixed.

5.3.3 Training

Being trained without explicit supervision, the PLM may forget what it learns during pretraining since the unsupervised consistency loss is different from the pretraining objective. Also, we note that prompt consistency may be achieved with a trivial solution – if the predictions from each example and each prompt collapse to the same label then maximal consistency among prompts can be reached. To mitigate such catastrophic forgetting and collapse issues, we propose two techniques:

Parameter-efficient tuning: It has recently been observed that updating a small number of added parameters in a PLM is able to achieve comparable performance to tuning all the parameters (Houlsby et al., 2019; Li and Liang, 2021b; Hu et al., 2022; He et al., 2022). Parameter-efficient tuning methods naturally mitigate catastrophic forgetting and collapse through fixing the original PLM parameters. Specifically, we use LoRA (Hu et al., 2022), a low-rank adaptation method for PLMs. As shown in Figure 5.2, LoRA learns a low-rank approximation of the pretrained matrix updates: given a pretrained weight matrix $W \in \mathbb{R}^{d\times m}$, LoRA learns to update it as $W \leftarrow W + \alpha BA$, where $B \in \mathbb{R}^{d\times b}$, $A \in \mathbb{R}^{b\times m}$ are low-rank matrices and α is a hyperparameter, and only B and A are updated during training. $b \ll d$ is referred to as the *bottleneck dimension*. Following He et al. (2022), we apply LoRA to the feed-forward weight matrices of every layer in the pretrained transformer (Vaswani et al., 2017) model. In our preliminary experiments, we found that LoRA is less likely to suffer from collapse, while we still observe collapse sometimes. This motivates us to develop a criterion to select the model checkpoint before the model falls into a collapsed local optimum, which we describe next.

Unsupervised model selection criterion: In supervised learning, model selection is typically performed on a held-out validation set using supervised metrics. However, our zero-shot setting

requires to develop an unsupervised selection criterion. Intuitively, it is straightforward to use a consistency metric as the criterion since we are optimizing towards prompt consistency, but a naive consistency metric would reach its maximum when the model is collapsed. Therefore, we would like to have a metric that encourages consistency but simultaneously penalizes collapse. With that in mind, we focus on Fleiss' kappa (Fleiss, 1971), a commonly used metric to assess the reliability of agreement between a fixed number of raters. In our setting, Fleiss' kappa expresses the extent to which the amount of agreement among prompts exceeds what would be expected if all prompts made their predictions according to the marginalized distribution of labels. This design naturally penalizes collapse and is computing a notion of "relative consistency". Formally, let n_{ij} be the number of prompts that predict the j-th label for the i-th example. There are a total of NK predictions where N is the number of examples and K is the number of prompts. Given an example $\mathbf{x}_i$, the agreement probability p_i is to compute how many prompt pairs are in agreement, divided by the number of all possible pairs:

$$p_i = \frac{1}{K(K-1)} \sum\nolimits_j n_{ij}(n_{ij}-1), \tag{5.3}$$

then p_i is averaged across examples to obtain the "absolute consistency":

$$\bar{P} = \frac{1}{N} \sum\nolimits_{i=1}^{N} p_i. \tag{5.4}$$

It can be seen that $\bar{P}$ is maximized in the case of collapse. However, Fleiss' kappa considers the marginalized distribution of labels: how likely are two prompts consistent if they make predictions randomly according to the marginalized label distribution? This chance probability $\bar{P}_e$ is:

$$\bar{P}_e = \sum\nolimits_j q_j^2, \qquad q_j = \frac{1}{NK} \sum\nolimits_{i=1}^{N} n_{ij}, \tag{5.5}$$

where q_j represents the marginalized distribution of labels, i.e. $p(\mathbf{y} = j)$. $\bar{P}_e$ is large when collapse happens and one label dominates in the entire corpus. Finally, Fleiss' kappa is computed as:

$$\kappa = \frac{\bar{P} - \bar{P}_e}{1 - \bar{P}_e}, \tag{5.6}$$

where $1 - \bar{P}_e$ gives the degree of consistency that is attainable above chance, $\bar{P} - \bar{P}_e$ gives the degree of consistency actually achieved above chance. κ ranges from -1 to 1. Eq. 5.6 naturally penalizes collapse, and in our experiments, we always observe a monotonic decrease of κ when collapse happens. Therefore, we select the model checkpoint after which κ monotonically decreases.[3] We emphasize that we perform validation on the data that the model is trained on

[3]In most of the settings, this criterion is equivalent to using maximal κ as the criterion, except for few cases where the beginning of training exhibits large fluctuations in κ.

and do not require an additional development dataset. We include ablation analysis for both the parameter-efficient tuning and model selection components in Table 5.5 that shows that they are important for the success of our method.

5.4 Experiments

Our experiments below are designed to (1) measure whether swarm distillation is able to improve zero-shot task generalization; and (2) analyze how much resource (number of prompts and unlabeled examples) our method demands.

5.4.1 General Setup

Datasets: Following Sanh et al. (2022), we evaluate our method on 11 NLP datasets across 4 unseen tasks. They are (1) natural language inference: ANLI (Nie et al., 2020) (there are three versions of ANLI with different levels of difficulty, which we denote as ANLI R1/R2/R3), CB (De Marneffe et al., 2019), RTE (Wang et al., 2019a); (2) sentence completion: COPA (Roemmele et al., 2011), HellaSwag (Zellers et al., 2019), Story Cloze (Mostafazadeh et al., 2016); (3) coreference resolution: WSC, Winogrande (Levesque et al., 2012); and (4) word sense disambiguation: WIC (Pilehvar and Camacho-Collados, 2019). We access them using Hugging Face Datasets (Lhoest et al., 2021) and most of them are from the SuperGLUE benchmark (Wang et al., 2019a). All of these datasets are classification-based, predicting a discrete label from a finite set. Each of these datasets has a diverse set of prompts provided by the Public Pool of Prompts (Sanh et al., 2022) The number of prompts ranges from 4 to 15. We present the statistics of the 11 datasets in Table 5.1. For the training-time tuning scenario, we use up to 10,000 data points from the training set for training if the train set contains more than 10,000 data points.

Setup: We build our method on top of the PLM T0 (Sanh et al., 2022). T0 is an adapted version of the pretrained T5 model (Raffel et al., 2020) that is continually trained on multiple tasks with supervised, prompt-formatted examples. T0 outperforms GPT3 (Brown et al., 2020) and demonstrates state-of-the-art performance in zero-shot task generalization. All the tasks that we are studying are not included in T0's training data. We focus our major study on the T0 model version with 3 billion parameters (T0-3B), while we also include results using the largest T0 model with 11 billion parameters (T0-11B) on some datasets, due to the high computational cost of training T0-11B. We tune the hyperparameters (e.g. the optimization hyperparameters) on the

Table 5.1: Statistics of the datasets

Task	Dataset	#train set	#validation set	#labels	#prompts
NLI	RTE	2,490	277	2	10
	CB	250	57	3	15
	ANLI R1	16,946	1000	3	15
	ANLI R2	45,460	1000	3	15
	ANLI R3	100,459	1200	3	15
Compl.	COPA	400	100	2	8
	HellaSwag	39,905	10,042	4	4
	Story Cloze	-	1,871	2	5
Coref.	Winogrande	40,398	1,267	2	5
	WSC	554	104	2	10
WSD	WIC	5,428	637	2	10

RTE dataset with its validation set and fix them for all other datasets. During optimization of Eq. 5.2, we randomly sample a batch of k pairs of prompts where k is the largest number that our GPU memory can fit and accumulate gradients for one update. We use a bottleneck dimension of 1 for LoRA.

Training Details: We use LoRA (Hu et al., 2022) as our parameter-efficient tuning model and set the bottleneck dimension of LoRA weight matrices to be 1 for both 3B and 11B models. We emphasize that the linear mapping matrix B (or A) in LoRA needs to be initialized as a zero matrix to ensure the output distribution after adding LoRA layers is the same as the original PLM before training, otherwise, the zero-shot ability of PLMs would be broken upon initialization and there is no supervision to learn it back. For both models, we set the dropout probability for the the LoRA intermediate representations to be 0.3. Let α denote the scaling factor of LoRA that is used to scale the output of the LoRA layer before adding to the hidden states of the pre-trained model. We set α to be 4 and 2 respectively for the 3B and 11B model. The peak learning rates of the 3B and 11B models are set to be 3e-5 and 5e-5 respectively with a warm-up stage of 100 steps and polynomial learning rate scheduler. We train for a maximum of 1,500 steps. Note that the hyperparameters for the 3B model is tuned on the RTE dataset and used for other datasets. We did not tune the hyperparameters of the 11B model. With respect to implementation details, at each update we first sample one input example $\mathbf{x}$ and apply multiple prompts to reformat it

as $r_x^1(\mathbf{x}), \cdots, r_x^K(\mathbf{x})$, then we perform inference for them and randomly shuffle the predictions. Next we iterate over them with a batch size of 5/10 (3B/11B)[4] and use the shuffled predictions to supervise them to compute the distillation loss, this implements the swarm distillation mechanism in Eq. 5.2 and in fact approximates the expectation over paired prompts with K random pairs. We accumulate the gradients for 16 steps for one update so that each gradient descent is computed from 16 data examples. And we use 1 A40 GPU (45GB memory) to train the 3B model and 4 A40 GPUs with DeepSpeed Zero-2 (Ren et al., 2021) to train the 11B model. In general, training converges pretty fast and takes around 1 - 3 GPU hours for the 3B model and 2 - 6 hours for the 11B model depending on early stop points of different datasets. We use Adam (Kingma and Ba, 2015) as the optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 1e-6$.

5.4.2 Evaluation

Metrics: We use accuracy as the metric for all datasets. We report two different types of accuracy given that we have multiple prompts. The *ensemble accuracy* (Ens.) averages the output distributions of multiple prompts and makes predictions according to it. Ensembling multiple prompts has been explored before and found superior to using a single prompt (Jiang et al., 2020; Qin and Eisner, 2021). The *median accuracy* (Med.) within the set of prompts serves as a proxy for the expected performance when users specify a single prompt and input a prompt-formatted example. As our approach assumes availability of a set of prompts for the downstream task, and it is relatively cheap to craft several prompts for a task, ensemble prediction is the better option given input $\mathbf{x}$, and it does empirically yield higher accuracy overall than the median for both the baseline and our method. Therefore, we will report both numbers but mainly discuss ensemble accuracy. We compute these metrics on the validation split of each dataset. We run the experiments with 3 random seeds and report the mean and standard deviation.

Evaluation scenarios: We provide our methods with different unlabeled sources which lead to two practical scenarios during evaluation: (1) *training-time tuning*: we use the unlabeled training split from the corresponding dataset to train the model. This is similar to traditional settings where training and test data are different; and (2) *test-time tuning* (Sun et al., 2020; Wang et al., 2021): we directly adapt the PLM on the test data. This setting is reasonable, as we will always have access to the test inputs at test time. Intuitively, the unlabeled test sample $\mathbf{x}$ often provides hints about the distribution it was drawn, suggesting that we may update the model before making the

[4]Because the GPU memory sometimes cannot handle all the prompts within one batch.

Table 5.2: Accuracy results on the validation set of 11 NLP datasets based on the T0-3B model. Swarm Distillation (train) and Swarm Distillation (test) use the unlabeled training split and validation split of datasets to train the model respectively, corresponding to training-time and test-time tuning. The Story Cloze dataset does not have a training split and its self distillation results are from tuning on the validation split. We report the mean and std across 3 random runs, and also denote the absolute accuracy change compared to the T0-3B baseline.

		T0-3B		Self Dist. (train)		Swarm Dist. (train)		Swarm Dist. (test)	
Task	Dataset	Ens.	Med.	Ens.	Med.	Ens.	Med.	Ens.	Med.
NLI	RTE	64.6	64.1	64.9$_{\pm 0.2}$	63.8$_{\pm 0.1}$	75.2$_{\pm 0.8}$ ↑10.6	73.9$_{\pm 0.8}$ ↑9.8	75.2$_{\pm 0.2}$ ↑10.6	73.5$_{\pm 0.1}$ ↑9.4
	CB	46.4	50.0	47.0$_{\pm 1.0}$	49.4$_{\pm 2.7}$	47.6$_{\pm 1.0}$ ↑1.2	48.2$_{\pm 0.0}$ ↓1.8	46.4$_{\pm 0.0}$ ↑0.0	48.8$_{\pm 1.0}$ ↓1.2
	ANLI R1	34.6	33.7	36.1$_{\pm 0.1}$	34.7$_{\pm 0.1}$	38.4$_{\pm 0.5}$ ↑3.8	35.7$_{\pm 0.4}$ ↑2.0	38.5$_{\pm 0.3}$ ↑3.9	35.7$_{\pm 0.5}$ ↑2.0
	ANLI R2	33.7	33.4	35.3$_{\pm 0.1}$	33.2$_{\pm 0.2}$	37.9$_{\pm 0.8}$ ↑4.2	36.6$_{\pm 0.5}$ ↑3.2	37.7$_{\pm 0.2}$ ↑4.0	35.4$_{\pm 0.4}$ ↑2.0
	ANLI R3	34.7	33.3	33.1$_{\pm 0.0}$	33.8$_{\pm 0.2}$	34.0$_{\pm 0.3}$ ↓0.7	34.6$_{\pm 0.1}$ ↑1.3	34.1$_{\pm 0.2}$ ↓0.6	33.5$_{\pm 0.0}$ ↑0.2
Compl.	COPA	78.0	79.0	82.3$_{\pm 0.6}$	78.2$_{\pm 0.3}$	82.7$_{\pm 0.6}$ ↑4.7	79.0$_{\pm 0.5}$ ↑0.0	83.0$_{\pm 1.0}$ ↑5.0	79.7$_{\pm 0.6}$ ↑0.7
	HellaSwag	27.8	27.5	32.5$_{\pm 0.2}$	32.7$_{\pm 0.3}$	34.2$_{\pm 0.2}$ ↑6.4	33.4$_{\pm 0.2}$ ↑5.9	33.7$_{\pm 0.6}$ ↑5.9	33.2$_{\pm 0.3}$ ↑5.7
	Story Cloze	86.5	85.1	89.6$_{\pm 0.0}$	88.7$_{\pm 0.0}$	–	–	87.3$_{\pm 0.1}$ ↑0.8	86.9$_{\pm 0.2}$ ↑1.8
Coref.	Wino.	50.9	50.5	51.1$_{\pm 0.1}$	50.7$_{\pm 0.1}$	52.0$_{\pm 0.3}$ ↑1.1	51.4$_{\pm 0.0}$ ↑0.9	52.1$_{\pm 0.3}$ ↑1.2	51.2$_{\pm 0.2}$ ↑0.7
	WSC	69.2	64.4	69.2$_{\pm 0.0}$	64.6$_{\pm 0.3}$	58.3$_{\pm 1.1}$ ↓10.9	59.3$_{\pm 2.0}$ ↓5.1	57.7$_{\pm 0.0}$ ↓11.5	58.8$_{\pm 0.6}$ ↓5.6
WSD	WIC	50.3	50.4	50.3$_{\pm 0.0}$	50.3$_{\pm 0.0}$	55.4$_{\pm 1.1}$ ↑5.1	54.4$_{\pm 0.7}$ ↑4.0	55.5$_{\pm 0.8}$ ↑5.2	54.8$_{\pm 0.5}$ ↑4.4

prediction. This scenario is attractive since it alleviates the common distribution mismatch issue when there is a distribution shift between the training and test data. Compared to training-time tuning, test-time tuning typically uses less unlabeled data in our experiments since it uses the validation split itself. In the major experiments, we focus on the offline test-time tuning where we assume access to the entire test data[5] and train our approach on all test examples, while in §5.4.4 we will discuss the potential for online adaptation where data arrives in a stream.

Baselines: As far as we know, there is no prior work studying unsupervised approaches for this prompt-based task generalization setting, thus T0 is the main baseline that we compare our approach against. However, we still implement an ablation baseline, self distillation, to separate the improvement brought by optimizing prompt consistency and the improvement brought by pseudo-label distillation. Specifically, self distillation minimizes the same loss as in Eq. 5.2 but with $r^{(i)} = r^{(j)}$ – instead of pairwise distillation, the prompt always distills its own prediction

[5]To clarify, test data is not the test split of the dataset, but the data that we evaluate on, i.e. the validation split.

Table 5.3: Accuracy on the validation set based on T0-11B.

	T0-11B		Swarm Dist.	
Dataset	Ens.	Med.	Ens.	Med.
WSC	63.5	62.5	65.4 ↑1.9	62.0 ↓0.5
RTE	83.8	82.0	86.6 ↑2.8	85.0 ↑3.0
HellaSwag	34.4	33.6	45.0 ↑10.6	43.0 ↑9.4
WIC	57.2	56.8	62.1 ↑4.9	60.7 ↑3.9

Table 5.4: Fleiss' kappa on 11 datasets based on T0-3B. Swarm distillation is trained on training split of the respective dataset.

	RTE	CB	ANLI R1	ANLI R2	ANLI R3	COPA	HS	Story.	Wino.	WSC	WIC	Avg.
T0-3B	0.644	0.440	0.221	0.189	0.170	0.586	0.164	0.765	0.396	0.255	0.398	0.384
Swarm Dist.	0.662	0.254	0.145	0.156	0.177	0.699	0.402	0.862	0.509	0.462	0.517	**0.440**

to itself. This baseline can be viewed as a prompt version of self-training, which has proven to effectively utilize unlabeled data and achieved success in various applications (He et al., 2020b; Xie et al., 2020b; Zhang et al., 2020b). For simplicity, we report self distillation results in the training-time tuning setting only.

5.4.3 Results

How well does swarm distillation work? We first compare swarm distillation against the T0-3B baseline. We run our own evaluation using the released T0 weights to obtain the T0 baseline accuracy.[6] As shown in Table 5.2, the ensemble accuracy of swarm distillation exceeds the T0-3B baseline on 9 out of 11 datasets in both training- and test-time tuning settings. Particularly, our approach improves the zero-shot performance on RTE by around 10 absolute points in all cases. Our approach slightly hurts ensemble accuracy of ANLI R3 and median accuracy of CB, but is overall comparable on these two datasets. Compared to self distillation, swarm distilltion outperforms it on 9 out of 11 datasets in terms of ensemble accuracy, by up to 10.3 absolute points. These results further confirm the effectiveness of encouraging prompt consistency. We note that swarm distillation severely fails on WSC with a 10-point accuracy decrease compared to

[6]We are able to reproduce the numbers reported in Sanh et al. (2022), except for COPA where our T0 median number is higher than the originally reported one.

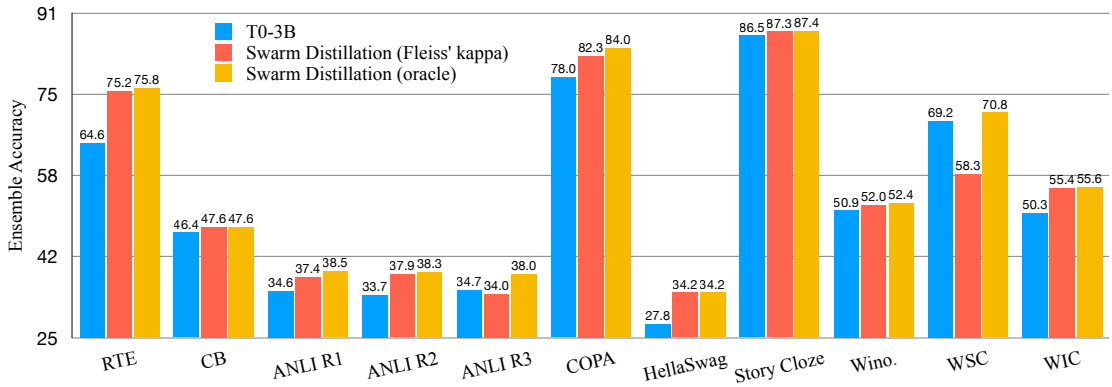


Figure 5.3: Analysis results to compare the model checkpoints selected by the unsupervised criterion Fleiss' kappa with the oracle model checkpoints selected by validation accuracy.

both T0 and self distillation, this is because Fleiss' kappa selects a bad model checkpoint, while our approach actually improves the performance on WSC in the middle of training as we will discuss more in §5.4.4. Although it may be argued that swarm distillation only works when the base PLM can attain reasonable performance in the first place, notably, our approach improves T0-3B greatly on several datasets where T0-3B only shows nearly chance accuracy, such as ANLI R1/R2/R3 (3 labels), HellaSwag (4 labels), Winogrande (2 labels), and WIC (2 labels). In addition, we observe that swarm distillation in the test-time tuning setting performs comparably well to the training-time one despite using much less training data, as shown in Table 5.1. It is worth noting that prompt-based zero-shot task generalization is challenging, for example, T0 with even 11 billion parameters reports a median accuracy of only $\sim$ 40 on ANLI R1/R2/R3, 33.7 on HellaSwag, and 57.2 on WIC (Sanh et al., 2022). These numbers are surely still far from satisfactory, yet we hope to inspire future research to explore prompt-formatted, unlabeled data to build better zero-shot learners.

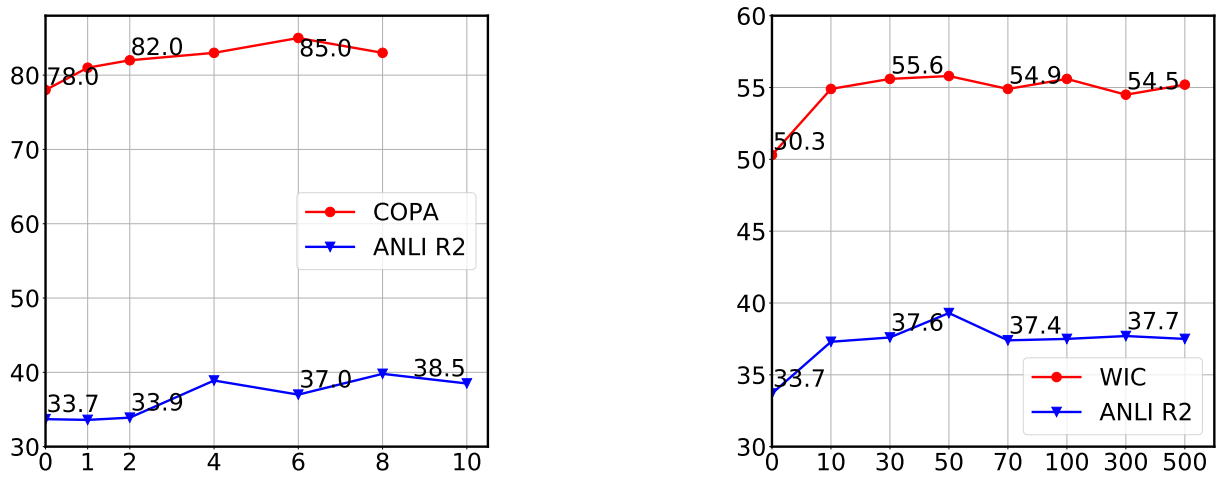
Scaling to 11B parameters: We now evaluate our method based on the largest version of T0 model, T0-11B. T0-11B is a very powerful zero-shot baseline that greatly outperforms GPT3 with 175 billion parameters. Due to the expensive computation to train T0-11B, we use one dataset per task, a total of 4 datasets as our benchmark, and only run with one random seed in the test-time tuning setting. Results are shown in Table 5.3. Swarm distillation outperforms T0-11B on all 4 datasets in terms of ensemble accuracy, and notably, improves the ensemble accuracy on HellaSwag from 34.4 to 45.0 without any annotation. Table 5.2 and Table 5.3 demonstrate the effectiveness of swarm distillation across different model sizes.

5.4.4 Analysis

Are predictions more consistent across different prompts after swarm distillation? We are interested to know whether the gains of swarm distillation are attained together with more consistent predictions across different prompts. To this end, we report Fleiss' kappa, a commonly used metric for group agreement as detailed in §5.3.3. Results are shown in Table 5.4. Fleiss' kappa on 8 out of 11 datasets increases after swarm distillation, which boosts the averaged Fleiss' kappa of T0-3B by 14.6% relatively. This implies that swarm distillation facilitates prompt consistency, and potentially improves the robustness of PLMs to different wording of prompts.

Does the unsupervised criterion select the best model checkpoint? In §5.3.3, we discussed using Fleiss' kappa to select the best model checkpoint for evaluation, here we report the oracle accuracy numbers obtained by selecting the model checkpoint with the best validation accuracy, and compare it to the one selected by Fleiss' kappa. We compare the ensemble accuracy using T0-3B in the training-time tuning setting, with results in Figure 5.3. On most of the datasets, Fleiss' kappa is able to achieve numbers close to the best ones. On all 11 datasets, our oracle number outperforms the T0-3B baseline. In Table 5.2 we show that swarm distillation hurts the performance on WSC a lot, while in Figure 5.3 swarm distillation (oracle) in fact outperforms T0-3B, implying that the issue lies on model selection. Therefore, swarm distillation could potentially work better if an annotated dev set is available or when it is combined with other techniques in few-shot learning settings, where good checkpoints may be selected out more easily.

How many prompts do we need? Our approach requires a diverse set of prompts to regularize prompt consistency. Here we perform ablation experiments to understand the effect of the number of prompts on the performance. We take COPA and ANLI R2 as example datasets which have 8 and 15 prompts, respectively. We then vary the number of available prompts by randomly sampling a subset of prompts before training. We report the ensemble accuracy of swarm distillation (train) in Figure 5.4a. On both COPA and ANLI R2, we observe gains as we increase the number of prompts from 0 (0 means the baseline), yet the performance saturates very quickly and relatively stabilizes when we provide 4 prompts. This implies that swarm distillation is not prompt-hungry and could work well with a small number of prompts. We note the with one prompt here Eq. 5.2 degenerates to a weaker version of self distillation compared to the one in Table 5.2 – self distillation in Table 5.2 utilizes all prompts during training while we assume access to only one prompt here.



(a) Accuracy v.s. #prompts

(b) Accuracy v.s. #examples

Figure 5.4: Ensemble accuracy of swarm distillation on three example datasets, demonstrating the effect of prompt size and unlabeled data size. The PLM is T0-3B.

How many unlabeled examples do we need? We measure the effect of unlabeled data size. Specifically, we randomly sample a subset of examples from the train split for training and report results on the entire validation dataset. Results on WIC and ANLI R2 are shown in Figure 5.4b. Notably, swarm distillation is able to outperform the baselines (#examples=0) by a large margin on both datasets with only 10 unlabeled examples, and the performance starts to saturate quickly afterward. These results suggest that swarm distillation is not data-hungry and works reasonably well with few *unlabeled* examples, allowing swarm distillation to remain as a relatively light approach while typical unsupervised training (e.g. pretraining) often requires a large amount of data and computation. Also, we argue that the phenomenon demonstrated in the results implies that swarm distillation may be applied to the online setting of test-time tuning, where the batches of test data arrive in a stream. Online test-time tuning is a practical setting in real life, and we leave the study of swarm distillation in this setting as future work.

Ablation on LoRA and Model Selection: We report the ablation results on LoRA and unsupervised model selection in Table 5.5. Full fine-tuning hurts the T0-3B performance on all datasets – actually it collapses on almost all the datasets when we check its predictions, which could partially explain the low accuracies. Using LoRA alone is able to improve full fine-tuning generally and outperforms the T0-3B baseline sometimes. Moreoever, we find that unsupervised model selection is very effective to mitigate collapse and greatly improves full fine-tuning results. Finally, combining LoRA and unsupervised model selection gives the best results overall on

Table 5.5: Ablation results on LoRA and unsupervised model selection. Numbers are ensemble accuracy in the training-time tuning setting based on T0-3B. "Full fine-tuning" updates all the model parameters, while "+LoRA" means that we freeze the T0 parameters and only update the LoRA parameters.

	RTE	CB	ANLI R1	ANLI R2	ANLI R3	COPA	HS	Story.	Wino.	WSC	WIC
T0-3B	64.6	46.4	34.6	33.7	34.7	78.0	27.8	86.5	50.9	69.2	50.3
Full fine-tuning	59.6	8.9	33.3	33.5	33.3	54.0	25.7	51.5	50.4	36.5	50.0
+ LoRA	54.0	44.6	33.3	33.3	34.4	82.7	33.6	87.4	52.0	36.5	50.0
+ model selection	75.8	35.7	37.0	33.5	33.3	80.0	32.2	86.5	50.8	71.2	54.6
+ LoRA + model selection	75.2	47.6	38.4	37.9	34.0	82.7	34.2	87.3	52.0	58.3	55.4

these 11 datasets. We clarify that the results in Table 5.5 are only for analysis purpose to better understand the effect of different components of our method, but were not used by us to make design decisions in our preliminary experiments– as stated in §5.3.3, we use LoRA because it collapses less often than full fine-tuning and develop a unsupervised model selection criterion since LoRA still collapses on some datasets. To judge collapse or not, we simply checked the model predictions, to see if the predictions for all the examples are almost the same.

5.4.5 Limitations of Our Work

We propose an unsupervised method for better zero-shot learner. There are two limitations of our work: (1) Because our method is operated in a fully unsupervised manner, there is no supervised development data for us to either select the best model or tune hyperparameters. Thus, we propose to use Fleiss' Kappa as our unsupervised development metric for model selection, which attains decent performance in most cases. However, we also see on very few datasets that the proposed metric fails to select the best checkpoints and hurt the model's performance. As discussed in §5.4.4, our method can be combined with few-shot learning where a few labeled data are provided and we believe this can largely alleviate the issues of model selection in the unsupervised setting. (2) The other limitation and at the same time an advantage of our method is that the proposed method can work well even with 10 unlabeled data points. This certainly makes our method a good candidate for the online setting where batches of test data come in a stream. However, as we discussed in §5.4.4, the performance of our model saturates quickly as we increase the number of unlabeled data, which means the performance of our method cannot scale well with tons of unlabeled data like self-supervised pretraining. As discussed in §5.5, we

expect combining our method with few-shot learning setting / pre-training can lead to further improvements as the supervised signals may guide the model to a better local optimum.

5.5 Summary

In this chapter, we explore prompt consistency regularization to make PLMs better zero-shot learners. Our approach utilizes unlabeled examples to attain zero-shot gains. While we use it in a post-adaptation way to adapt PLMs with the proposed swarm distillation loss alone, our regularization loss could be potentially combined with the pretraining objectives in the pretraining stage, with the multi-prompt training loss (Sanh et al., 2022; Wei et al., 2022), or even with annotated data in few-shot learning settings. Combining the swarm distillation loss with these other losses may easily bypass the model collapse issue since the other loss typically discourages the collapsed local optimum. The potential applications of unsupervised swarm distillation on sequence generation tasks are also worth studying in the future.

Part II

Parameter-Efficient Natural Language Generation

Chapter 6

A Unified View of Parameter-Efficient Transfer Learning

In previous chapters we explore label-efficiency in NLG to mitigate the requirement of annotated examples. In this chapter, we study the efficiency of NLG from another aspect: parameter-efficiency, which focuses on the concerns about the large number of parameters in pretrained models nowadays. In short, we pursue a large portion of parameter-sharing among different NLG tasks, but only a small number of tasks-specific parameters are required. This work is presented in:

- Junxian He*, Chunting Zhou* (equal contribution), Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a Unified View of Parameter-Efficient Transfer Learning. In *International Conference on Learning Representations (ICLR)*, 2022.

6.1 Introduction

Transfer learning from pre-trained language models (PLMs) is now the prevalent paradigm in natural language processing, yielding strong performance on many tasks (Peters et al., 2018; Devlin et al., 2019a; Qiu et al., 2020). The most common way to adapt general-purpose PLMs to downstream tasks is to fine-tune all the model parameters (*full fine-tuning*). However, this results in a separate copy of fine-tuned model parameters for each task, which is prohibitively expensive when serving models that perform a large number of tasks. This issue is particularly salient with the ever-increasing size of PLMs, which now range from hundreds of millions (Radford et al., 2019; Lewis et al., 2020a) to hundreds of billions (Brown et al., 2020) or even trillions of

parameters (Fedus et al., 2021).

To mitigate this issue, a few lightweight alternatives have been proposed to update only a small number of extra parameters while keeping most pretrained parameters frozen. For example, *adapter tuning* (Houlsby et al., 2019) inserts small neural modules called adapters to each layer of the pretrained network and only the adapters are trained at fine-tuning time. Inspired by the success of prompting methods that control PLMs through textual prompts (Brown et al., 2020; Liu et al., 2021a), *prefix tuning* (Li and Liang, 2021a) and *prompt tuning* (Lester et al., 2021) prepend an additional l tunable prefix tokens to the input or hidden layers and only train these soft prompts when fine-tuning on downstream tasks. More recently, Hu et al. (2022) learn low-rank matrices to approximate parameter updates. We illustrate these methods in Figure 6.1. These approaches have all been reported to demonstrate comparable performance to full fine-tuning on different sets of tasks, often through updating less than 1% of the original model parameters. Besides parameter savings, parameter-efficient tuning makes it possible to quickly adapt to new tasks without catastrophic forgetting (Pfeiffer et al., 2021) and often exhibits superior robustness in out-of-distribution evaluation (Li and Liang, 2021a).

However, we contend that the important ingredients that contribute to the success of these parameter-efficient tuning methods are poorly understood, and the connections between them are still unclear. In this chapter, we aim to answer three questions: (1) How are these methods connected? (2) Do these methods share design elements that are essential for their effectiveness, and what are they? (3) Can the effective ingredients of each method be transferred to others to yield more effective variants?

In order to answer these questions, we first derive an alternative form of prefix tuning that reveals prefix tuning's close connections with adapters (§6.3.1). Based on this we then devise a unified framework that frames the aforementioned methods as different ways to modify the hidden representations of frozen PLMs (§6.3.2). Our unified framework decomposes previous methods along a *shared* set of design dimensions, such as the function used to perform the modification, the position in which to impose this modification, and how to integrate the modification. This framework allows us to transfer design choices across approaches to propose new variants such as adapters with multiple heads (§6.3.3). In experiments, we first show that existing parameter-efficient tuning methods still lag behind full fine-tuning on higher-resource and challenging tasks (§6.4.2), as exemplified in Figure 6.2. Then we utilize the unified framework to identify critical design choices and validate the proposed variants empirically (§6.4.3-6.4.6). Our experiments on four NLP benchmarks covering text summarization, machine translation (MT), text classification, and general language understanding, demonstrate that the proposed variant uses less parameters

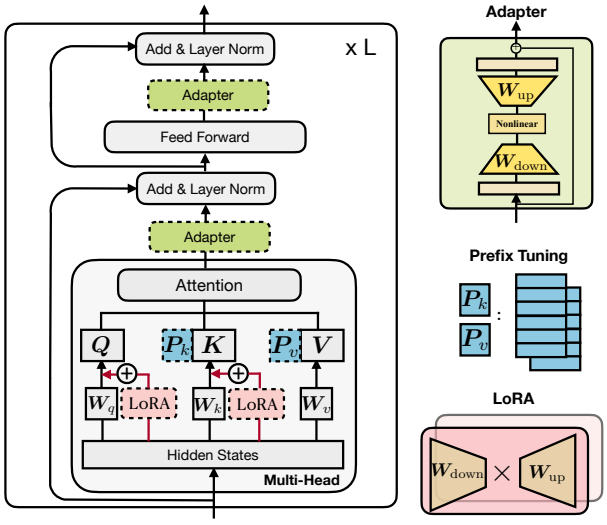


Figure 6.1: Illustration of the transformer architecture and several state-of-the-art parameter-efficient tuning methods. We use blocks with dashed borderlines to represent the added modules by those methods.

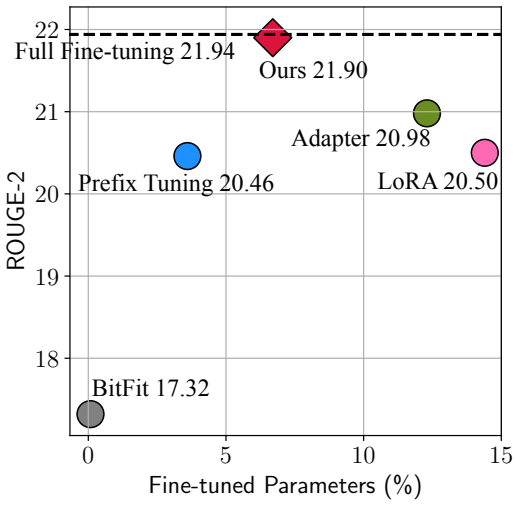


Figure 6.2: Performance of different methods on the XSum (Narayan et al., 2018) summarization task. The number of fine-tuned parameters is relative to the tuned parameters in full fine-tuning.

than existing methods while being more effective, matching full fine-tuning results on all four tasks.

6.2 Preliminaries

6.2.1 Recap of the transformer Architecture

The transformer model (Vaswani et al., 2017) is now the workhorse architecture behind most state-of-the-art PLMs. In this section we recap the equations of this model for completeness. Transformer models are composed of L stacked blocks, where each block (Figure 6.1) contains two types of sub-layers: multi-head self-attention and a fully connected feed-forward network (FFN).[1] The conventional attention function maps queries $\boldsymbol{Q} \in \mathbb{R}^{n \times d_k}$ and key-value pairs $\boldsymbol{K} \in \mathbb{R}^{m \times d_k}, \boldsymbol{V} \in \mathbb{R}^{m \times d_v}$:

$$\text{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V}, \tag{6.1}$$

[1]In an encoder-decoder architecture, the transformer decoder usually has another multi-head cross-attention module between the self-attention and FFN, which we omit here for simplicity.

where n and m are the number of queries and key-value pairs respectively. Multi-head attention performs the attention function in parallel over N_h heads, where each head is separately parameterized by $\boldsymbol{W}_q^{(i)}, \boldsymbol{W}_k^{(i)}, \boldsymbol{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$ to project inputs to queries, keys, and values. Given a sequence of m vectors $\boldsymbol{C} \in \mathbb{R}^{m \times d}$ over which we would like to perform attention and a query vector $\boldsymbol{x} \in \mathbb{R}^d$, multi-head attention (MHA) computes the output on each head and concatenates them:[2]

$$\mathrm{MHA}(\boldsymbol{C}, \boldsymbol{x}) = \mathrm{Concat}(\mathrm{head}_1, \cdots, \mathrm{head}_\mathrm{h})\boldsymbol{W}_o, \ \mathrm{head}_\mathrm{i} = \mathrm{Attn}(\boldsymbol{x}\boldsymbol{W}_q^{(i)}, \boldsymbol{C}\boldsymbol{W}_k^{(i)}, \boldsymbol{C}\boldsymbol{W}_v^{(i)}), \tag{6.2}$$

where $\boldsymbol{W}_o \in \mathbb{R}^{d \times d}$. d is the model dimension, and in MHA d_h is typically set to d/N_h to save parameters, which indicates that each attention head is operating on a lower-dimensional space. The other important sublayer is the fully connected feed-forward network (FFN) which consists of two linear transformations with a ReLU activation function in between:

$$\mathrm{FFN}(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2, \tag{6.3}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_m}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_m \times d}$. Transformers typically use a large d_m, e.g. $d_m = 4d$. Finally, a residual connection is used followed by layer normalization (Ba et al., 2016).

6.2.2 Overview of Previous Parameter-efficient Tuning Methods

Below and in Figure 6.1, we introduce several state-of-the-art parameter-efficient tuning methods. Unless otherwise specified, they only tune the added parameters while the PLM's are frozen.

Adapters (Houlsby et al., 2019): The adapter approach inserts small modules (adapters) between transformer layers. The adapter layer generally uses a down-projection with $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{d \times r}$ to project the input $\boldsymbol{h}$ to a lower-dimensional space specified by bottleneck dimension r, followed by a nonlinear activation function $f(\cdot)$, and a up-projection with $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{r \times d}$. These adapters are surrounded by a residual connection, leading to a final form:

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + f(\boldsymbol{h}\boldsymbol{W}_{\text{down}})\boldsymbol{W}_{\text{up}}. \tag{6.4}$$

Houlsby et al. (2019) places two adapters sequentially within one layer of the transformer, one after the multi-head attention and one after the FFN sub-layer. Pfeiffer et al. (2021) have proposed a more efficient adapter variant that is inserted only after the FFN "add & layer norm" sub-layer.

[2]Below, we sometimes ignore the head index i to simplify notation when there is no confusion.

Prefix Tuning (Li and Liang, 2021a): Inspired by the success of textual prompting methods (Liu et al., 2021a), prefix tuning prepends l tunable prefix vectors to the keys and values of the multi-head attention at every layer. Specifically, two sets of prefix vectors $\boldsymbol{P}_k, \boldsymbol{P}_v \in \mathbb{R}^{l \times d}$ are concatenated with the original key $\boldsymbol{K}$ and value $\boldsymbol{V}$. Then multi-head attention is performed on the new prefixed keys and values. The computation of head_i in Eq. 6.2 becomes:

$$\text{head}_i = \text{Attn}(\boldsymbol{x}\boldsymbol{W}_q^{(i)}, \text{concat}(\boldsymbol{P}_k^{(i)}, \boldsymbol{C}\boldsymbol{W}_k^{(i)}), \text{concat}(\boldsymbol{P}_v^{(i)}, \boldsymbol{C}\boldsymbol{W}^{(i)})), \tag{6.5}$$

$\boldsymbol{P}_k$ and $\boldsymbol{P}_v$ are split into N_h head vectors respectively and $\boldsymbol{P}_k^{(i)}, \boldsymbol{P}_v^{(i)} \in \mathbb{R}^{l \times d/N_h}$ denote the i-th head vector. Prompt-tuning (Lester et al., 2021) simplifies prefix-tuning by only prepending to the input word embeddings in the first layer; similar work also includes P-tuning (Liu et al., 2021b).

LoRA (Hu et al., 2022): LoRA injects trainable low-rank matrices into transformer layers to approximate the weight updates. For a pre-trained weight matrix $\boldsymbol{W} \in \mathbb{R}^{d \times k}$, LoRA represents its update with a low-rank decomposition $\boldsymbol{W} + \Delta W = \boldsymbol{W} + \boldsymbol{W}_{\text{down}}\boldsymbol{W}_{\text{up}}$, where $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{d \times r}, \boldsymbol{W}_{\text{up}} \in \mathbb{R}^{r \times k}$ are tunable parameters. LoRA applies this update to the query and value projection matrices $(\boldsymbol{W}_q, \boldsymbol{W}_v)$ in the multi-head attention sub-layer, as shown in Figure 6.1. For a specific input $\boldsymbol{x}$ to the linear projection in multi-head attention, LoRA modifies the projection output $\boldsymbol{h}$ as:

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + s \cdot \boldsymbol{x}\boldsymbol{W}_{\text{down}}\boldsymbol{W}_{\text{up}}, \tag{6.6}$$

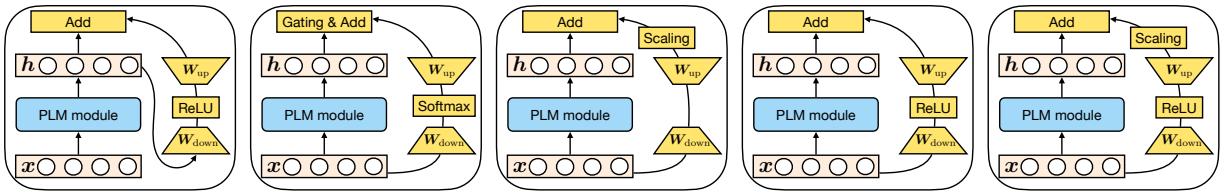
where $s \geq 1$ is a tunable scalar hyperparameter.[3]

Others: Other parameter-efficient tuning methods include BitFit (Ben Zaken et al., 2021), which only fine-tunes bias vectors in the pre-trained model, and diff-pruning (Guo et al., 2021), which learns a sparse parameter update vector.

6.3 Bridging the Gap – A Unified View

In this section we first derive an equivalent form of prefix tuning to establish its connection with adapters. We then propose a unified framework for parameter-efficient tuning that includes several state-of-the-art methods as instantiations.

[3]While Hu et al. (2022) did not mention this tunable hyperparameter in the preprint at the time of this submission (10/05/2021), their latest code at https://github.com/microsoft/LoRA tunes this hyperparameter, and we have verified that this has a significant effect on results.



(a) Adapter (b) Prefix Tuning (c) LoRA (d) Parallel Adapter (e) Scaled PA

Figure 6.3: Graphical illustration of existing methods and the proposed variants. "PLM module" represents a certain sublayer of the PLM (e.g. attention or FFN) that is frozen. "Scaled PA" denotes scaled parallel adapter. We do not include multi-head parallel adapter here to save space.

6.3.1 A Closer Look at Prefix Tuning

Eq. 6.5 describes the mechanism of prefix tuning which changes the attention module through prepending l learnable vectors to the original attention keys and values. Here, we derive an equivalent form of Eq. 6.5 and provide an alternative view of prefix tuning:[4]

$$\begin{aligned}
&\text{head} = \text{Attn}(\boldsymbol{x}\boldsymbol{W}_q, \text{concat}(\boldsymbol{P}_k, \boldsymbol{C}\boldsymbol{W}_k), \text{concat}(\boldsymbol{P}_v, \boldsymbol{C}\boldsymbol{W}_v)) \\
&= \text{softmax}\big(\boldsymbol{x}\boldsymbol{W}_q \text{concat}(\boldsymbol{P}_k, \boldsymbol{C}\boldsymbol{W}_k)^\top\big) \begin{bmatrix} \boldsymbol{P}_v \\ \boldsymbol{C}\boldsymbol{W}_v \end{bmatrix} \\
&= (1-\lambda(\boldsymbol{x}))\text{softmax}(\boldsymbol{x}\boldsymbol{W}_q\boldsymbol{W}_k^\top\boldsymbol{C}^\top)\boldsymbol{C}\boldsymbol{W}_v + \lambda(\boldsymbol{x})\text{softmax}(x\boldsymbol{W}_q\boldsymbol{P}_k^\top)\boldsymbol{P}_v \\
&= (1-\lambda(\boldsymbol{x}))\underbrace{\text{Attn}(\boldsymbol{x}\boldsymbol{W}_q, \boldsymbol{C}\boldsymbol{W}_k, \boldsymbol{C}\boldsymbol{W}_v)}_{\text{standard attention}} + \lambda(\boldsymbol{x})\underbrace{\text{Attn}(\boldsymbol{x}\boldsymbol{W}_q, \boldsymbol{P}_k, \boldsymbol{P}_v)}_{\text{independent of } \boldsymbol{C}},
\end{aligned} \tag{6.7}$$

where $\lambda(\boldsymbol{x})$ is a scalar that represents the sum of normalized attention weights on the prefixes:

$$\lambda(\boldsymbol{x}) = \frac{\sum_i \exp(\boldsymbol{x}\boldsymbol{W}_q\boldsymbol{P}_k^\top)_i}{\sum_i \exp(\boldsymbol{x}\boldsymbol{W}_q\boldsymbol{P}_k^\top)_i + \sum_j \exp(\boldsymbol{x}\boldsymbol{W}_q\boldsymbol{W}_k^\top\boldsymbol{C}^\top)_j}. \tag{6.8}$$

Note that the first term in Eq. 6.7, $\text{Attn}(\boldsymbol{x}\boldsymbol{W}_q, \boldsymbol{C}\boldsymbol{W}_k, \boldsymbol{C}\boldsymbol{W}_v)$, is the original attention without prefixes, whereas the second term is a position-wise modification independent of $\boldsymbol{C}$. Eq. 6.7 gives an alternative view of prefix tuning that essentially applies a position-wise modification to the original head attention output $\boldsymbol{h}$ through linear interpolation:

$$\boldsymbol{h} \leftarrow (1-\lambda(\boldsymbol{x}))\boldsymbol{h} + \lambda(\boldsymbol{x})\Delta\boldsymbol{h}, \quad \Delta\boldsymbol{h} := \text{softmax}(\boldsymbol{x}\boldsymbol{W}_q\boldsymbol{P}_k^\top)\boldsymbol{P}_v. \tag{6.9}$$

The Connection with Adapters: We define $\boldsymbol{W}_1$=$\boldsymbol{W}_q\boldsymbol{P}_k^\top$, $\boldsymbol{W}_2$=$\boldsymbol{P}_v$, f=softmax, and rewrite Eq. 6.9:

$$\boldsymbol{h} \leftarrow (1-\lambda(\boldsymbol{x}))\boldsymbol{h} + \lambda(\boldsymbol{x})f(\boldsymbol{x}\boldsymbol{W}_1)\boldsymbol{W}_2, \tag{6.10}$$

[4]Without loss of generalization, we ignore the softmax scaling factor $\sqrt{d}$ for ease of notation.

which reaches a very similar form to the adapter function in Eq. 6.4, except that prefix tuning is performing weighted addition while the adapter one is unweighted.[5] Figure 6.3b demonstrates the computation graph of prefix tuning from this view, which allows for abstraction of prefix tuning as a plug-in module like adapters. Further, we note that $\boldsymbol{W}_1 \in \mathbb{R}^{d_h \times l}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{l \times d_h}$ are low-rank matrices when l is small, and thus they function similarly to the $\boldsymbol{W}_{\text{down}}$ and $\boldsymbol{W}_{\text{up}}$ matrices in adapters. This view also suggests that the number of prefix vectors, l, plays a similar role to the bottleneck dimension r in adapters: they both represent the rank limitation of computing the modification vector $\Delta\boldsymbol{h}$. Thus we also refer l as the bottleneck dimension. Intuitively, the rank limitation implies that $\Delta\boldsymbol{h}$ is a linear combination of *the same* l (or $\leq l$) basis vectors for any $\boldsymbol{x}$.

The Difference from Adapters: In addition to the gating variable λ, we emphasize three differences between prefix tuning and adapters. (1) As demonstrated in Figure 6.3, prefix tuning uses $\boldsymbol{x}$, the input of the PLM layer, to compute $\Delta\boldsymbol{h}$, while adapters use $\boldsymbol{h}$, the output of the PLM layer. Thus, prefix tuning can be thought of as a "parallel" computation to the PLM layer, whereas the typical adapter is "sequential" computation. (2) Adapters are more flexible with respect to where they are inserted than prefix tuning: adapters typically modify attention or FFN outputs, while prefix tuning only modifies the attention output of each head. Empirically, this makes a large difference as we will show in §6.4.4. (3) Eq. 6.10 applies to each attention head, while adapters are always single-headed, which makes prefix tuning more expressive: head attention is of dimension d/N_h – basically we have full rank updates to each attention head if $l \geq d/N_h$, but we only get full-rank updates to the whole attention output with adapters if $r \geq d$. Notably, prefix tuning is not adding more parameters than adapters when $l = r$.[6] We empirically validate such multi-head influence in §6.4.4.

6.3.2 The Unified Framework

Inspired by the connections between prefix tuning and adapters, we propose a general framework that aims to unify several state-of-the-art parameter-efficient tuning methods. Specifically, we cast them as learning a modification vector $\Delta\boldsymbol{h}$, which is applied to various hidden representations. Formally, we denote the hidden representation to be directly modified as $\boldsymbol{h}$, and the direct input to the PLM sub-module that computes $\boldsymbol{h}$ as $\boldsymbol{x}$ (e.g. $\boldsymbol{h}$ and $\boldsymbol{x}$ can be the attention output and input respectively). To characterize this modification process, we define a set of design dimensions, and

[5] $\boldsymbol{h}$ in adapters and prefix tuning are usually different, as described more below. However, here we mainly discuss the functional form as adapters can, in principle, be inserted at any position.

[6] We will detail in §6.4.1 the number of parameters added of different methods.

Table 6.1: Parameter-efficient tuning methods decomposed along the defined design dimensions. Here, for clarity, we directly write the adapter nonlinear function as ReLU which is commonly used. The bottom part of the table exemplifies new variants by transferring design choices of existing approaches.

Method	$\Delta \boldsymbol{h}$ functional form	insertion form	modified representation	composition function
		Existing Methods		
Prefix Tuning	$\text{softmax}(\boldsymbol{x}\boldsymbol{W}_q\boldsymbol{P}_k^\top)\boldsymbol{P}_v$	parallel	head attn	$\boldsymbol{h} \leftarrow (1-\lambda)\boldsymbol{h} + \lambda\Delta\boldsymbol{h}$
Adapter	$\text{ReLU}(\boldsymbol{h}\boldsymbol{W}_{\text{down}})\boldsymbol{W}_{\text{up}}$	sequential	ffn/attn	$\boldsymbol{h} \leftarrow \boldsymbol{h} + \Delta\boldsymbol{h}$
LoRA	$\boldsymbol{x}\boldsymbol{W}_{\text{down}}\boldsymbol{W}_{\text{up}}$	parallel	attn key/val	$\boldsymbol{h} \leftarrow \boldsymbol{h} + s \cdot \Delta\boldsymbol{h}$
		Proposed Variants		
Parallel adapter	$\text{ReLU}(\boldsymbol{h}\boldsymbol{W}_{\text{down}})\boldsymbol{W}_{\text{up}}$	parallel	ffn/attn	$\boldsymbol{h} \leftarrow \boldsymbol{h} + \Delta\boldsymbol{h}$
Muti-head parallel adapter	$\text{ReLU}(\boldsymbol{h}\boldsymbol{W}_{\text{down}})\boldsymbol{W}_{\text{up}}$	parallel	head attn	$\boldsymbol{h} \leftarrow \boldsymbol{h} + \Delta\boldsymbol{h}$
Scaled parallel adapter	$\text{ReLU}(\boldsymbol{h}\boldsymbol{W}_{\text{down}})\boldsymbol{W}_{\text{up}}$	parallel	ffn/attn	$\boldsymbol{h} \leftarrow \boldsymbol{h} + s \cdot \Delta\boldsymbol{h}$

different methods can be instantiated by varying values along these dimensions. We detail the design dimensions below, and illustrate how adapters, prefix tuning, and LoRA fall along these dimensions in Table 6.1:

Functional Form is the specific function that computes $\Delta\boldsymbol{h}$. We have detailed the functional form for adapters, prefix tuning, and LoRA in Eq. 6.4, 6.6, and 6.10 respectively. The functional forms of all these methods are similar with a `proj_down` $\rightarrow$ `nonlinear` $\rightarrow$ `proj_up` architecture, while "nonlinear" degenerates to the identity function in LoRA.

Modified Representation indicates which hidden representation is directly modified.[7]

Insertion Form is how the added module is inserted into the network. As mentioned in the previous section and shown in Figure 6.3, traditionally adapters are inserted at a position in a sequential manner, where both the input and output are $\boldsymbol{h}$. Prefix tuning and LoRA – although not originally described in this way – turn out to be equivalent to a parallel insertion where $\boldsymbol{x}$ is the input.

Composition Function is how the modified vector $\Delta\boldsymbol{h}$ is composed with the original hidden representation $\boldsymbol{h}$ to form the new hidden representation. For example, adapters perform simple additive composition, prefix tuning uses a gated additive composition as shown in Eq. 6.10, and LoRA scales $\Delta\boldsymbol{h}$ by a constant factor and adds it to the original hidden representation as in Eq. 6.6.

[7]Strictly speaking, all the hidden representations would be indirectly influenced by modifying the ones before them. Here we refer to the position being *directly* modified by the added module.

We note that many other methods not present in Table 6.1 fit into this framework as well. For example, prompt tuning modifies the head attention in the first layer in a way similar to prefix tuning, and various adapter variants (Pfeiffer et al., 2021; Mahabadi et al., 2021) can be represented in a similar way as adapters. Critically, the unified framework allows us to study parameter-efficient tuning methods along these design dimensions, identify the critical design choices, and potentially transfer design elements across approaches, as in the following section.

6.3.3 Transferring Design Elements

Here, and in Figure 6.3, we describe just a few novel methods that can be derived through our unified view above by transferring design elements across methods: (1) *Parallel Adapter* is the variant by transferring the parallel insertion of prefix tuning into adapters. Interestingly, while we motivate the parallel adapter due to its similarity to prefix tuning, concurrent work (Zhu et al., 2021) independently proposed this variant and studied it empirically; (2) *Multi-head Parallel Adapter* is a further step to make adapters more similar to prefix tuning: we apply parallel adapters to modify head attention outputs as prefix tuning. This way the variant improves the capacity for free by utilizing the multi-head projections as we discuss in §6.3.1. (3) *Scaled Parallel Adapter* is the variant by transferring the composition and insertion form of LoRA into adapters, as shown in Figure 6.3e.

Our discussion and formulation so far raise a few questions: Do methods varying the design elements above exhibit distinct properties? Which design dimensions are particularly important? Do the novel methods described above yield better performance? We answer these questions next.

6.4 Experiments

Our experiments below are designed to (1) identify critical design choices through studying parameter-efficient methods along the design dimensions, and (2) evaluate new method variants.

6.4.1 General Setup

Datasets: We study four downstream tasks: (1) XSum (Narayan et al., 2018) is an English single-document summarization dataset where models predict a one-sentence summary given a news article. (2) English to Romanian translation using the WMT 2016 en-ro translation

Table 6.2: Dataset Statistics of the four tasks.

Dataset	#train	#dev	#test
XSum	204,045	113,332	113,334
WMT16 en-ro	610,320	1,999	1,999
MNLI	392,702	9815	9832
SST-2	67,349	872	1,821

dataset (Bojar et al., 2016); (3) MNLI (Williams et al., 2018) is an English natural language inference dataset where models predict whether one sentence entails, contradicts, or is neutral to another. (4) SST2 (Socher et al., 2013) is an English sentiment classification benchmark where models predict whether a sentence's sentiment is positive or negative. We obtain the MNLI and SST2 datasets from the GLUE benchmark (Wang et al., 2019b). The detailed dataset statistics is present in Table 6.2.

Setup: We use $\texttt{BART}_{\texttt{LARGE}}$ (Lewis et al., 2020a) and a multilingual version of it, $\texttt{mBART}_{\texttt{LARGE}}$ (Liu et al., 2020b), as the underlying pretrained models for XSum and en-ro translation respectively, and we use $\texttt{RoBERTa}_{\texttt{BASE}}$ (Liu et al., 2019a) for MNLI and SST2. We vary the bottleneck dimension within $\{1, 30, 200, 512, 1024\}$ if needed.[8] We mainly study adapters, prefix tuning (prefix), and LoRA which greatly outperform bitfit and prompt tuning in our experiments. In the analysis sections (§6.4.3-6.4.5) we insert adapters *either* at the attention or FFN layers for easier analysis, but include the results of inserting at both places in the final comparison (§6.4.6). We re-implement these methods based on their respective public code.[9] We use the huggingface transformers library (Wolf et al., 2020) for our implementation. For MT and classifications tasks, the max token lengths of training data are set to be 150 and 512 respectively. For XSum, we set the max length of source articles to be 512 and the max length of the target summary to be 128. In our summarization experiments, we only use 1600 examples for validation to save time. we test bottleneck dimension 1024 only when the modified representation is FFN, because the training of prefix tuning does not fit into 48GB GPU memory when $l = 1024$. While other methods do not have memory issues, we keep the bottleneck dimension of attention modification at most 512 to have a relatively fair comparison with prefix tuning. For LoRA we always tune its scaling

[8] In some settings we use other values to match the number of added parameters of different methods.

[9] We verify that our re-implementation can reproduce adapter and prefix tuning on XSum, and LoRA on MNLI, by comparing with the results of running the original released code.

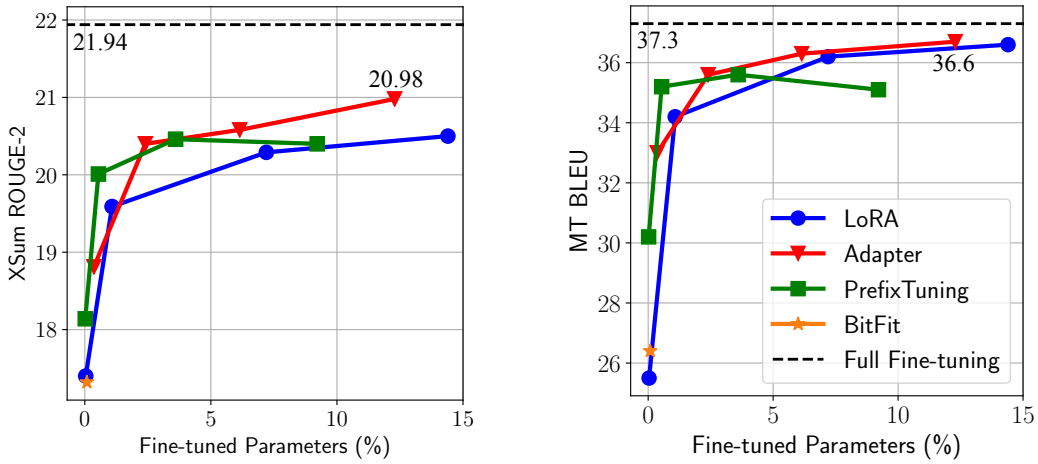


Figure 6.4: Performance of previous state-of-the-art parameter-efficient tuning methods on XSum (left) and en-ro (right).

hyperparameters s on the dev set.

Evaluation: We report ROUGE 1/2/L scores (R-1/2/L, Lin (2004)) on the XSum test set, BLEU scores (Papineni et al., 2002) on the en-ro test set, and accuracy on the MNLI and SST2 dev set. For MNLI and SST2, we take the median of five random runs. We also report the number of tuned parameters relative to that in full fine-tuning (#params).

Number of Tunable Parameters: BART and mBART have an encoder-decoder structure that has three types of attention: encoder self-attention, decoder self-attention, and decoder cross-attention. RoBERTa only has encoder self-attention. For each attention sub-layer, the number of parameters used of each method is: (1) prefix tuning prepends l vectors to the keys and values and uses $2 \times l \times d$ parameters; (2) adapter has $\boldsymbol{W}_{\text{down}}$ and $\boldsymbol{W}_{\text{up}}$ thus uses $2 \times r \times d$ parameters; (3) LoRA employs a pair of $\boldsymbol{W}_{\text{down}}$ and $\boldsymbol{W}_{\text{up}}$ for query and value projections, hence uses $4 \times r \times d$ parameters. For the adapter modification at ffn, it uses $2 \times r \times d$ parameters which is the same as adapter at attention. Therefore, for a specific value of r or l, prefix tuning uses the same number of parameters as adapters, while LoRA uses more parameters.

6.4.2 The Results of Existing Methods

We first overview the results of existing methods on the four tasks. As shown in Figure 6.4 and Table 6.3, while existing methods can achieve competitive (sometimes comparable to full

Table 6.3: Accuracy on the dev set of MNLI and SST2 . All results are from existing methods except MAM Adapter which is proposed in §6.4.6.

Method (# params)	MNLI	SST2
Full-FT (100%)	87.6	94.6
bitfit (0.1 %)	84.7	93.7
Prefix (0.5%)	86.3	94.0
LoRA (0.5%)	87.2	94.2
Adapter (0.9%)	87.3	**94.8**
MAM Adapter (0.5%)	**87.4**	94.2

fine-tuning) performance on MNLI and SST2 by tuning fewer than 1% parameters of PLM's, a large gap is still present if we add 5% parameters in XSum and en-ro. The gap remains significant even though we increase the relative parameter size to >10% (1 R-2 points on XSum and 0.7 BLEU points on en-ro). Even larger gaps have been observed in (Raffel et al., 2020) where adapters are more than 5 absolute points worse than full fine-tuning on SuperGLUE (Wang et al., 2019a) and high-resource MT tasks when #params >10%. This shows that many methods that claimed comparable accuracy to full fine-tuning on standard classification-based benchmarks such as GLUE or lower-resource, relatively simple tasks (Guo et al., 2021; Ben Zaken et al., 2021; Mahabadi et al., 2021) may not generalize to higher-resource and more challenging tasks. We thus advocate for future research on this line to report results on more challenging tasks as well to exhibit a more complete picture of their performance profile. Below, our analysis will mainly focus on the XSum and en-ro datasets to better distinguish different design choices.

6.4.3 Which Insertion Form – Sequential or Parallel?

We first study the insertion form design dimension, comparing the proposed parallel adapter (PA) variant to the conventional sequential adapter (SA) over both the attention (att) and FFN modification. We also include prefix tuning as a reference point. As shown in Table 6.4, prefix tuning, which uses parallel insertion, outperforms attention sequential adapters. Further, the parallel adapter is able to beat sequential adapters in all cases,[10] with PA (ffn) outperforming SA (ffn) by 1.7 R-2 points on XSum and 0.8 BLEU points on en-ro respectively. Given the

[10]More results with different r can be found in Appendix .3, which exhibits similar observations.

Table 6.4: Comparison of different insertion forms for adapters, i.e. sequential adapter (SA) and parallel adapter (PA). We include the results of prefix tuning as a reference point.

Method	# params	XSum (R-1/2/L)	MT (BLEU)
Prefix, l=200	3.6%	43.40/20.46/35.51	35.6
SA (attn), r=200	3.6%	42.01/19.30/34.40	35.3
SA (ffn), r=200	2.4%	43.21/19.98/35.08	35.6
PA (attn), r=200	3.6%	43.58/20.31/35.34	35.6
PA (ffn), r=200	2.4%	**43.93/20.66/35.63**	**36.4**

Table 6.5: Results on en-ro dataset when using 0.1% parameters.

Method	# params	MT (BLEU)
PA (attn), r=200	3.6%	35.6
Prefix, l=200	3.6%	35.6
MH PA (attn), r=200	3.6%	35.8
Prefix, l=30	0.1%	35.2
-gating, l=30	0.1%	34.9
PA (ffn), r=30	0.1%	33.0
PA (attn), r=30	0.1%	33.7
MH PA (attn), r=30	0.1%	**35.3**

superior results of parallel adapters over sequential adapters, we focus on parallel adapter results in following sections.

6.4.4 Which Modified Representation – Attention or FFN?

Setup: We now study the effect of modifying different representations. We mainly compare attention and FFN modification. For easier analysis we categorize methods that modifies any hidden representations in the attention sub-layer (e.g. the head output, query, etc) as modifying the attention module. We compare parallel adapters at attention and FFN and prefix tuning. We also transfer the FFN modification to LoRA to have a LoRA (ffn) variant for a complete comparison. Specifically, we use LoRA to approximate the parameter updates for the FFN weights $\boldsymbol{W}_1 \in \mathbb{R}^{d\times d_m}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{d_m\times d}$. In this case $\boldsymbol{W}_{\text{up}}$ in LoRA for $\boldsymbol{W}_1$ (similar for $\boldsymbol{W}_{\text{down}}$ of $\boldsymbol{W}_2$) would have dimensions of $r \times d_m$, where $d_m = 4d$ as described in §6.2.1. Thus we typically use smaller r for LoRA (ffn) than other methods to match their overall parameter size in later experiments.

Results: As shown in Figure 6.5, *any* method with FFN modification outperforms *all* the methods with attention modification in all cases (the red markers are generally above all the blue ones, the only exception is ffn-PA with 2.4% params), often with fewer parameters. Second, the same method applied at FFN always improves over its counterpart at attention. For example, LoRA (ffn) improves LoRA (attn) by 1 R-2 points on XSum. We also highlight that prefix tuning does not keep improving when we further increase the capacity, which is also observed in (Li

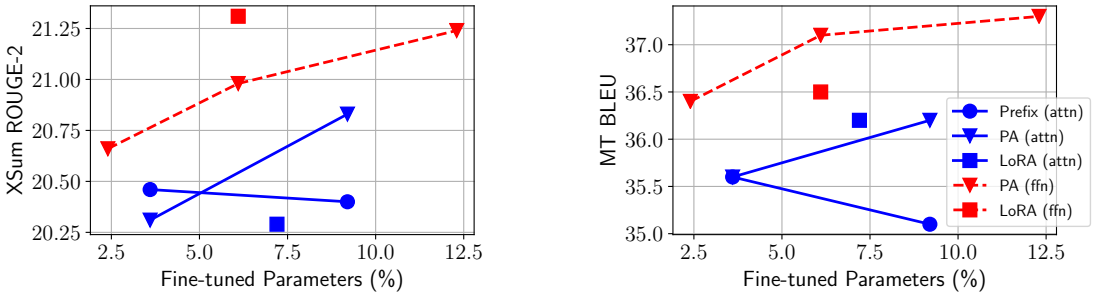


Figure 6.5: Results on XSum (left) and en-ro (right). PA represents parallel adapter. Red and blue markers apply modifications at attention and FFN sub-layers respectively (best viewed in color).

and Liang, 2021a). These results suggest that *FFN modification can utilize the added parameters more effectively than attention, no matter what the functional form or composition function is*. We hypothesize that this is because the FFN learns task-specific textual patterns (Geva et al., 2021), while attention learns pairwise positional interactions which do not require large capacity for adapting to new tasks.

Is the story different when we use 0.1% parameters? In §6.3.1 we reason that prefix tuning is more expressive than adapters (attn), which, however, is not reflected in Figure 6.5. We conjecture that this is because multi-head attention is only superior when the parameter budget is small. To validate this hypothesis, we compare prefix tuning to parallel adapters when they add 0.1% of the pretrained parameters. To ablate the impact of the composition function, we also report the results of removing the gating in prefix tuning as $\boldsymbol{h} + \Delta\boldsymbol{h}$. We include the results of the multi-head parallel adapter variant (MH PA) described in §6.3.3. As shown in Table 6.5, the multi-head methods – prefix tuning and MH PA (attn) – outperform all others by at least 1.6 BLEU points when using 0.1% of the parameters. Surprisingly, reducing l from 200 to 30 only causes 0.4 BLEU loss for prefix tuning while PA (attn) loses 1.9 points. The gating composition function in prefix tuning slightly helps the results by 0.3 points. We highlight that the MH parallel adapter improves the single-headed version by 1.6 points, which again verifies the effectiveness of the multi-head formulation. Comparing at $r = 200$, however, MH PA does not improve much, similarly to prefix tuning.

Combining the results in Figure 6.5 and Table 6.5, we conclude that *modifying head attention shows the best results when the parameter budget is very small, while the FFN can better utilize modification at larger capacities.* This suggests that it may be effective to allocate a larger parameter budget to FFN modification instead of treating attention and FFN equally as in (Houlsby

Table 6.6: Results on XSum when using different composition functions. The modified representation is FFN. The bottleneck dimension $r = 512$ for (Scaled) PA and $r = 102$ for LoRA.

Method (# params)	XSum (R-1/2/LSum)
LoRA (6.1%), s=4	44.59/21.31/36.25
LoRA (6.1%), s=1	44.17/20.83/35.74
PA (6.1%)	44.35/20.98/35.98
Scaled PA (6.1%), s=4	**44.85/21.54/36.58**
Scaled PA (6.1%), trainable s	44.56/21.31/36.29

et al., 2019).

6.4.5 Which Composition Function?

We have presented three composition functions in §6.3.2: simple addition (adapter), gated addition (prefix tuning) and scaled addition (LoRA). As it is unnatural to incorporate the exact gated addition into methods whose functional form does not use softmax, we examine the other two by ablating on LoRA and comparing with the proposed scaled parallel adapter (Scaled PA), we constrain modified representation to be FFN since it is generally more effective as shown in §6.4.4. Table 6.6 reports the results on XSum. We set r as 512 for adapters and 102 for LoRA so that their tuned parameter sizes are the same. We tune s on the dev set. We observe that LoRA with a tuned scaling factor ($s = 4$) performs better than parallel adapter. However, the advantage disappears if we remove the scaling by setting $s = 1$. Through plugging the composition function of LoRA into parallel adapter, the resulted Scaled PA improves the vanilla parallel adapter by 0.56 ROUGE-2 points. We also experiment with a learned scalar which does not give better results. Therefore, we conclude that *the scaling composition function is better than the vanilla additive one while being easily applicable.*

6.4.6 An Effective Integration by Transferring Favorable Design Elements

We first highlight three findings in previous sections: (1) Scaled parallel adapter is the best variant to modify FFN; (2) FFN can better utilize modification at larger capacities; and (3) modifying head attentions like prefix tuning can achieve strong performance with only 0.1% parameters. Inspired by them, we mix and match the favorable designs behind these findings: specifically,

Table 6.7: Comparison of various parameter-efficient tuning methods and the proposed variants. "†" are results copied from (Lewis et al., 2020a) and (Liu et al., 2020b). We could not reproduce exactly the same full fine-tuning numbers with the same hyperparameters or even searching them. The reason may be the different libraries which the training code is based on – full fine-tuning is very sensitive to training hyperparameters.

Method	# params	XSum (R-1/2/L)	MT (BLEU)
Full fine-tuning†	100%	45.14/22.27/37.25	37.7
Full fine-tuning (our run)	100%	44.81/21.94/36.83	37.3
Bitfit (Ben Zaken et al., 2021)	0.1%	40.64/17.32/32.19	26.4
Prompt tuning (Lester et al., 2021)	0.1%	38.91/15.98/30.83	21.0
Prefix tuning (Li and Liang, 2021a), l=200	3.6%	43.40/20.46/35.51	35.6
Pfeiffer adapter (Pfeiffer et al., 2021), r=600	7.2%	43.92/20.80/35.82	36.8
LoRA (ffn), r=102	7.2%	44.59/21.31/36.25	36.5
Parallel adapter (PA, ffn), r=1024	12.3%	44.53/21.24/36.23	37.3
PA (attn, r=30) + PA (ffn, r=512)	6.7%	43.95/20.86/35.93	37.1
Prefix tuning (attn, l=30) + LoRA (ffn, r=102)	6.7%	44.88/21.75/36.71	36.9
MAM Adapter (our variant, l=30, r=512)	6.7%	**45.12/21.90/36.91**	**37.5**

we use prefix tuning with a small bottleneck dimension ($l = 30$) at the attention sub-layers and allocate more parameter budgets to modify FFN representation using the scaled parallel adapter ($r = 512$). Since prefix tuning can be viewed as a form of adapter in our unified framework, we name this variant as *Mix-And-Match adapter (MAM Adapter)*. In Table 6.7, we compare MAM adapter with various parameter-efficient tuning methods. For completeness, we also present results of other combination versions in Table 6.7: using parallel adapters at both attention and FFN layers and combining prefix tuning (attn) with LoRA (ffn) – both of these combined versions can improve over their respective prototypes. However, MAM Adapter achieves the best performance on both tasks and is able to match the results of our full fine-tuning by only updating 6.7% of the pre-trained parameters. In Table 6.3, we present the results of MAM Adapter on MNLI and SST2 as well, where MAM Adapter achieves comparable results to full fine-tuning by adding only 0.5% of pretrained parameters.

6.5 Summary

We provide a unified framework for several performant parameter-tuning methods, which enables us to instantiate a more effective model that matches the performance of full fine-tuning method through transferring techniques across approaches. We hope our work can provide insights and guidance for future research on parameter-efficient tuning and we also suggest these methods to perform experiments on tasks of varying resources and properties to exhibit a more complete view.

Part III

Efficient Retrieval-Augmented Natural Language Generation

Chapter 7

Learning Sparse Prototypes for Text Generation

In addition to label-efficiency and parameter-efficiency explored in previous chapters, in this chapter, we study compute-efficiency in a specific model class, retrieval-augmented language models. Retrieval-augmented language models retrieves from an external datastore such as Wikipedia or simply the training dataset, and utilize the retrieved information to assist in prediction. One main limitation of this model class is the space and time overhead at inference time due to the potentially big datastore and the retrieval process. This chapter focuses on a specific model in this class, the latent prototype language model (Guu et al., 2018), and we propose a new latent prototype model to automatically learn a sparse distribution on the datastore to prune the datastore, and thus reduce the space and retrieval time cost. This work is presented in:

- Junxian He, Taylor Berg-Kirkpatrick, Graham Neubig. In *Proceedings of the Thirty-fourth Conference on Neural Information Processing Systems (NeurIPS)*, 2020

7.1 Introduction

Language models (LMs) predict a probability distribution over text, and are a fundamental technology widely studied in the natural language processing (NLP) community (Bengio et al., 2003; Merity et al., 2018; Dai et al., 2019). Modern LMs are almost exclusively based on *parametric* recurrent (Mikolov et al., 2010; Sundermeyer et al., 2012) or self-attentional (Vaswani et al., 2017; Al-Rfou et al., 2019) neural networks. These models are of interest scientifically as one of the purest tests of our ability to capture the intricacies of human language mathematically

(Linzen et al., 2016; Kuncoro et al., 2017; Petroni et al., 2019). They also have broad downstream applications in generating text in systems such as machine translation (Bahdanau et al., 2015), summarization (Rush et al., 2015), or dialog generation (Sordoni et al., 2015b), as well as in the unsupervised representation learners that now power many applications in NLP (Devlin et al., 2018; Liu et al., 2019a; Yang et al., 2019).

However, there has been a recent move towards *non-parametric* neural LMs (Guu et al., 2018; Khandelwal et al., 2020) that generate sentences by first selecting examples from an external datastore. For instance, Khandelwal et al. (2020) model the token-level probability at test time by interpolating the language model with a kNN distribution from the nearest context-token pairs in the datastore, while Guu et al. (2018) store external memories on sentence level and feature a prototype-then-edit process of (1) selecting a *prototype* sentence from a the prototype datastore, and (2) editing this prototype to the final desired output. In this chapter, we focus on the prototype-then-edit model family which is a lot lighter relatively in terms of memory and time cost at test time.

Intuitively, these non-parametric LMs are attractive because they help remove some of the pressure on the parametric model to memorize the entirety of the language it must model. These intuitive advantages are also borne out in superior performance on language modeling tasks (Guu et al., 2018; Khandelwal et al., 2020), as well as down-stream applications such as dialogue response generation (Weston et al., 2018; Wu et al., 2019), machine translation (Gu et al., 2018b; Bapna and Firat, 2019; Khandelwal et al., 2021), and code generation (Hashimoto et al., 2018; Hayati et al., 2018). In addition, the prototypes and continuous representations of the edits in prototype-based models lend an element of interpretability to the modeling process. On the downside, however, previous prototype-driven generation methods usually need to store and index a large prototype candidate pool (in general the whole training dataset), leading to significant issues with memory and speed efficiency at test time.

In this chapter, we hypothesize that, in fact, a *small* set of prototypes is sufficient to achieve the great majority of the gains afforded by such non-parametric models. Intuitively, in a large corpus many sentences look very similar and may be represented by minor transformations of a single prototype sentence. For example, the sentence "I ordered a burger with fries" can serve as the prototype for data samples with the form "I ordered [NOUN PHRASE] with [NOUN PHRASE]". This is evidenced by Guu et al. (2018)'s observation that 70% of the test set in the Yelp restaurant review corpus (Yelp, 2017) is within word-token Jaccard distance 0.5 of one training sentence.

To take advantage of this intuition, we propose a novel generative model that samples prototypes from a *latent* prototype distribution, which itself is sampled from a symmetric Dirichlet

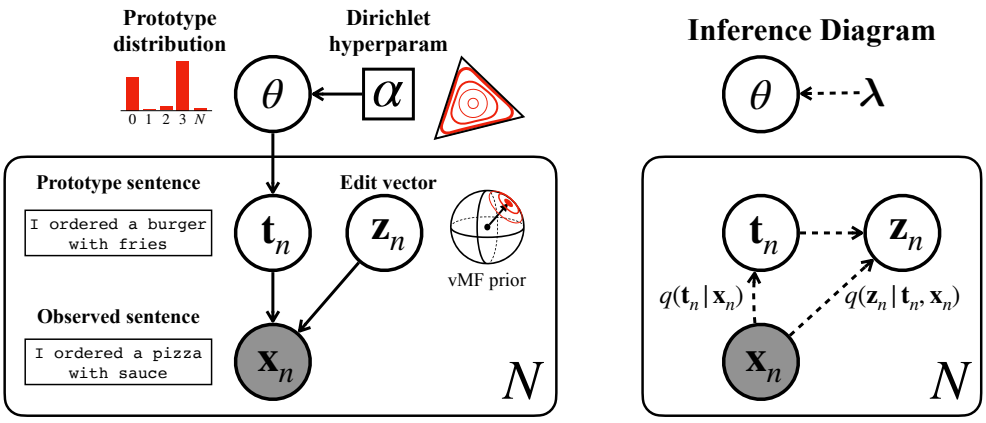


Figure 7.1: **Left:** the proposed generative model to generate data by editing prototypes. Shaded circles denote the observed variables and unshaded denote the latents. Prototypes are sampled from a sparse prototype distribution which itself is a random variable sampled from a Dirichlet prior distribution. **Right:** the inference diagram of the model, with $q(\mathbf{t}_n|\mathbf{x}_n)$ being the prototype retriever and $q(\mathbf{z}_n|\mathbf{t}_n, \mathbf{x}_n)$ being the inverse editor.

prior, as shown in Figure 7.1 (Section 7.3.1). The Dirichlet prior with appropriate hyperparameters is able to encourage a *sparse* prototype selection distribution, allowing us to reduce the prototype support set at test time to greatly improve efficiency. Moreover, we utilize amortized variational inference (Kingma and Welling, 2014) to train our model, which introduces a learnable *prototype retriever* to identify prototypes useful for generating each sentence (Section 7.3.2). This is different from (Guu et al., 2018) where prototypes for each sentence are fixed before training through edit distance heuristics.

We evaluate our approach on the MSCOCO (Lin et al., 2014) and Yelp restaurant review (Yelp, 2017) corpora. Our method is able to improve perplexity over the neural language model baseline by up to 14 points and previous neural editor model by 6 points while achieving over 1000x memory savings and a 1000x speedup at test time (Section 7.4.2). Interestingly, we find that the learned prototypes are able to represent different features when varying sparsity levels – a strong sparsity prior forces the model to share prototypes and the induced prototypes turn out to represent more generic features (e.g. syntactic form of the sentence). On the text generation side, our model is able to generate sentences that resemble the given prototype while allowing for smooth interpolation on the edit space as well (Section 7.4.3).

7.2 Background

The prototype-then-edit framework defines a non-parametric way to augment text generation models. Formally, given a corpus $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$,[1] the model generates each observed example $\mathbf{x}_n$ by: (1) retrieving a prototype sequence $\mathbf{t}_n$, (2) generating a continuous edit representation $\mathbf{z}_n$, and (3) generating $\mathbf{x}_n$ conditioned on $\mathbf{t}_n$ and $\mathbf{z}_n$. These intermediate prototype and edit representation variables can depend on extra context in conditional generation tasks (Hodosh et al., 2013; Gu et al., 2018b), or are randomly sampled in unconditioned language modeling. In this chapter, we focus on the latter, but our methods could likely be applied to the former as well.

For unconditioned language modeling, Guu et al. (2018) define the data likelihood as:

$$p(\mathbf{X}) = \prod_n \sum_{\mathbf{t}_n} \int_{\mathbf{z}_n} p(\mathbf{z}_n) p(\mathbf{t}_n) p_{\boldsymbol{\gamma}}(\mathbf{x}_n | \mathbf{t}_n, \mathbf{z}_n) d\mathbf{z}_n, \tag{7.1}$$

where $p(\mathbf{t}_n)$ is the prior distribution over prototypes and defined as a uniform distribution over *all* training examples, $p(\mathbf{z}_n)$ is a continuous distribution over the edit vector, and $p_{\boldsymbol{\gamma}}(\mathbf{x}_n | \mathbf{t}_n, \mathbf{z}_n)$ represents a sequence-to-sequence model parameterized by $\boldsymbol{\gamma}$. This model is referred to as the *neural editor*. Guu et al. (2018)'s stated goal of the neural editor is to take direct advantage of training examples to improve language modeling performance while capturing interpretable semantic or syntactic properties in the latent prototype and edit vector variables. However, because the prototypes are selected from the *entire* training dataset, such a formulation sacrifices memory and speed efficiency due to the necessity of indexing and searching every training example at test time. In the following section, we detail our approach to mitigate this issue through the learning of *sparse* prototypes.

7.3 Method

First we present our our proposed generative model, then we describe the learning and inference techniques for this model class.

7.3.1 Model Structure

In the previous formulation, Eq. 7.1, maintaining the entire training dataset at test time is necessary due to assuming a uniform prior over prototypes $p(\mathbf{t})$. Motivated by the hypothesis that a much

[1] Below, we sometimes ignore the subscript to simplify notation when there is no confusion.

smaller prototype set would suffice to achieve comparable performance, however, we believe that $p(\mathbf{t})$ can be a sparse distribution where the probability mass concentrates on only a few representative prototypes. Since which training examples are representative as prototypes is unknown in advance, we propose to model the prototype distribution $p(\mathbf{t})$ as a latent variable, endowing the model with freedom to infer a sparse prototype posterior automatically. We define $\theta \equiv p(\mathbf{t})$ and further assume that the latent prototype distribution θ is sampled from a prior distribution $p_\alpha(\theta)$ (detailed below) which is able to encourage a sparse probability distribution, given appropriate hyperparameters. The graphical model is depicted in Figure 7.1, which gives the following joint likelihood:

$$p(\{\mathbf{x}_n, \mathbf{t}_n, \mathbf{z}_n\}_{n=1}^N, \theta; \alpha, \boldsymbol{\gamma}) = p_\alpha(\theta) \prod_n p(\mathbf{t}_n|\theta) p(\mathbf{z}_n) p_{\boldsymbol{\gamma}}(\mathbf{x}_n|\mathbf{t}_n, \mathbf{z}_n). \tag{7.2}$$

The log marginal likelihood of the data, which we will approximate during training is:

$$\log p(\mathbf{X}; \boldsymbol{\gamma}, \alpha) = \log \int_\theta p_\alpha(\theta) \Big[\prod_n \sum_{\mathbf{t}_n} \int_{\mathbf{z}_n} p(\mathbf{t}_n|\theta) p(\mathbf{z}_n) p_{\boldsymbol{\gamma}}(\mathbf{x}_n|\mathbf{t}_n, \mathbf{z}_n) d\mathbf{z}_n \Big] d\theta. \tag{7.3}$$

This is a general framework for learning sparse prototypes that we refer to as the *sparse neural editor*, and in this work we specifically experiment with the following parameterization to instantiate this model class:

Prior over prototype distribution $p_\alpha(\theta)$: We employ the Dirichlet distribution as $p_\alpha(\theta)$: $p_\alpha(\theta) \propto \prod_{k=1}^N \theta_k^{\alpha_k - 1}$. The support of Dirichlet distribution is the standard $N-1$ probability simplex. Here we use the *symmetric* Dirichlet distribution which has the same α value for all components since we have no prior knowledge favoring one component over another. α is positive and also referred to as the concentration parameter, where smaller α prefers a sparser prototype distribution θ, with $\alpha = 1$ equivalent to a uniform distribution over the probability simplex. In our experiments, we often choose $\alpha < 1$ to encourage sparsity.

Prior over edit vector $p(\mathbf{z})$: We follow Guu et al. (2018) and utilize a von-Mises Fisher (vMF) distribution to model $p(\mathbf{z})$. The vMF distribution places mass on the surface of the unit sphere, and is parameterized by the mean direction vector $\boldsymbol{\mu}$ and concentration parameter κ as $\text{vMF}(\cdot|\boldsymbol{\mu}, \kappa)$. Thus, information about the edit is captured through the directions of different unit vector samples. Xu and Durrett (2018) empirically show that the vMF distribution has the advantage of overcoming posterior collapse that plagues a large amount of previous work in latent variable models of text (Bowman et al., 2016; He et al., 2019). While Guu et al. (2018)

add additional randomness on the norm of edit vectors by multiplying the vMF distribution with another uniform distribution, we sample edit vectors from a uniform vMF distribution directly, which simplifies the model but we nonetheless found sufficient to obtain competitive results. Formally, we define $p(\mathbf{z}) = \text{vMF}(\mathbf{z}|\cdot, 0)$.

The editor $p_{\boldsymbol{\gamma}}(\mathbf{x}|\mathbf{t}, \mathbf{z})$**:** Generally $p_{\boldsymbol{\gamma}}(\mathbf{x}|\mathbf{t}, \mathbf{z})$ can be parameterized by any standard Seq2Seq model with the edit vector $\mathbf{z}$ incorporated. To compare with Guu et al. (2018) directly in the experiments, in this work we adopt the same attentional LSTM architecture (Hochreiter and Schmidhuber, 1997). $\mathbf{z}$ is utilized to predict the initial hidden state of the decoder and concatenated to the input for the decoder.

7.3.2 Learning and Inference

Ideally the log marginal likelihood in Eq. 7.3 should be optimized during training. However, computation is intractable due to marginalization of latent variables, and we resort to amortized variational inference (Kingma and Welling, 2014), optimizing its evidence lower bound (ELBO) instead:

$$\begin{aligned}
\log p(\mathbf{X}; \boldsymbol{\gamma}, \alpha) &\geq \mathcal{L}_{\text{ELBO}}(\mathbf{X}; \boldsymbol{\gamma}, \alpha, \boldsymbol{\phi}_{t|x}, \boldsymbol{\phi}_{z|t,x}, \boldsymbol{\lambda}) \\
&= \sum_n \Big\{ \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}_n|\mathbf{x}_n) q_{\boldsymbol{\phi}_{z|t,x}}(\mathbf{z}_n|\mathbf{t}_n, \mathbf{x}_n)}[\log p_{\boldsymbol{\gamma}}(\mathbf{x}_n|\mathbf{t}_n, \mathbf{z}_n)]}_{\text{reconstruction log likelihood } \mathcal{L}_{\text{rec}}} \\
&\quad - \mathbb{E}_{q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}_n|\mathbf{x}_n)}[D_{\text{KL}}(q_{\boldsymbol{\phi}_{z|t,x}}(\mathbf{z}_n|\mathbf{t}_n, \mathbf{x}_n)||p(\mathbf{z}_n))] \\
&\quad - \mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[D_{\text{KL}}(q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}_n|\mathbf{x}_n)||p(\mathbf{t}_n|\theta))] \Big\} - D_{\text{KL}}(q_{\boldsymbol{\lambda}}(\theta)||p_\alpha(\theta)),
\end{aligned} \tag{7.4}$$

where q represents the variational distribution to approximate the model posterior distribution and admits the following factorization form:

$$q(\theta, \{\mathbf{t}_n, \mathbf{z}_n\}_{n=1}^N|\mathbf{X}; \boldsymbol{\lambda}, \boldsymbol{\phi}_{t|x}, \boldsymbol{\phi}_{z|t,x}) = q_{\boldsymbol{\lambda}}(\theta) \prod_n q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}_n|\mathbf{x}_n) q_{\boldsymbol{\phi}_{z|t,x}}(\mathbf{z}_n|\mathbf{t}_n, \mathbf{x}_n). \tag{7.5}$$

Note that we make conditional independence assumption between θ and other latent variables in q to simplify the approximate posterior, following common practice in traditional mean field variational inference. The inference diagram is depicted in Figure 7.1. The optimal $q_{\boldsymbol{\lambda}}(\theta)$ to maximize Eq. 7.4 is a Dirichlet distribution parameterized by $\boldsymbol{\lambda} \in \mathbb{R}_+^N$ (proof is in Appendix .1), i.e., $q_{\boldsymbol{\lambda}}(\theta) = \text{Dir}(\theta; \boldsymbol{\lambda})$.[2] And $q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}|\mathbf{x}) = \text{Cat}(\mathbf{t}; f_{\boldsymbol{\phi}_{t|x}}(\mathbf{x}))$, the *prototype retriever*, is a categorical distribution over training examples parameterized by a neural

[2] $q_{\boldsymbol{\lambda}}(\theta)$ is not symmetric and $\boldsymbol{\lambda}$ is a vector.

network $f_{\boldsymbol{\phi}_{t|x}}(\mathbf{x})$. We assume $q_{\boldsymbol{\phi}_{z|t,x}}(\mathbf{z}|\mathbf{t},\mathbf{x})$, the *inverse neural editor*, is a vMF distribution $q_{\boldsymbol{\phi}_{z|t,x}}(\mathbf{z}|\mathbf{t},\mathbf{x}) = \text{vMF}(g_{\boldsymbol{\phi}_{z|t,x}}(\mathbf{t},\mathbf{x}),\kappa)$ where the mean direction parameter is from an encoder g that encodes $\mathbf{t}$ and $\mathbf{x}$ parameterized by $\boldsymbol{\phi}_{z|t,x}$, and the scalar concentration parameter κ is a hyperparameter. Pre-fixing κ results in a constant KL divergence term associated with $\mathbf{z}$ and proves to be effective to mitigate the posterior collapse issue (Xu and Durrett, 2018) where $\mathbf{x}$ and $\mathbf{z}$ become independent. Yet there might be still posterior collapse on $\mathbf{t}$ where, for example, the prototype retriever always predicts a uniform distribution or a degenerate distribution concentrated on a certain prototype regardless of $\mathbf{x}$. To overcome this issue, we follow (Li et al., 2019) to combine annealing (Bowman et al., 2016) and free-bits techniques (Kingma et al., 2016) and apply them to the term $\mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[D_{\text{KL}}(q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}_n|\mathbf{x}_n)||p(\mathbf{t}_n|\theta))]$.

Notably, the variational distribution family defined in Eq. 7.5 admits tractable closed-form expressions of all three KL divergence terms in Eq. 7.4 (detailed derivations and expressions are in Appendix .1). To compute the reconstruction log likelihood $\mathcal{L}_{\text{rec}}$, expectations over $\mathbf{z}$ can be efficiently approximated by the reparameterization trick for the vMF distribution (Guu et al., 2018). However, the prototype $\mathbf{t}$ is discrete and non-differentiable, and summing over all prototypes $\mathbf{t}$ to compute $\mathcal{L}_{\text{rec}}$ is infeasible due to the evaluation burden of $\log p_{\boldsymbol{\gamma}}(\mathbf{x}|\mathbf{t},\mathbf{z})$. Thus, we use the REINFORCE algorithm (Williams, 1992) to compute the gradients of $\boldsymbol{\phi}_{t|x}$ contributed from $\mathcal{L}_{\text{rec}}$ as:

$$\frac{\partial \mathcal{L}_{\text{rec}}}{\partial \boldsymbol{\phi}_{t|x}} = \frac{1}{L}\sum_{l=1}^{L}\Big(\mathbb{E}_{q_{\boldsymbol{\phi}_{z|t,x}}(\mathbf{z}|\mathbf{t}^{(l)},\mathbf{x})}\big[\log p_{\boldsymbol{\gamma}}(\mathbf{x}|\mathbf{t}^{(l)},\mathbf{z})\big] - b\Big)\frac{\partial \log q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}^{(l)}|\mathbf{x})}{\partial \boldsymbol{\phi}_{t|x}}, \tag{7.6}$$

where $\mathbf{t}^{(l)}$ are samples from $q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}|\mathbf{x})$. We use an average reward from L samples as the baseline b. The neural parameters $\boldsymbol{\gamma}, \boldsymbol{\phi}_{t|x}, \boldsymbol{\phi}_{z|t,x}$ are updated with stochastic gradient descent to maximize Eq. 7.4. With respect to the posterior Dirichlet parameter $\boldsymbol{\lambda}$, we found in preliminary experiments that classic gradient descent was unable to update it effectively – $\boldsymbol{\lambda}$ was updated too slowly and the Dirichlet prior became decoupled with the model. Thus, we instead update $\boldsymbol{\lambda}$ with stochastic variational inference (SVI, Hoffman et al. (2013)) based on the formula of the optimal $\boldsymbol{\lambda}^*$ given $q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}|\mathbf{x})$ (derivations can be found in Appendix .1):

$$\lambda_k^* = \alpha + \sum\nolimits_{n=1}^{N} q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}_n = \mathbf{x}_k|\mathbf{x}_n). \tag{7.7}$$

It is infeasibly expensive to keep $\boldsymbol{\lambda}$ optimal under current $q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}|\mathbf{x})$ at each training step, as it would involve summing over all training examples. Thus we perform SVI, which uses a batch of examples to approximate Eq. 7.7, leading to the following update form:

$$\lambda_k^{(t)} = (1-\rho_t)\lambda_k^{(t-1)} + \rho_t\Big(\alpha + \frac{N}{B}\sum\nolimits_{i=1}^{B} q_{\boldsymbol{\phi}_{t|x}}(\mathbf{t}_i = \mathbf{x}_k|\mathbf{x}_i)\Big), \quad \rho_t = (t+\sigma)^{-\tau}, \tag{7.8}$$

where B is the batch size, ρ_t is the step-size at iteration t, $\tau \in (0.5, 1]$ is the forgetting rate, and $\sigma \geq 0$ is the delay parameter to down-weight early iterations.

We note that our training algorithm is different from Guu et al. (2018) in that we use a learnable prototype retriever $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$ to derive a lower bound as the objective while Guu et al. (2018) directly approximate marginalization over $\mathbf{t}$. They use heuristics to fix the prototype set for each $\mathbf{x}$ to be examples similar to $\mathbf{x}$ in terms of edit distance, which might produce suboptimal prototypes for the generative model and also does not permit the learning of sparse prototype support.

Sparsity and scalability: After training we expect to be able to infer a *sparse* prototype distribution θ with most components being almost zero, based on which we can prune and store the entries over a particular probability threshold only, improving memory- and time-efficiency at test time. Specifically, we compute mean of θ under the Dirichlet posterior: $\mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[\theta_k] = \lambda_k / \sum_i \lambda_i$, and then take the largest M entries that occupy 90% of the probability mass. At test time, we only maintain these M prototypes and the prototype retriever $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$ is re-normalized accordingly. One issue present during training is that $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$ cannot fit into memory when dealing with large datasets since it is a categorical distribution over all training examples. In this work, we randomly downsample a subset of training data as our prototype library before training if memory is unable to fit all training examples, and learn the sparse prototypes on top of this subsampled corpus. This acts like a rough pre-filtering and in Section 7.4 we show that it suffices to learn good prototypes and achieve competitive language modeling performance. We leave more advanced techniques to address this issue (e.g. dynamically updating the prototype library) as future work.

Architectures: We now describe the neural architectures we use for the prototype retriever $q(\mathbf{t}|\mathbf{x})$ and inverse neural editor $q(\mathbf{z}|\mathbf{t}, \mathbf{x})$. $q(\mathbf{t}|\mathbf{x})$ is defined as:

$$q(\mathbf{t} = \mathbf{x}_k|\mathbf{x}) \propto \begin{cases} \exp\left(h(\mathbf{x}_k, \mathbf{x})/\mu\right) & \text{if } \mathbf{x}_k \neq \mathbf{x} \\ 0 & \text{if } \mathbf{x}_k = \mathbf{x}, \end{cases} \qquad h(\mathbf{x}_k, \mathbf{x}) = \texttt{Embed}(\mathbf{x}_k)^\top \boldsymbol{W} \texttt{Embed}(\mathbf{x}), \tag{7.9}$$

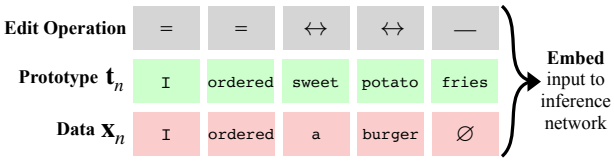


Figure 7.2: Example of aligned sequences.

where we prevent selecting the data example itself as the prototype during training to avoid overfitting. $\texttt{Embed}(\cdot)$ is a pretrained sentence encoder, $\boldsymbol{W}$ is a linear transformation matrix, and μ is a temperature hyperparameter

to control the entropy of $q_{\phi_{t|x}}(\mathbf{t}|\mathbf{x})$, which is critical to stabilize the Reinforce algorithm at the initial training stage. To ease the computation of encoding all training examples at each update step, we fix the parameters of $\texttt{Embed}(\cdot)$ and update $\boldsymbol{W}$ only, which proves to be sufficient in our experiments. We use the average embeddings of the last layer in pretrained BERT (Devlin et al., 2018)[3] as our sentence encoder.

While Guu et al. (2018) uses sum of inserted/deleted word vectors as the mean direction parameter of vMF distribution $q(\mathbf{z}|\mathbf{t}, \mathbf{x})$, we choose a more powerful encoder following recent advances on representing edits (Yin et al., 2019). Specifically, a standard diffing algorithm is run to compute an alignment of tokens in $\mathbf{t}$ and $\mathbf{x}$, and produces two aligned sequences $\mathbf{t}', \mathbf{x}'$ and an additional edit sequence that indicates edit operation $+$ (insertion), $-$ (deletion), $\leftrightarrow$ (substitution), and $=$ (equal) at each position. We use $\varnothing$ to denote padding. This is illustrated in Figure 7.2. Word embeddings of all three sequences are concatenated and fed into a single-layer LSTM to obtain the edit representation.

7.4 Experiments

Our experiments below are designed to (1) examine the efficacy of the proposed sparse neural editor on language modeling, (2) examine the efficiency of sparse neural editor on memory savings and speed-up at test time, and (3) demonstrate the interpretable semantics/syntax captured by prototypes and edit vectors.

7.4.1 Setup

We perform experiments on three different-scale datasets to test our method in different scenarios:

- MSCOCO (Lin et al., 2014): MSCOCO is an image caption dataset and we only focus on its captions as our data. The average length of captions is 12.6. The sentences do not have complex variations and are easy to find similar sentences as prototypes. We randomly sample 40K sentences as our training data and 4K as validation and test set respectively. This dataset represents a simple and small-scale setting to test our approach.
- Yelp Medium/Yelp Large: These datasets consist of sentences from Yelp restaurant reviews (Yelp, 2017) preprocessed by Guu et al. (2018), allowing us to perform a direct comparison with their method. The medium and large datasets consist of 1.5M and 17M sentences respectfully,

[3] We use pretrained uncased BERT base from the transformers library (Wolf et al., 2019).

allowing us to test in moderate and relatively large data settings respectively. Note that Yelp Medium is obtained by further filtering Yelp Large to keep sentences that are generally shorter and have less variations, but the test sets for these two are the same.

Baselines: We mainly consider neural language models (NLM) and the neural editor (Guu et al., 2018) as our baseline. It is worth noting that the neural editor model does not have likelihood defined on test sentences that are not similar to any example in the prototype library, thus it is necessary to interpolate with another NLM at test time for smoothing purposes, while our model is able to be used on its own. Note that we only report the neural editor baseline on Yelp Large since their public code is not ready to run on other datasets due to the required pre-built index to retrieve prototypes.

Evaluations: We evaluate language modeling performance with perplexity (PPL). For our model, we approximate the log marginal data likelihood through 1000 importance-weighted latent variable samples (Burda et al., 2015) and compute PPL based on this likelihood. At test time our approach prunes and has access to only M prototypes that occupy 90% probability mass of the posterior prototype distribution as described in Section 7.3.2. We report M as "#prototypes". To have an intuitive notion about how similar the prototype and data examples are, we compute average of smoothed sentence-level BLEU scores (Lin and Och, 2004) of the data examples on the validation set with their most likely prototype as the reference. We also report BLEU scores based on part-of-speech sequences (POS-BLEU)[4] to view the similarity from a more syntactic perspective. Test speed is evaluated on a single Nvidia 1080 Ti GPU.

Hyperparameters: We try different Dirichlet prior parameters α to control the sparsity and report different sparsity settings for all the datasets. The temperature parameter μ in the prototype retriever $q(\mathbf{t}|\mathbf{x})$ is tuned on the MSCOCO validation data and set as 0.3 for all datasets. The concentration parameter κ of the vMF distribution is tuned and set as 30 for MSCOCO and Yelp Medium and 40 for Yelp Large. The number of Reinforce samples L is set as 10 across all datasets. We sample 50K examples for Yelp Medium and 100K examples for Yelp Large as our training prototype library to address the training memory issue discussed in Section 7.3.2. We employ the same attentional LSTM Seq2Seq model as in Guu et al. (2018) to parameterize $p_\gamma(\mathbf{x}|\mathbf{t}, \mathbf{z})$ for a direct comparison. Our implementation is based on the fairseq toolkit (Ott et al., 2019) and complete hyperparameter settings can be found in Appendix .2.

[4]POS tagging is performed using the Stanza library (Qi et al., 2020).

Table 7.1: Results on three datasets. Numbers in the parentheses indicate the percentage of prototypes over all training examples. BLEU score is computed by comparing validation examples against their most likely prototypes. POS-BLEU represents the BLEU score on part-of-speech sequences. We also list BLEU scores from random prototype retrieval as a reference point. Results in the starred entry ($*$) are obtained by running the public code of the neural editor.

Dataset	Model	PPL↓	#prototypes	test speed (sent/s)↑	BLEU	POS-BLEU
MSCOCO	random retrieval	–	–	–	10.9	31.6
	NLM	20.0	–	3714	–	–
	Sparse Neural Editor ($\alpha = 10^{-3}$)	18.9	25 (0%)	388	13.2	38.8
	Sparse Neural Editor ($\alpha = 0.1$)	**18.6**	778 (2%)	313	17.2	42.5
	Sparse Neural Editor ($\alpha = 0.2$)	19.0	16K (40%)	250	20.9	46.7
	Sparse Neural Editor ($\alpha = 0.3$)	19.2	22K (56%)	217	22.2	47.9
Yelp Medium	random retrieval	–	–	–	8.1	17.8
	NLM	74.7	–	236	–	–
	Sparse Neural Editor ($\alpha = 10^{-3}$)	63.6	77 (0%)	157	12.3	24.7
	Sparse Neural Editor ($\alpha = 0.5$)	**61.9**	1.5K (0.1%)	107	21.6	38.4
	Sparse Neural Editor ($\alpha = 10$)	63.2	31K (2.1%)	95	29.9	48.3
Yelp Large	random retrieval	–	–	–	6.6	16.0
	NLM	34.2	–	272	–	–
	Sparse Neural Editor ($\alpha = 0.7$)	**30.2**	2K (0.01%)	108	10.5	24.8
	Sparse Neural Editor ($\alpha = 10$)	30.3	5.5K (0.03%)	98	10.8	25.3
	Interpolated w/ NLM					
	Neural Editor (Guu et al., 2018)	26.9	17M (100%)	–	–	–
	Neural Editor (our runs)*	31.2	17M (100%)	0.1	–	–
	Sparse Neural Editor ($\alpha = 0.7$)	**20.2**	2K (0.01%)	108	10.5	24.8

7.4.2 Results

Results are shown in Table 7.1. Our sparse neural editor outperforms the NLM baseline across all datasets in terms of PPL, often by a large margin. When interpolated with an NLM at test time, our method outperforms the NLM baseline by 14 PPL points and neural editor by 6.7 PPL points.[5] This effect is also observed in (Guu et al., 2018) – prototype-driven language models are especially strong at modeling test sentences that have similar prototypes but relatively weak at modeling others, thus interpolation with a normal NLM is likely to help. Furthermore, in line with the

[5]For a fair comparison, we interpolate with the same pretrained NLM from (Guu et al., 2018).

goal of this work to learn a sparse prototype set, our method is able to achieve superior language modeling performance while utilizing only a small fraction of training examples as prototypes. This verifies our hypothesis that a sparse prototype set suffices for such non-parametric language modeling. Also, sparsity learned in our model allows for over a 1000x memory savings and 1000x speed-up[6] at test time on Yelp Large compared with a previous neural editor that memorizes all training examples.

Table 7.1 demonstrates the trend that a smaller Dirichlet hyperparameter α leads to a sparser prototype set, which agrees with our expectation. BLEU scores are also improved as α increases, implying that the less sparse the prototype set the closer the match between the sentence and its prototype. Interestingly, the BLEU score on Yelp Large is a bit low and the model tends to generally favor sparse prototypes. We suspect that this is because it is difficult to learn prototypes that capture fine-grained semantic features and large sentence variations among 17M examples with a limited prototype memory budget, thus the model has to learn prototypes that represent more generic shared features among examples to reach an optimum – for example, the syntactic feature as somewhat reflected by the decent POS-BLEU scores.

We want to emphasize that different sparsity may lead to different notions of prototypes and it is hard to judge which one should be preferred – memorizing more prototypes pays cost on language modeling performance and does not necessarily produce better PPL. Also, prototypes that are "less similar" to the examples on the superficial level but capture coarse-grained features may have potentially interesting application on sentence generation since the model is able to generate more diverse output conditioned on prototypes.

7.4.3 Analysis

How do prototypes change when they grow sparser? In Table 7.1 we notice the BLEU scores usually drop when the prototype set is sparser, implying the learned prototypes change in some way under different sparsity settings. Here we take the Yelp Medium dataset as an example to analyze how prototypes are trained to capture sentence attributes differently in the sparse ($\alpha = 0.5$) and relatively dense ($\alpha = 10$) situations. Specifically, we align prototype and example sequences to minimize edit distance, and focus on words that were aligned between the two sequences. This

[6]We include the time to retrieve prototypes for new test sentences when computing test speed. We use Guu et al. (2018)'s public implementation of the neural editor, where the computation of edit distance between all training examples and test sentences to find nearest neighbors accounts for much of the runtime. More efficient implementation of this operation, for example through tries, may speed this to some extent.

Table 7.2: Number of matching tokens between examples and their prototypes on the Yelp Medium validation set. Results are reported in cluster of POS tags. Relative changes that are larger than the overall change are bolded.

Model	**Overall**	**NOUN**	**DET**	**AUX**	**PRON**	**ADJ**	**VERB**	**CCONJ**
Sparse Neural Editor (31K prototypes)	91.2K	14.4K	9.6K	9.3K	9.0K	7.2K	6.4K	5.5K
Sparse Neural Editor (1.5K prototypes)	74.7K	9.9K	8.5K	8.2K	7.3K	5.6K	4.4K	5.0K
Relative Change	-18.1%	**-31.3%**	-11.5%	-11.8%	**-18.9%**	**-22.2%**	**-31.3%**	-9.1%

Table 7.3: Qualitative examples of prototypes when using denser and sparser prototype supports.

Data Examples	**Prototypes**
the best corned beef hash i 've ever had !	(dense) the best real corned beef hash i 've had . (sparse) the chicken satay is the best i 've ever had .
the grilled chicken was flavorful , but too flavorful .	(dense) the chicken was moist but it lacked flavor . (sparse) my sandwich was good but the chicken was a little plain .
i asked her what time they close and she said <cardinal> o'clock .	(dense) i asked what time they closed <date> , and was told <cardinal> . (sparse) we asked how long the wait was and we were informed it would be <time> .

Table 7.4: Qualitative examples from the MSCOCO dataset on interpolated sentence generation given the prototype. The first row is the given prototype, the second-row and the last-row sentences are obtained by sampling edit vectors from the prior, the rest three sentences are generated by interpolating between the two edit vectors.

Prototype: A man is using a small laptop computer	Prototype: A cat sitting on a sidewalk behind a bush
A man is using his laptop computer with his hands on the keyboard	A cat laying on top of a wooden bench
A man is using a laptop computer with his hands on the keyboard	A cat standing next to a tree in a park
A man is using a laptop computer while sitting on a bench	Two cats sitting on a bench near a park bench
A man is using a laptop computer in the middle of a room	A dog sitting on a bench near a park bench
A young man is using a laptop computer in the middle of a room	A dog sitting on a bench near a park bench

allows us to obtain a notion of what kind of words are more likely to be kept the same as the prototype during the model's editing process and how this pattern changes in different sparsity settings. We cluster these matched words in terms of POS tag and report the most common ones.[7]

Results are shown in Table 7.2. While the overall number of matching tokens decreases as the

[7]Note that alignment is performed on word sequences instead of POS sequences.

Table 7.5: Comparison with Neural Editor that uses heuristic pre-clustering to select a sparse prototype support. The results are on MSCOCO dataset.

Model	**PPL**↓	**#prototypes**	**BLEU**	**POS-BLEU**
NLM	20.0	–	–	–
Neural Editor (pre-cluster)	19.5	778	17.9	40.4
Sparse Neural Editor	18.6	778	17.2	42.5

prototype set becomes sparser, the content words exhibit a more drastic change (e.g. the nouns, adjectives, and verbs). In contrast, the function words experience a moderate decrease only (e.g. the determiners, auxiliaries, and coordinating conjunctions). This shows that the model tends to learn prototypes that drop fine-grained semantic distinctions but keep the same general syntax when a sparsity constraint is enforced, which is not surprising since it is difficult for a limited number of prototypes to capture large semantic variations in a large dataset. We list qualitative examples in Table 7.3 to demonstrate this phenomenon, where some content words such as "beef" or "chicken" differ between data examples and prototypes in the sparser setting, yet these words do not change in the dense setting.

Ablation Study on Pre-clustering Prototypes: There is a simple method that is able to endow the neural editor baseline with sparse prototypes – pre-clustering the training examples and using only a subset of the them as the prototype pool during training and test. To compare the effectiveness of this heuristic sparse prototypes and our learned sparse prototypes, we experiment this baseline on MSCOCO dataset and pre-cluster 778 prototypes that correspond to our setting $\alpha = 0.1$. Concretely, we run k-means to obtain 778 clusters of training sentences embedded by Sentence-BERT (Reimers and Gurevych, 2019b) which produces state-of-the-art semantic sentence embeddings.Then for each cluster we identify one prototype whose embedding is the closest to the cluster centroid. We use these 778 prototypes in the neural editor model (Guu et al., 2018) and train with the same objective as Guu et al. (2018).[8] Results are shown in Table 7.5. Our method outperforms the neural editor with the same number of prototypes and presents similar BLEU scores, which demonstrates the superiority of learned prototypes over the heuristically selected prototypes.

[8]The prior over edit vector $p(\mathbf{z})$ and the inverse neural editor $q(\mathbf{z}|\mathbf{t}, \mathbf{x})$ in sparse neural editor is different from the original neural editor model. For a fair comparison we use $p(\mathbf{z})$ and $q(\mathbf{z}|\mathbf{t}, \mathbf{x})$ described in this paper for the neural editor baseline here.

Interpolation on the edit vector space: We take our model with 1.5K prototypes on the MSCOCO dataset (i.e. $\alpha = 0.1$) and perform sentence interpolation in edit vector space. Specifically, we sample two edit vectors from the uniform vMF prior to produce two sentences for each prototype with beam search decoding, then we perform spherical linear interpolation of the two edit vectors to generate interpolated sentences in-between. Qualitative examples in Table 7.4 (more examples in Appendix .3) show that the edit vectors are able to *smoothly* capture minor edits over the given prototypes.

7.5 Summary

In this work, we propose a novel generative model that discovers a sparse prototype set automatically by optimizing a variational lower bound of the log marginal data likelihood. We demonstrate its effectiveness on language modeling and its efficiency advantages over previous prototype-driven generative models. The framework proposed here might be generalized to automatically discover salient prototypes from a large corpus. New kinds of prototype structure in text might be discovered through either injecting different biases into the model (e.g. sparsity biases in this paper), or incorporating prior knowledge into the prototype library before training. Finally, the approach might be easily extended to conditional generation (e.g. with the edit vectors depending on other data input), and we envision that inducing a sparse prototype set in this case may potentially facilitate controlling text generation through prototypes. We leave exploration in this direction as our future work.

Chapter 8

Efficient Nearest Neighbor Language Models

The previous chapter focuses on the retrieval efficiency of latent prototype models, where the retrieval happens on the sentence level. In this chapter, we aim to optimize the retrieval efficiency of another token-level, retrieval-augmented language model – the nearest neighbor language model (Khandelwal et al., 2020), where the model retrieves multiple tokens for each token to be predicted. We propose and explore three different techniques to reduce the datastore size and speed-up the evaluation of the token-level retrieval. This work is presented in:

- Junxian He, Graham Neubig, Taylor Berg-Kirkpatrick. Efficient Nearest Neighbor Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021

8.1 Introduction

Language models (LMs) are one of the most fundamental technologies in NLP, with applications spanning text generation (Bahdanau et al., 2015; Rush et al., 2015), representation learning (Peters et al., 2018; Devlin et al., 2019a; Yang et al., 2019), and few-shot learning (Radford et al., 2019; Brown et al., 2020). Modern neural language models (NLMs) based on recurrent (Mikolov et al., 2010; Sundermeyer et al., 2012) or self-attentional (Vaswani et al., 2017; Al-Rfou et al., 2019) neural networks are mostly *parametric*, where the predictions are solely dependent on the model parameters given the input data.

In contrast, recent *non-parametric* LMs (Guu et al., 2018; Khandelwal et al., 2020; He et al.,

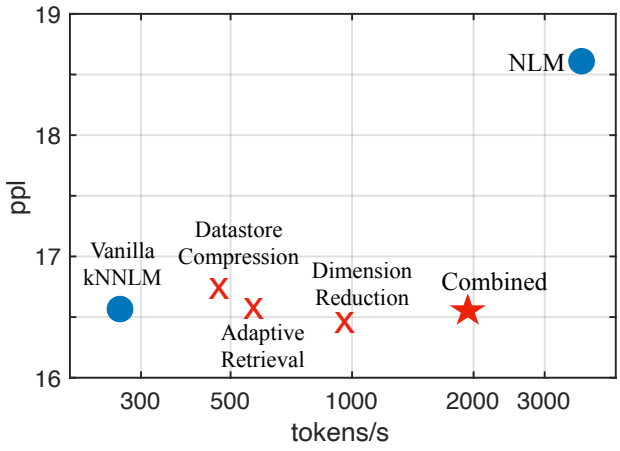


Figure 8.1: Perplexity (ppl) and evaluation speed for different models. Circled points represent the neural language model (NLM) and k-nearest neighbors language model (kNN-LM) baselines respectively, while others are the methods that we propose and explore in this chapter.

2020a) model text distributions by referencing both the parameters of the underlying model *and* examples from an external datastore. Non-parametric LMs are appealing since they allow for effective language modeling – particularly for rarer patterns – through explicit memorization via a datastore, which mitigates the burden on model parameters to learn to encode all information from a large dataset. One effective and representative example is the k-nearest neighbors LM (kNN-LM, Khandelwal et al. (2020)). The kNN-LM computes the probability of the next token by interpolating a parametric LM with a distribution calculated from the k nearest context-token pairs in the datastore, as demonstrated in Figure 8.2. This model is particularly notable for its large improvements in performance – it outperforms the previous best parametric LMs by a large margin in standard language modeling benchmarks, in domain adaptation settings, and on other conditional generation tasks such as machine translation (Khandelwal et al., 2021).

However, one downside to the kNN-LM is that the datastore stores high-dimensional dense vectors for *each token in the training data*; this can easily scale to hundreds of millions or even billions of records. As a result, the extra retrieval step from such datastores greatly decreases model efficiency at test time. For example, a 100M-entry datastore can lead to an over 10x slow-down compared to parametric models (§8.3.3) as shown in Figure 8.1. This issue poses a serious hurdle for the practical deployment of non-parametric LMs, despite their effectiveness.

In this work, we attempt to address this issue of test-time inefficiency and make non-parametric LMs more applicable in real-world settings. We take kNN-LM as an example, first analyzing the evaluation overhead, and raise three questions that we aim to answer in this chapter: (1) Do we

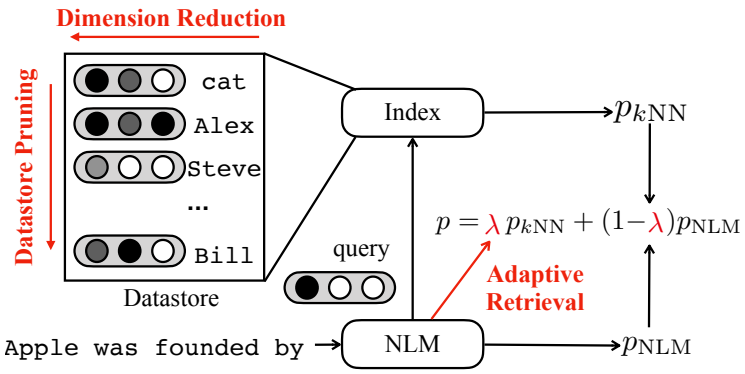


Figure 8.2: Illustration of kNN-LM . The datastore consists of paired context representations and the corresponding next tokens. The index represents a method that performs approximate k-nearest neighbors search over the datastore. Red and bolded text represent three dimensions that we explore in this chapter to improve the efficiency. In adaptive retrieval, we use a predictive model to decide whether or not to query the datastore.

really need to perform retrieval on the prediction of every single token? (2) Can we identify and prune redundant records from the datastore? (3) Is it possible to further compress the datastore by reducing the vector dimensionality without losing performance? We propose and explore potential solutions for each question to aid efficiency. Specifically, we (1) show that a lightweight network can be learned to automatically prune unnecessary retrieval operations (adaptive retrieval, §8.4.1), (2) explore several different methods for datastore pruning based on clustering, importance-guided filtering, or greedy merging (§8.4.2), and (3) empirically demonstrate that simple dimension reduction techniques are able to improve *both* the performance and speed (§8.4.3). Figure 8.1 illustrate the overall performance of these methods. Our experiments on the WikiText-103 language modeling benchmark (Merity et al., 2017) and a training-free domain-adaptation setting demonstrate speed improvements of up to 6x with comparable perplexity to the kNN-LM. On a higher level, we expect the empirical results and analysis in the chapter to help researchers better understand the speed-performance tradeoff in non-parametric NLMs, and provide a springboard for future research on more efficient non-parametric LMs.

8.2 Nearest Neighbors Language Model

In this section, we overview kNN-LM (Khandelwal et al., 2020) and its implementation details.

Formulation. kNN-LM (Khandelwal et al., 2020) is an LM that estimates token distributions by interpolating a pre-trained autoregressive NLM's distribution with another distribution computed using an external datastore. Specifically, given a sequence of context tokens $c_t = (w_1, \cdots, w_{t-1})$, the kNN-LM's next token probability $p(w_t|c_t)$ is calculated through the interpolation of the probability estimated by a standard parametric NLM $p_{\text{NLM}}(w_t|c_t)$ and a probability computed using an external datastore $p_{kNN}(w_t|c_t)$ (detailed later):[1]

$$\begin{aligned} &p(w_t|c_t) \\ &= \lambda p_{\text{kNN}}(w_t|c_t) + (1-\lambda) p_{\text{NLM}}(w_t|c_t), \end{aligned} \tag{8.1}$$

where λ is the interpolation hyperparameter. Note that the application of kNN-LM requires no additional training; the parameters of the NLM remain as-is, and Eq. 8.1 is only applied at test time. The workflow of kNN-LM is shown in Figure 8.2

Datastore. The datastore in kNN-LM stores context vectors from the pretrained NLM as keys, and their corresponding next tokens as values. Formally, let f be the key function that maps context sequence c to a fixed-size vector, then the datastore $(\mathcal{K}, \mathcal{V})$ contains all the key-value pairs constructed from the entire training examples $\mathcal{D}$:

$$(\mathcal{K}, \mathcal{V}) = \{(f(c_t), w_t) | (c_t, w_t) \in \mathcal{D}\}. \tag{8.2}$$

The size of such a datastore is almost equal to the number of training tokens because the context c_t is (nearly) unique due to the large context window size in modern recurrent (Sundermeyer et al., 2012) or self-attentional (Vaswani et al., 2017) NLMs. This suggests that the datastore can easily scale to hundreds of millions or even billions of records. Also, each $f(c_t)$ is a high-dimensional dense vector, which makes the datastore difficult to fit in memory. For example, a datastore from a 100M-token training dataset, using 1024-dimension context vectors at 16-bit precision, could require 200GB of memory.[2]

The Nearest Neighbor Distribution $p_{\text{kNN}}(w_t|c_t)$. At inference time, the kNN-LM (1) computes the context vector $f(c)$ from the current sequence using the pretrained NLM, (2) uses $f(c)$ as

[1] Below, we sometimes ignore the subscript to simplify notation when there is no confusion.

[2] Note that the dataset to contruct the datastore may not necessarily be the training data that trains the parametric NLM in Eq. 8.1 – a separate dataset may be used for the datastore construction which would lead to potential applications such as training-free domain adaptation or a gradient-free way to utilize extra training data (Khandelwal et al., 2020).

the query to retrieve k nearest neighbors $\mathcal{N} = \{(q_i, v_i)|i = 1, \cdots, k\}$ from the datastore, and (3) aggregates the retrieved tokens to form the distribution $p_{\text{kNN}}(w|c)$ to be used in Eq. 8.1 as:

$$p_{\text{kNN}}(w = y|c) \propto \sum_{(q_i,v_i)\in\mathcal{N}} \mathbb{I}_{v_i=y} \exp(-d(q_i, f(c))). \tag{8.3}$$

$d(\cdot, \cdot)$ is a distance function between the two vectors, and L^2 was shown to be more effective than other alternatives (Khandelwal et al., 2020). Intuitively, kNN-LM finds context sequences in the datastore that are similar to the test context, and then utilizes the next tokens observed after these contexts to help prediction. Such a mechanism allows language modeling through explicit memorization from the datastore, and may be particularly helpful for patterns rarely seen by the pretrained NLM (Khandelwal et al., 2020, 2021).

Sources of Inference Overhead. The extra inference overhead stems from the kNN search process in $p_{\text{kNN}}(w_t|c_t)$ computation. We denote the inference time per token as $t = t_{\text{NLM}} + t_{k\text{NN}}$. While t_{NLM} remains constant with different datasets, $t_{k\text{NN}}$ unfortunately grows as the datastore scales.

In practice, the kNN search process is often performed only approximately (ANN, Gionis et al. (1999); Muja and Lowe (2009)) to reduce computational cost. Khandelwal et al. (2020) implemented ANN search in kNN-LM[3] using FAISS (Johnson et al., 2021), which combines inverted file systems (Sivic and Zisserman, 2003) and product vector quantization (Jégou et al., 2011). This type of index reduces memory usage by only storing quantized vectors and accelerates kNN search by pre-clustering the datastore vectors; interested readers can refer to (Jégou et al., 2011) for more details. For the purpose of this work we study kNN-LM using this indexing method as a black box, aiming to improve efficiency in an index-agnostic way. At the same time, we note that building fast and accurate indexing methods remains an active area of research (André et al., 2019; Guo et al., 2020b), and selection or improvement of the index itself (possibly in concert with the methods proposed in this work) is an interesting avenue for future work.

Distance Recomputation. The distances to the nearest neighbors are required to compute $p_{\text{kNN}}(w_t|c_t)$ as shown in Eq. 8.3. However, as described above, kNN-LM's nearest neighbor search process performs search over quantized vectors, and as a result it can only return *approximate* distances. While it is possible to compute the accurate distances by reading the full-precision

[3] https://github.com/urvashik/knnlm.

vectors from the datastore after retrieval, this presents challenges as well: (1) storing the entire datastore in memory is not scalable for large datastores, (2) reading the vectors from a large datastore on disk on-the-fly is too slow to be practical (< 1 token per second).[4] Therefore, in this work we use the approximate distances directly to compute p_{kNN}. This comes at the cost of a minor performance loss, as we will show in §8.3.3. Similar approximations were adopted to apply kNN-LM to machine translation tasks (Khandelwal et al., 2021).

8.3 The Efficiency of NNLM

In this section, we first introduce the datasets and setup that we will use throughout the chapter, and then compare the inference speed of kNN-LM to parametric NLMs.

8.3.1 Datasets

We study kNN-LM in two different settings: (1) the standard setting where the datastore is constructed from the same data used to train the NLM, and (2) a domain adaptation setting where the datastore is based on the training data in the test domain, in which case the NLM never sees the examples included in the datastore. The following two datasets are used for the two settings respectively:

WikiText-103 (Merity et al., 2017) is a standard language modeling benchmark from Wikipedia that has 250K word-level vocabulary. It consists of 103M training tokens, and thus leads to a datastore that has 103M records and takes 200G space. Following (Khandelwal et al., 2020), we use the transformer-based (Vaswani et al., 2017) language model checkpoint released by (Baevski and Auli, 2019) as the underlying pretrained NLM, which is trained on the WikiText-103 training split.

Law-MT is an English-German machine translation dataset in the law domain originally released by (Koehn and Knowles, 2017) and resplit by (Aharoni and Goldberg, 2020). We only use the English text for language modeling. The training set consists of 19M tokens which we use to build the datastore that occupies 55G space. To inspect the domain-adaptation performance, our pretrained NLM is a 12-layer transformer model trained on WMT News Crawl[5] released by (Ng

[4] Disk random I/O is another aspect that may be improved by further engineering effort, which is also interesting future work.

[5] http://data.statmt.org/news-crawl/

et al., 2019).

8.3.2 Setup

Throughout the rest of the chapter, we adopt the same hyperparameters and index as (Khandelwal et al., 2020) for kNN-LM.[6] Specifically, the number of nearest neighbors is set to 1024 during evaluation.[7] Our pretrained NLMs are the state-of-the-art decoder-only transformers as mentioned above, and the key function $f(c)$ to obtain context vectors is the input to the final layer's feedforward network. The context vectors are 1024-dimensional and 1536-dimensional for WikiText-103 and Law-MT respectively. Given a dataset, we tune the interpolation weight λ on validation set in terms of the vanilla kNN-LM performance, and fix it unless otherwise specified. The tuning range of λ is [0,1, 0.9] with interval 0.05. As a result, $\lambda = 0.25$ in WikiText-103 and $\lambda = 0.9$ in Law-MT.

Hardware: Evaluation efficiency is benchmarked on 32 CPU cores (1.5 GHz AMD EPYC 7282) and 1 NVIDIA RTX 3090 GPU which represents a normalized environment – the index searching uses all the CPU cores while neural network computation is based on the GPU. Running retrieval on 32 CPU cores is also used by the FAISS repo[8] as a standard setting to benchmark large-scale retrieval.

8.3.3 Baseline Speed

We measure the perplexity (ppl) and speed of evaluation in term of tested tokens per second, and Table 8.1 reports the results on the test set of the two datasets. We also include "kNN-LM (exact)" for reference, which represents the kNN-LM variant that re-computes accurate distances as explained in §8.2. While very effective with 2 ppl points gains on WikiText-103 and over 90 points gains on Law-MT in a domain-adaptation setting, kNN-LM is 10x – 30x slower to evaluate on these datasets because of the extra retrieval step. When exact distances are computed by reading vectors from the disk on-the-fly, kNN-LM (exact) takes over 1 second to evaluate a single token.

[6] We directly base our experiments on the original kNN-LM implementation.

[7] The perplexity continues improving as k grows as shown in (Khandelwal et al., 2020) and confirmed by us. Yet k does not have an effect on the evaluation speed in the range [8, 1024] from our observation.

[8] https://github.com/facebookresearch/faiss/wiki/Indexing-1G-vectors

Table 8.1: Evaluation performance and speed of baseline models. The ppl numbers of kNN-LM * (exact) are from (Khandelwal et al., 2020), which recomputes accurate distances.

Model	WikiText-103		Law-MT	
	ppl	tokens/s	ppl	tokens/s
kNN-LM * (exact)	16.12	<1	–	–
NLM	18.66	3847	106.25	28K
kNN-LM	16.65	277	12.32	1052

Table 8.2: The features used to train the retrieval adaptor.

Feature	Description
$f(c)$	the context embeddings from pretrained NLM
conf(c)	the maximal value (confidence) of p_{NLM}
ent(c)	the entropy of the distribution p_{NLM}
$\log$ freq$(c[-n:])$	log of frequency of the immediate n context tokens computed from the training data. $n = 1, 2, 3, 4$ which leads to four scalar features.
$\log$ fert$(c[-n:])$	fert$(c[-n:])$ is the number of unique word (fertility) that succeeds the immediate n context tokens computed from the training data. $n = 1, 2, 3, 4$ which leads to four scalar features.

8.4 The Remedies

In this section we propose and explore several different methods that may potentially improve the efficiency of kNN-LM along three axes: (1) adaptive retrieval, (2) datastore pruning, and (3) dimension reduction. We analyze the performance of each method on WikiText-103, trying to conclude the best practices that we will evaluate in §8.5.

8.4.1 Adaptive Retrieval

Just as humans refer to books only when they are uncertain in an open-book quiz, the parametric NLMs may not *always* need help from the external datastore. To inspect this hypothesis, we compare $p_{\text{kNN}}(w|c)$ and $p_{\text{NLM}}(w|c)$ for every token in the WikiText-103 validate set. Interestingly, $p_{\text{kNN}}(w|c) \geq p_{\text{NLM}}(w|c)$ only 39% of the time – the likelihood of 61% of the tokens becomes worse after interpolation despite the overall improvement. This indicates that if we were able to identify these locations perfectly, 61% of the retrieval operations could be removed completely and we would achieve even better perplexity. Inspired by this observation, we aim to automatically

identify and prune unnecessary retrieval operations to speed up inference.

Methods: We propose to train a light neural network, the retrieval adaptor, to identify when we should remove the retrieval operation. Specifically, given the context c as the input, the retrieval adaptor may be trained with either (1) a classification objective to predict whether $p_{\text{kNN}}(w|c) \geq p_{\text{NLM}}(w|c)$, or (2) a likelihood maximization objective to predict the interpolation weight $\lambda(c)$ and maximize the overall likelihood of kNN-LM as in Eq. 8.1. In our preliminary results the classification method performs only on par with a random removal baseline, partially due to the discretized noisy supervision. Therefore, we directly maximize the kNN-LM log likelihood by modeling λ as a function of the context:

$$\mathcal{L} = \frac{1}{T} \sum_t \big[\log p(w_t|c_t; \lambda_\theta(c_t)) - a \cdot \lambda_\theta(c_t) \big], \tag{8.4}$$

where only θ – the parameters of the retrieval adaptor – are updated. The second term is an L^1 regularizer that encourages learning sparse weights for $p_{k\text{NN}}$, which we find helpful to prune unnecessary retrievals. At inference time, we prune a given fraction of retrievals with the smallest kNN weight $\lambda(c)$ by resetting $\lambda(c)$ to zero. The hyperparameters of the retrieval adaptor network including the regularizer coefficient, a, are tuned on the validation set in terms of perplexity at 50% retrieval pruning. Learning the interpolation weights to prune is related to (Johansen and Socher, 2017) where they learn to skip text for classification tasks. Optimizing the interpolation weights in kNN-LM has also been applied at training time to train the NLM jointly (Yogatama et al., 2021).

Input Features: We concatenate several neural and count-based features as input to the retrieval adaptor as shown in Table 8.2. For the scalar features (basically all the features excluding $f(c)$), we found it helpful to map them to a vector with a small network before concatenation. We note that all the features are trivial to obtain at test time – the neural features are from intermediate computation of $p_{\text{NLM}}(w|c)$ and count-based features are looked-up values.

Architecture and Hyperparameters: The retrieval adaptor is a light MLP network with linear transformation followed by ReLU activation at each layer. The output layer maps the hidden representation to a 2-dimensional vector followed by a `LogSoftmax` layer to yield $\log(\lambda)$ and $\log(1-\lambda)$ respectively. We use the same adaptive retrieval configuration hyperparameters for different datasets, which are validated on the WikiText-103 dev set: the retrieval adaptor is an MLP network with 4 hidden layers, 1 input layer and 1 output layer. Each layer is a

linear transformation followed by the ReLU non-linear activation, and a dropout layer with 0.2 dropout probability, except for the output layer where the hidden unites are transformed to 2 dimensions followed by a log softmax to produce $\log \lambda$ and $\log(1 - \lambda)$. The number of hidden units in each layer is 128. Before passing the input features to MLP, we transform each of the scalar features (all the features except for $f(c)$) into an m-dim vector, where $m = \dim(f(c))/n$ and n is the number of scalar feature types. This is to balance the context vector feature and other features. The scalar-feature transformation is performed with a one-layer `Linear(in, out)-ReLU-Linear(out, out)` network. We also tried using LSTM (Hochreiter and Schmidhuber, 1997) network to capture the temporal relations yet found it leads to very unstable training and fails to converge, though we note that MLP is faster at test time and the $f(c)$ feature already captures the temporal correlations between tokens. The coefficient of the L^1 regularizer a is tuned on WikiText-103 validation set among $\{0.01, 0.05, 0.1, 0.2, 0.5, 1\}$ and fixed as 0.05 for both WikiText-103 and Law-MT. The model is trained using the Adam optimizer (Kingma and Ba, 2015) with learning rate 0.0005. The checkpoint with the best validation perplexity at 50% pruning is saved.

Training: During training, only the retrieval adaptor is updated while the pretrained NLM is fixed. Note that it is inappropriate to train the retrieval adaptor on the training dataset, which would lead to biased solutions since p_{NLM} may have already overfit on the training data and the datastore includes the training example itself. To generalize to the test data, we hold out 10% of the validation data for validation and use the remaining 90% to train the retrieval adaptor. The retrieval adaptor is light and converges quickly; it took several minutes to train it on WikiText-103 with a single GPU.

Results: Figure 8.4a shows the perplexity and evaluation speed of adaptive retrieval on the test set of WikiText-103, varying the percent of removed retrieval operations. The different threshold values of λ used to cut off retrieval is selected based on the synthetic validation set mentioned above. We also add a random retrieval baseline which uniformly selects a certain fraction of retrieval operations to discard. We observe that adaptive retrieval (AR) exhibits a much flatter increase of perplexity than the random baseline when the number of removed retrievals grows. Notably, AR is able to achieve comparable perplexity to the original kNN-LM model (16.67 vs. 16.65) while being nearly 2x faster (530 vs. 277 tokens/s) through removing 50% of the operations.

Table 8.3: Results of kNN-LM + adaptive retrieval using different input features. The perplexities are based on removing 50% of the retrieval operations after training retrieval adaptor.

Features	**ppl**
$f(c)$ + conf + ent + log freq + log fert	16.63
-log freq	16.67
-log fert	16.72
-ent, conf	16.77
-$f(c)$	16.71
-conf, ent, log freq, log fert	17.03

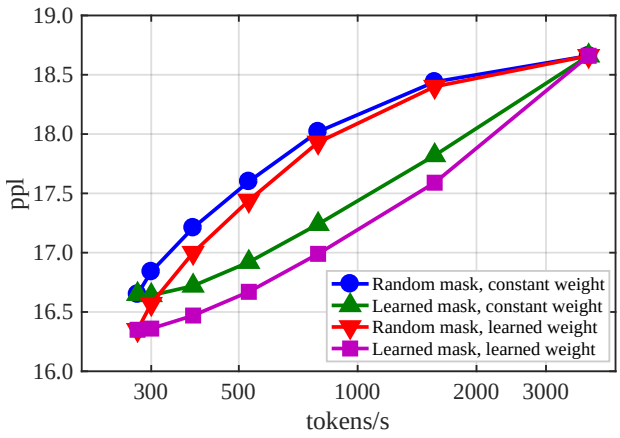
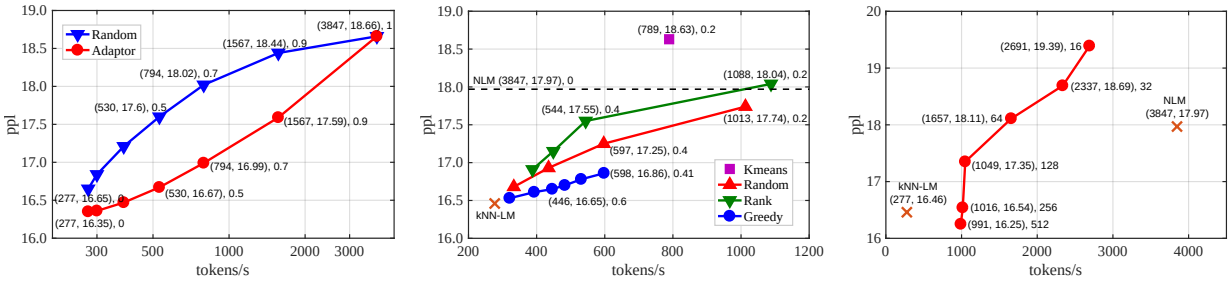


Figure 8.3: Perplexity and speed results of adaptive retrieval on WikiText-103 test set. This figure includes different variants of adaptive retrieval for ablation analysis.

Ablation on Input Features: We analyze the effect of different input features to the retrieval adaptor by removing a subset of features. We report the perplexities at 50% retrieval pruning, because using different features only has a marginal effect on the evaluation speed. Results on the WikiText-103 test set are shown in Table 8.3. All features together produce the best results, while the perplexity is relatively robust to removal of a single feature. In our experiments (§8.4 and §8.5) we drop off the log freq feature and use the others to save memory while achieving comparable perplexities to using all features.

Effect of Learnable Interpolation Weights: In the adaptive retrieval analysis, we observed gains of a learned retrieval adaptor over a random baseline at different fractions of retrieval pruning. However, the advantages may come from two sources: (1) the automatically identified pruning



(a) Adaptive Retrieval (b) Datastore Pruning (c) Dimension Reduction

Figure 8.4: Perplexity and speed results on WikiText-103. For datastore pruning and dimension reduction we show the results on validation data while results on the test set is reported for adaptive retrieval. We annotate the coordinates of some points, and the third number in the annotation is: (a) the fraction of retrieval operations that are removed; (b) the compression rate (fraction of records remained); (c) the PCA dimensions.

masks against the random masks, and (2) the learned interpolation weights on the remaining retrievals against the constant weights that random baseline uses. To separate the two effects, we perform an ablation study to analyze the results of (1) random mask, constant weight, (2) random mask, learned weight – the weights are from the trained retrieval adaptor, (3) learned mask, constant weight, and (4) learned mask, learned weight. The results are shown in Figure 8.3, "learned mask, learned weight" performs the best. While minor gains are from the automatically learned weights ("Random mask, learned weights"), most of the superiority can be attained with the smart pruning strategy even with constant weights ("Learned mask, constant weights").

8.4.2 Datastore Pruning

The information present in a large training dataset is often redundant, which suggests that a datastore constructed from training tokens may be pruned with no or only minor performance cost. To validate this hypothesis, we propose several different methods to prune the number of entries and reduce the datastore size:

Random Pruning: As a simple baseline, a certain fraction of the datastore entries are randomly selected. Random pruning has been shown to work well with a billion-scale datastore in machine translation tasks (Khandelwal et al., 2021).

k-Means Pruning: Clustering is a common technique to prune redundant vectors by only keeping the centroids of the clusters. Yet in our task specifically, we note that a general clustering on the context vectors is not directly applicable since the vectors in the same cluster may still correspond to various target tokens, as language use in context is not deterministic. Therefore, we propose to perform target-aware k-means clustering – for a word w_i in the vocabulary, we perform a separate k-means clustering for all the context vectors that have w_i as the target token, then we only keep centroids of each cluster as well as saving the cluster size s. The `(centroid vector, cluster size, target token)` triples form a new compressed datastore. Since we approximate multiple vectors in the same cluster with the centroid and only save the centroid vector once in the new datastore, the computation of the kNN distribution $p_{k\text{NN}}$ needs to be rectified as:

$$p_{\text{kNN}}(w = y|c) \propto \sum_{(q_i,v_i)\in\mathcal{N}} \mathbb{I}_{v_i=y}\ s_i \cdot \exp(-d(q_i, f(c))), \tag{8.5}$$

the cluster size s_i acts like weights for each datastore entry. Eq. 8.5 recovers Eq. 8.1 when every cluster is of size 1.[9] In practice, we perform 5000 separate k-means clustering passes only for the most frequent 5000 words due to high computational cost, which accounts for 84% of all the training tokens. For other vectors we treat each of them as a separate clusters with size 1. The number of clusters in k-means are set to 1/20 of the number of vectors to be clustered, which produces a 5x smaller datastore overall. We did not intensively tune the k-means hyperparameters due to the computational burden. We note that the clustering here is different from the pre-clustering in the ANN index with inverted file systems mentioned in §8.2– the index's pre-clustering does not actually reduce size and is just for lookup.

Greedy Merging: Generally we aim to merge records that share the same target token while being close to each other in vector space. Token-aware clustering is an attempt to achieve this goal, but forcing all points to participate in clustering – and the resulting large clusters – causes some points within the same cluster to be distant in some clusters with high variance. Thus approximating all the vectors with the cluster centroids may lead to large errors. To address this issue, we propose a simple approach, greedy merging (GM), which inspects every record in the datastore and greedily merges their nearest neighbors if a merging condition is satisfied. The

[9]In addition, the centroid formulation is roughly equivalent to saving vectors within the same cluster as the centroids multiple times without pruning in the original formulation.

Algorithm 2 Greedy Merging

$(\mathcal{K}, \mathcal{V}) = \{(q_i, v_i)\}_{i=1}^{N} \leftarrow$ the old datastore
$s \leftarrow \mathbf{1}$ ▷ weight vector with size N
for $(q_i, v_i) \in (\mathcal{K}, \mathcal{V})$ **do**
    retrieve K neighbors of (q_i, v_i) as $\{q_{t_k}, v_{t_k}\}_{k=1}^{K}$
    for $k = 1, 2, \cdots, K$ **do**
        if $s_{t_k} = 1$ & $v_i = v_{t_k}$ & $t_k \neq i$ **then** ▷ merge condition
            $s_i \leftarrow s_i + 1$ ▷ merge (q_{t_k}, v_{t_k}) into (q_i, v_i)
            $s_{t_k} \leftarrow s_{t_k} - 1$ ▷ Remove record t_k
        end if
    end for
end for
Save datastore $\{(q_i, v_i, s_i) | s_i > 0, i = 1, \cdots, N\}$

detailed algorithm is shown in Algorithm 2. Intuitively, GM is density-based to group points with nearest neighbors, but the merging operation only happens *locally* between a point and its nearest neighbors – it never propagates to merge the nearest neighbors of nearest neighbors unlike typical density-based clustering methods (Ester et al., 1996) which may amplify errors. Similar to k-means pruning, we also compute the weights s_i of each entry in the compressed datastore to correct $p_{k\text{NN}}$ computation using Eq. 8.5. Without a global clustering mechanism, this approach ensures that the merging vectors are close enough by inspecting only a small number of nearest neighbors. In the following analysis we vary the number of nearest neighbors K within range [2,100] to achieve different compression rates.

Rank-based Pruning: It is well known that embedding spaces contain "hubs" which are nearest neighbors of many other embeddings (Tomasev et al., 2014), and other points that are not nearest neighbors of any other points. We hypothesize that these entries which are rarely nearest neighbors may be removed without significant impact on the performance. To verify this assumption, we iterate every (c_i, w_i) pair in the training data as queries to search their k nearest neighbors from the datastore (k is set to a large number as 1024 here). In this process we compute an "importance score" for every entry in the datastore as $g = \sum_i 1/\text{rank}_i$, where rank_i is the rank of this entry among the nearest neighbors of the query $f(c_i)$. rank$= +\infty$ if it is not in the retrieval results. Intutively, the "importance score" up-weights the datastore records that appear more often with lower ranks in the retrieval results. Then we sort all the datastore records in terms of g and

remove the ones with small scores, varying the compression rate. This method shares spirit with the technique in (Min et al., 2021) which filters out the articles that are never retrieved in memory-constrained open-domain question answering tasks.

Results: Figure 8.4b demonstrates the perplexity v.s. speed results on Wikitext-103 validation set of different datastore pruning methods described above. Only one solution point is reported for k-means since we do not vary the hyperparameters of k-means for different compression rate, given that its computational cost is much higher than other methods. Using 20% of the original datstore, k-means even underperforms the vanilla NLM baseline, suggesting that the cluster centroids approximation may lead to large distance errors which reduce the accuracy of the kNN distribution. Surprisingly, the simple random pruning method outperforms more complicated ones such as k-means and rank-based pruning. The best approach is greedy merging, which demonstrates a relatively flat curve compared with others.

8.4.3 Dimension Reduction

The context vectors $f(c)$ from large NLMs are often high-dimensional. For example, the pretrained NLMs that we use produce vectors of 1024 and 1536 dimensions in WikiText-103 and Law-MT respectively, which incurs significant datastore space and distance computation cost. To mitigate this issue, we empirically explore the effect of dimension reduction in kNN-LM . Specifically, we use principal component analysis (PCA), an efficient and scalable dimension reduction algorithm, to reduce the dimensions and generate a new compressed datastore. We vary the new PCA dimensions as the hyperparameter and report the results.

Results: As shown in Figure 8.4c, the evaluation becomes faster as expected with smaller dimensions, yet a too aggressive compression (dimension < 256) incurs large perplexity cost and even loses advantages over NLM when the dimension is smaller than 128. However, at 256 and 512 dimensions PCA is able to achieve comparable or even better performance than the original 1024-dim vectors, while attaining 3x-4x speed-up.[10]

[10]The tool we use for PCA, the FAISS PCA implementation, applies random rotation to the PCA output vectors by default to re-balance variances of components of a vector (Gong et al., 2013), which may provide additional benefits over vanilla PCA on product vector quantization inside the index.

Table 8.4: Perplexity and speed results on the test set of WikiText-103 and Law-MT. AR, GM, DR denote adaptive retrieval, datastore pruning, and dimension reduction respectively, "+All" denotes the combination of all the three technique.

Methods	ppl	tokens/s	speedup
WikiText-103			
NLM	18.66	3847	13.9x
kNN-LM	16.65	277	1x
+AR ($r = 0.5$)	16.67	530	1.9x
+GM ($n = 0.6$)	16.86	446	1.6x
+DR ($d = 512$)	16.40	991	3.6x
+All	16.67	1835	6.6x
Law-MT			
NLM	106.56	27.8K	264.3x
NLM (fine-tuned)	8.61	27.8K	264.3x
kNN-LM	12.64	1052	1x
+AR ($r = 0.1$)	12.74	1290	1.2x
+GM ($n = 0.6$)	13.33	1451	1.4x
+DR ($d = 512$)	11.59	3420	3.3x
+All	12.29	5708	5.4x

8.5 Putting it All Together

Based on the analysis results in §8.4, in this section we combine best practices in adaptive retrieval, datastore pruning, and dimension reduction to assess the performance. We select the retrieval pruning rate r, datastore pruning rate n, and the reduced dimensions d on the validation set,[11] so that they achieve the largest speed-up at the cost of $<= 0.1$ perplexity compared to vanilla kNN-LM . We report the results on the test set.

Results: Table 8.4 shows the results on the test set of WikiText-103 and Law-MT, where we assess the combination of all three different strategies. Separate performance for each strategy is also included for reference points. On WikiText-103, adaptive retrieval is able to remove 50% of

[11] Adaptive retrieval uses part of the validation data to training the retrieval adaptor network, thus we select r separately on its own held-out validation and then combine it to others.

the retrieval and achieve nearly 2x speed-up, greedy merging prunes 40% of the datastore at the cost of 0.2 perplexity points. The dimension reduction method PCA leads to a minor improvement of perplexity over kNN-LM while being 3.6x faster. Combination of all the three techniques yields comparable perplexity to vanilla kNN-LM (16.67 v.s. 16.65) and a 6.6x speed-up (1835 v.s. 277).

Different from WikiText-103 where the datastore is contructed from the data that trains the pretrained NLM, in the Law-MT domain adaptation setting the datastore represents the domain-specific knowledge that the pretrained NLM never sees during training and thus is critical to produce good perplexity. This may be inferred from by the large ppl gains that the datastore offers (94 points). From another perspective though, the big improvement from the datastore retrieval leads to difficulties removing retrieval operations adaptively[12] – our learned retrieval adaptor is able to remove only 10% of the retrieval operations costing 0.1 ppl points. Greedy merging is able to prune 40% of the datastore losing 0.7 ppl points. We suspect that the Law-MT datastore is more vulnerable to pruning than the WikiText-103 one because of its smaller size (19M v.s. 103M) and corresponding lack of redundancy. Interestingly, the PCA dimension reduction yields 1 point ppl gain over the vanilla kNN-LM while achieving 3.3x speed-up, consistent with WikiText-103. This implies that a PCA transformation may be able to produce a new vector space that is more appropriate for defining $p_{k\text{NN}}$ with L^2 distances, we leave the underlying reasons for future work to discuss. Finally, a combination of the three allows kNN-LM to be evaluated 5.4x faster and even obtain superior perplexity.

8.6 Summary

In this chapter, we explore several different ways to improve efficiencies of the k-nearest neighbors language model, achieving up to 6x speed-up while attaining comparable performance. As for future work, it is interesting to explore features from the datastore side to better know when to retrieve, and the gap between retrieval-based NLMs and parametric NLMs may be further reduced by combining more optimized indexing methods and the approaches in this chapter.

[12]This can be reflected from the oracle comparison: $p_{\text{kNN}}(w|c) \geq p_{\text{NLM}}(w|c)$ 76% of the time compared to 39% in WikiText-103.

Chapter 9

Conclusions and Future Directions

As natural language generation is driven by big data and large models nowadays, we raise three different types of efficiency concerns that limit a broad usage of many techniques. They are label-efficiency on the number of annotated examples, parameter-efficiency on the number of tunable model parameters, and compute-efficiency in retrieval augmented models. This thesis presents our work to provide a deeper understanding of research problems, and utilize the insights to design better and more efficient approaches.

Part I focuses on improving NLG with unlabeled data. In Chapter 3, we propose a probabilistic framework for unsupervised sequence to sequence transduction, which unifies existing techniques and the probabilistic view naturally leads to better performance on a wide variety of sequence to sequence tasks without using any annotations; In Chapter 4 we try to analyze and understand the principle behind classic self-training techniques in the context of semi-supervised NLG, after concluding that noise plays an important role in self-training as a form of consistency regularization, we naturally propose noisy self-training to corrupt part of the input, which yields significant gains; In Chapter 5 we extend this consistency training from example-level to prompt-level to encourage consistency between different prompts. By formatting all NLP tasks as generation tasks through prompts, our method achieves the state-of-the-art zero-task generalization on multiple NLP datasets.

Part II studies parameter-efficient transfer learning (PETL) and proposes a unified framework for previous competitive approaches. Our method connects existing approaches and views them from a unified perspective, which helps us further obtain a deep understanding of the good or bad designs of the baselines. Last, it is straightforward to instantiate new PETL methods from the framework, which greatly outperforms the baselines and achieves comparable performance to fine-tuning all parameters despite tuning few parameters.

Part III focuses on the compute-efficiency in retrieval augmented NLG where information is retrieved from an external datastore to facilitate generation. In Chapter 7 we propose a new prototype-based language model that regularizes the datastore sampling distribution to be sparse, and thus prunes the datastore at inference time to compress the datastore size and speed up the retrieval process. The resulted model also outperforms the previous prototype-based baselines without sparsity regularization; In Chapter 8 we explore three different techniques to speed up the retrieval of nearest neighbor language models.

In addition to the contributions of the proposed approaches themselves, another higher-level takeaway from this thesis is that a deeper understanding or unified view of existing methods could often help other researchers take a different perspective to view previous progress, inspire future research, and directly lead to better approaches. Our work in Chapter 3, Chapter 4, and Chapter 6 are examples, where we start by understanding or reformulating previous techniques, and then deliver superior methods. We expect the results and discussions in this thesis to not only give approaches that work better empirically but reveal previously unknown insights into the focused research problems to inspire others.

While this thesis puts efforts to facilitate more efficient and practical NLG, we remark that these contributions are far from sufficient to achieve our final goal of building *practical, trustable, and efficient* NLG. To describe our vision specifically, we aim to create NLG systems that (1) are practical and robust enough to succeed in real-world scenarios; (2) are truthful and explainable – some model errors are not tolerable in practice, for example, an NLG model may produce toxic contents in psychological counseling. Therefore, NLG systems cannot be broadly deployed unless they are trustable, explainable, and even controllable. Currently, almost all the neural models are black boxes and lack explanations of their predictions, and state-of-the-art neural generation is still prone to hallucinating facts; (3) can be efficiently developed with reasonable resources, for example, we hope that any undergraduate student with a laptop or a small workstation could create or deploy state-of-the-art NLG models. However, the current mainstream trend of NLP development mostly emphasizes state-of-the-art numbers on benchmarks through training ever large models with huge compute (e.g. using thousands of GPUs). They are not efficient, explainable, controllable, or trustable. While we recognize the significant contributions and progress from training large transformer models, in the meanwhile we doubt whether this path is the way to go in the future 10 years or even towards artificial general intelligence (AGI) in the long run. Therefore, we still aim to continue working on, and expect to witness the emergence of brand-new models that are explainable and controllable, and are able to bring training efficiency to the next level.

Future Work

Interpretable, Trustable, and Controllable Methods

End-to-end neural networks composed of mysterious, real-valued neurons have achieved great success in the past decades. They are easy to be optimized through back-propagation and function like a black box. However, many important mechanisms behind these NNs remain poorly understood, such as why the model makes the prediction, how to control the prediction, how much we can trust the prediction, how the model learns to generalize compositionally (if at all), whether the model really "reasons" like humans, or how the model evolves during training (i.e. the training dynamics). The current architecture and learning paradigm makes it very challenging to study these questions. However, answering these questions, in our opinion, is crucial for general intelligence. In the short term, a deep understanding of the working mechanism of current NNs could help diagnose and develop better models as well. While some works try to study these problems within the current modeling paradigm through post-hoc explanation or post-hoc control of the model after training (Ribeiro et al., 2016; Sundararajan et al., 2017; Radford et al., 2017; Dai et al., 2022), we expect better frameworks or architectures for explainable, trustable, and controllable AI by design. Below we describe two examples as future directions.

Modular Learning: Modular learning aims to train various submodules with interpretable functionalities that are composable (Farooq, 2000; Andreas et al., 2016). Such disentanglement of functionalities among submodules could help interpret and control the model predictions to some extent. Parameter-efficient modules as we work on in Part II could be used in modular learning. Take adapters for an example, we may learn a "formal style adapter" to control the generated text to be formal language, and efficiently integrate this adapter into other sequence generation models to control the text style generally. Such modular and transferrable usage may also imply a new principle of community-wise model development based on adapters (or prefix vectors similarly), to enable us to share, download, and compose submodules from the cloud.

Integrating symbolic components into neural learning: On the other hand, we are looking forward to a better fusion of probabilistic symbolic components and neural learning to combine the strengths from both worlds – symbolic methods are generally explainable and controllable while neural networks are more expressive. Also, probabilistic and symbolic components in neural networks could open opportunities to inject human priors into the modeling process more easily.

Symbolic methods are appealing because humans often reason in a symbolic and composable manner. With such symbolic reasoning abilities, humans are able to learn fast, generalize compositionally, and explain predictions well. Combining symbolic and neural learning dates back to more than 20 years ago (Shavlik, 1994) and is still an active research direction (Mao et al., 2019; Hudson and Manning, 2019; Feng et al., 2022).

We note that developing new frameworks towards interpretable, trustable, and controllable AI as exemplified above could have implications on efficiency as well to achieve better Pareto efficiency. We share the point presented in Thompson et al. (2020) that "the progress of AI along current lines is rapidly becoming economically, technically, and environmentally unsustainable" and continued progress requires "changes to deep learning or moving to other machine learning methods". Therefore, we vision that the new models that better satisfy "interpretability, trustability, and controllability" are potential candidates for the next generation of AI techniques.

Data-Centric NLP

The development of NLP in the past decades is mostly model-centric, led by model advances from statistical symbolic methods to deep neural networks. In the deep learning era, we also experienced transitions from MLP to RNN, LSTM, and Transformer. As large-scale self-supervised pretraining becomes the standard and model architectures in different tasks converge, we believe that data-centric techniques could potentially have a big role in the next decade. Traditional data processing methods include simple text preprocessing and cleaning, such as removing noisy links, lowercasing the characters, and segmenting words in some languages (e.g. Chinese). Those techniques aim to clean the data so that models could learn more easily. More advanced data-centric approaches are mostly focused on data augmentation so far as reviewed in § 2.2.1. They are effective to improve the model performance and robustness on the test data. More recently, deduplication as a simple data preprocessing method has been found to improve large-scale language model training (Lee et al., 2022). Inspired by this progress, we think that data manipulation is a relatively efficient way to improve model performance and control the learned information, compared to engineering modeling techniques. For example, mitigating biases and replacing toxic contents in a dataset could expectantly reduce the bias and toxicity in the resulted models; approaching the private data leakage in the data provides an alternative to addressing the privacy issue through specially-designed learning algorithms (e.g. differentially private learning, Dwork et al. (2006)).

Also, data augmentation could go beyond traditional, generic transformation such as adding noise or paraphrasing, but be more goal-specific, such as compositional data augmentation for

compositional generalization (Andreas, 2020), and supervised data synthesis from pretrained language models to facilitate few-shot learning (Meng et al., 2022). In our view, data-centric NLP is a promising future direction that is complementary to model development.

Semi-parametric NLP

Semi-parametric NLP refers to the methods that involve retrieving from an external datastore, as described in § 2.2.3. We believe that this is a promising direction to acquire knowledge and improve training efficiency as shown in Borgeaud et al. (2021). The retrieve-and-predict mechanism also closely resembles how humans complete tasks – humans often acquire necessary information from search engines, books, or other material to complete various tasks such as writing or coding. Therefore, we doubt that all knowledge in the world should be stored into model parameters and think that semi-parametric methods are the way to go in NLP generally. That being said, we are enthusiastic about exploring retrieval-augmented approaches in various real-world applications. On the other hand, the retrieval overhead on both space and memory is the main limitation of the respective methods in practice, as we identify and try to solve in Part II. One future direction is to completely drop the bi-encoder retrieval paradigm, and directly generate the retrieved information through constrained decoding, dubbed "generative retrieval", which has been explored recently (De Cao et al., 2020; Tay et al., 2022).

Appendix of Chapter 3

.1 Sentiment Transfer Example Outputs

We list some examples of the sentiment transfer task in Table 1. Notably, the BT+NLL method tends to produce extremely short and simple sentences.

.2 Repetitive Examples of BT+NLL

In Section 3.5 we mentioned that the baseline BT+NLL has a low perplexity for some tasks because it tends to generate overly simple and repetitive sentences. From Table 3.1 we see that two representative tasks are sentiment transfer and formatliy transfer. In Appendix .1 we have demonstrated some examples for sentiment transfer, next we show some repetitive samples of BT+NLL in Table 2.

Table 1: Random Sentiment Transfer Examples

Methods	negative to positive
Original	the cake portion was extremely light and a bit dry .
UNMT	the cake portion was extremely light and a bit spicy .
BT+NLL	the cake portion was extremely light and a bit dry .
Ours	the cake portion was extremely light and a bit fresh .
Original	the “ chicken ” strip were paper thin oddly flavored strips .
UNMT	the “ chicken ” were extra crispy noodles were fresh and incredible .
BT+NLL	the service was great .
Ours	the “ chicken ” strip were paper sweet & juicy flavored .
Original	if i could give them a zero star review i would !
UNMT	if i could give them a zero star review i would !
BT+NLL	i love this place .
Ours	i love the restaurant and give a great review i would !
	positive to negative
Original	great food , staff is unbelievably nice .
UNMT	no , food is n’t particularly friendly .
BT+NLL	i will not be back .
Ours	no apologies , staff is unbelievably poor .
Original	my wife and i love coming here !
UNMT	my wife and i do n’t come here !
BT+NLL	i will not be back .
Ours	my wife and i walked out the last time .
Original	the premier hookah lounge of las vegas !
UNMT	the worst museum of las vegas !
BT+NLL	the worst frame shop of las vegas !
Ours	the hallways scam lounge of las vegas !

Table 2: Repetitive examples of BT+NLL baseline on Formality transfer.

Original	**Transferred**
formal to informal	
I like Rhythm and Blue music .	I like her and I don't know .
There's nothing he needs to change .	I don't know , but I don't know .
I enjoy watching my companion attempt to role @-@ play with them .	I don't know , but I don't know .
I am watching it right now .	I don't know , but I don't know .
That is the key point , that you fell asleep .	I don't know , but I don't know .
informal to formal	
its a great source just download it .	I do not know , but I do not know .
Happy Days , it was the coolest !	I do not know , but I do not know .
I used to play flute but once I started sax , I got hooked .	I do not know , but I do not know .
The word you are looking for is ..... strengths	The word you are looking for is :)
Plus you can tell she really cared about her crew .	Plus you can tell she really cared about her crew .

Appendix of Chapter 6

.3 Full Results on Different Bottleneck Dimensions

Table 3: Performance on the test sets of abstractive summarization (XSum) and WMT EN-RO translation.

Method	# params (%)	XSum (R-1/2/L)	MT BLEU
Modified Representation: attention			
Prefix Tuning, $r = 200$	3.6	43.40/20.46/35.51	35.6
Prefix Tuning, $r = 512$	9.2	43.29/20.40/35.37	35.1
LoRA, $r = 200$	7.2	43.09/20.29/35.37	36.2
Sequential Adapter, $r = 200$	3.6	42.01/19.30/34.40	35.3
Sequential Adapter, $r = 512$	9.2	41.05/18.87/33.71	34.7
Parallel Adapter, $r = 200$	3.6	43.58/20.31/35.34	35.6
Parallel Adapter, $r = 512$	9.2	43.99/20.83/35.77	36.2
Modified Representation: FFN			
LoRA, $r = 102$	6.1	44.59/21.31/36.25	36.5
Sequential Adapter, $r = 200$	2.4	43.21/19.98/35.08	35.6
Sequential Adapter, $r = 512$	6.1	43.72/20.75/35.64	36.3
Sequential Adapter, $r = 1024$	12.3	43.95/21.00/35.90	36.7
Parallel Adapter, $r = 200$	2.4	43.93/20.66/35.63	36.4
Parallel Adapter, $r = 512$	6.1	44.35/20.98/35.98	37.1
Parallel Adapter, $r = 1024$	12.3	44.53/21.24/36.23	37.3

Appendix of Chapter 7

.1 Derivations of Variational Inference and ELBO

.1.1 Derivation of optimal $q^*(\theta)$

Here we try to show that the optimal variational distribution over θ, $q^*_{\boldsymbol{\lambda}}(\theta)$, is a Dirichlet distribution and derive its optimal $\boldsymbol{\lambda}^*$ as given in Eq. 7.7. According to (Bishop, 2006) (Chapter 10.1.1), we have:

$$q^*_{\boldsymbol{\lambda}}(\theta) \propto \exp\Big(\mathbb{E}_{-\theta}[\log p(\{x_n, \mathbf{t}_n, \mathbf{z}_n\}_{n=1}^N, \theta)]\Big), \tag{1}$$

where $\mathbb{E}_{-\theta}$ denotes expectation over all latent variables except for θ. We expand Eq. 1 as:

$$\begin{aligned}
q^*_{\boldsymbol{\lambda}}(\theta) &\propto \exp\Big(\mathbb{E}_{-\theta}[\log p(\{x_n, \mathbf{t}_n, \mathbf{z}_n\}_{n=1}^N, \theta)]\Big) \\
&\propto \exp\Big(\mathbb{E}_{-\theta}[\log p_\alpha(\theta) + \sum_n \log p(\mathbf{t}_n|\theta)]\Big) \\
&\propto \exp\Big(\mathbb{E}_{-\theta}[\sum_k \log \theta_k^{\alpha-1} + \sum_n \log \prod_k \theta_k^{\mathbb{1}(\mathbf{t}_n = x_k)}]\Big) \\
&\propto \exp\Big(\mathbb{E}_{-\theta}[\sum_k \log \theta_k^{\alpha-1} + \sum_n \sum_k \mathbb{1}(\mathbf{t}_n = x_k) \log \theta_k]\Big) \\
&\propto \exp\Big(\sum_k \log \theta_k^{\alpha-1} + \sum_n \sum_k q(\mathbf{t}_n = x_k|x_n) \log \theta_k\Big) \\
&\propto \prod_k \theta_k^{\alpha-1+\sum_n q(\mathbf{t}_n = x_k|x_n)},
\end{aligned} \tag{2}$$

where $\mathbb{1}(\cdot)$ is the indicator function. We conclude that $q^*_{\boldsymbol{\lambda}}(\theta)$ has the form of Dirichlet distribution and the optimal Dirichlet parameter $\lambda^*_k = \alpha + \sum_n q(\mathbf{t}_n = x_k|x_n)$.

.1.2 Derivation of Three KL Divergence Terms

There are three KL divergence terms in our training objective ELBO (Eq. 7.4). Now we show that all three KL divergence terms can be computed exactly and efficiently at training time and we

derive their expressions respectively:

(1). $\mathbb{E}_{q(\mathbf{t}_n|\mathbf{x}_n)}[D_{\mathrm{KL}}(q(\mathbf{z}_n|\mathbf{t}_n,\mathbf{x}_n)||p(\mathbf{z}_n))]$**:** As shown in (Xu and Durrett, 2018), the KL divergence between any vMF distribution with fixed concentration parameter and a uniform vMF distribution is a constant:

$$\begin{aligned} D_{\mathrm{KL}}(\text{vMF}(\mu,\kappa)||\text{vMF}(\cdot,0)) = \kappa\frac{I_{d/2}(\kappa)}{I_{d/2-1}(\kappa)} + (\frac{d}{2}-1)\log\kappa - \frac{d}{2}\log(2\pi) \\ - \log I_{d/2-1}(\kappa) + \frac{d}{2}\log\pi + \log 2 - \log\Gamma(\frac{d}{2}), \end{aligned} \tag{3}$$

where d is the number of dimensions, I_v stands for the modified Bessel function of the first kind at order v. Therefore,

$$\mathbb{E}_{q(\mathbf{t}_n|\mathbf{x}_n)}[D_{\mathrm{KL}}(q(\mathbf{z}_n|\mathbf{t}_n,\mathbf{x}_n)||p(\mathbf{z}_n))] = D_{\mathrm{KL}}(\text{vMF}(\mu,\kappa)||\text{vMF}(\cdot,0)) = const \tag{4}$$

(2). $\mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[D_{\mathrm{KL}}(q(\mathbf{t}_n|\mathbf{x}_n)||p(\mathbf{t}_n|\theta))]$**:**

$$\begin{aligned} &\mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[D_{\mathrm{KL}}(q(\mathbf{t}_n|\mathbf{x}_n)||p(\mathbf{t}_n|\theta))] \\ &= \mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}\Big[\mathbb{E}_{q(\mathbf{t}_n|\mathbf{x}_n)}\big[\log q(\mathbf{t}_n|\mathbf{x}_n) - \log p(\mathbf{t}_n|\theta)\big]\Big] \\ &= \mathbb{E}_{q(\mathbf{t}_n|\mathbf{x}_n)}\log q(\mathbf{t}_n|\mathbf{x}_n) - \mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}\mathbb{E}_{q(\mathbf{t}_n|\mathbf{x}_n)}\big[\log\prod_k \theta_k^{\mathbb{1}(\mathbf{t}_n=x_k)}\big] \\ &= \sum_k q(\mathbf{t}_n|\mathbf{x}_n)\log q(\mathbf{t}_n|\mathbf{x}_n) - \mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}\mathbb{E}_{q(\mathbf{t}_n|\mathbf{x}_n)}\big[\sum_k \mathbb{1}(\mathbf{t}_n=x_k)\log\theta_k\big] \\ &= \sum_k q(\mathbf{t}_n=\mathbf{x}_k|\mathbf{x}_n)\log q(\mathbf{t}_n=\mathbf{x}_k|\mathbf{x}_n) - \sum_k q(\mathbf{t}_n=\mathbf{x}_k|\mathbf{x}_n)\mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}\log\theta_k \\ &\overset{(i)}{=} \sum_k q(\mathbf{t}_n=\mathbf{x}_k|\mathbf{x}_n)\log q(\mathbf{t}_n=\mathbf{x}_k|\mathbf{x}_n) - \sum_k q(\mathbf{t}_n=\mathbf{x}_k|\mathbf{x}_n)[\Psi(\lambda_k) - \Psi(\sum_i^N \lambda_i)], \end{aligned} \tag{5}$$

where $\Psi(\cdot)$ is the digamma function, and step (i) computes the expectation of $\log\theta_k$ over Dirichlet variable θ by using the general fact that the derivative of the log normalization factor with respect to the natural parameter is equal to the expectation of the sufficient statistic.

(3). $D_{\mathrm{KL}}(q_{\boldsymbol{\lambda}}(\theta)||p_{\alpha}(\theta))$**:**

$$
\begin{aligned}
D_{\mathrm{KL}}(q_{\boldsymbol{\lambda}}(\theta)||p_{\alpha}(\theta)) &= \mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}\big[\log q_{\boldsymbol{\lambda}}(\theta) - \log p_{\alpha}(\theta)\big] \\
&= \mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}\big[\log\big(\prod_k \theta_k^{\lambda_k - 1}/B(\boldsymbol{\lambda})\big) - \log\big(\prod_k \theta_k^{\alpha-1}/B(\alpha)\big)\big] \\
&= -\log B(\boldsymbol{\lambda}) + \sum_k (\lambda_k - 1)[\Psi(\lambda_k) - \Psi(\sum_i^N \lambda_i)] \\
&+ \log B(\alpha) - \sum_k (\alpha - 1)[\Psi(\lambda_k) - \Psi(\sum_i^N \lambda_i)],
\end{aligned} \tag{6}
$$

where $B(\cdot)$ is the multivariate beta function and $B(\boldsymbol{\lambda})$ is the normalization factor for Dirichlet distribution parameterized by $\boldsymbol{\lambda}$.

.2 Experimental Details

On MSCOCO dataset, we use a single-layer attentional LSTM seq2seq architecture with word embedding size 100 and hidden state size 400 as $p_{\gamma}(\mathbf{x}|\mathbf{t}, \mathbf{z})$, the latent edit vector dimension is 50. This configuration follows the hyperparameters for vMF-VAE (Xu and Durrett, 2018). On Yelp Medium and Yelp Large datasets, we follow (Guu et al., 2018) to use a three-layer attentional LSTM seq2seq architecture as $p_{\gamma}(\mathbf{x}|\mathbf{t}, \mathbf{z})$ with word embedding size 300 and hidden state size 256, the edit vector dimension is 128. Skip connections are also used between adjacent LSTM layers. In the inverse editor $q_{\phi_{z|t,x}}(\mathbf{z}|\mathbf{t}, \mathbf{x})$, we use a single-layer LSTM to encode three sequences – the aligned prototype, aligned data example, and the edit operation sequence, of which the word embedding size for text sequences and the hidden state size are the same as in $p_{\gamma}(\mathbf{x}|\mathbf{t}, \mathbf{z})$, and the word embedding size for edit operation sequence is 10 (since the vocobulary size of edit operations is very small). Across all datasets, we initialize word embeddings in $p_{\gamma}(\mathbf{x}|\mathbf{t}, \mathbf{z})$ (both encoder and decoder sides) and $q_{\phi_{z|t,x}}(\mathbf{z}|\mathbf{t}, \mathbf{x})$ with GloVe word embeddings (Pennington et al., 2014) following (Guu et al., 2018). All NLM baselines use the same architecture as $p_{\gamma}(\mathbf{x}|\mathbf{t}, \mathbf{z})$ in our model for a fair comparison.

With respect to hyperparameter tuning, we tune the temperature parameter μ in the prototype retriever $q(\mathbf{t}|\mathbf{x})$ on the MSCOCO validation data in the range of $\{0.1, 0.3, 0.5, 0.7, 0.9, 1.0\}$, and set as 0.3 for all datasets. The concentration parameter κ of the vMF distribution is tuned in the range of $\{30, 40, 50\}$ for all datasets. κ is set as 30 for MSCOCO and Yelp Medium and 40 for Yelp Large. We run different α as $\{0.1, 0.3, 0.5, 0.7, 0.9, 1, 10\}$ for each dataset to obtain prototype set with varying sparsity. On MSCOCO dataset we also add an additional

run with $\alpha = 0.2$ since 0.5 is a large value on this dataset already (90% of training examples are selected as the prototype set when $\alpha = 0.5$). We apply annealing and free-bits techniques following (Li et al., 2019) to the KL term on prototype variable, $\mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[D_{\mathrm{KL}}(q(\mathbf{t}_n|\mathbf{x}_n)||p(\mathbf{t}_n|\theta))]$, to mitigate posterior collapse. Specifically, $\mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[D_{\mathrm{KL}}(q(\mathbf{t}_n|\mathbf{x}_n)||p(\mathbf{t}_n|\theta))]$ in our training objective becomes $\beta \cdot \max\{\mathbb{E}_{q_{\boldsymbol{\lambda}}(\theta)}[D_{\mathrm{KL}}(q(\mathbf{t}_n|\mathbf{x}_n)||p(\mathbf{t}_n|\theta))], c\}$ in practice. This objective means that we can downweight this KL term with $\beta < 1$ and optimize it only when this KL is larger than a threshold value c. We increase β from 0 to 1 linearly in the first m epochs (annealing). m is tuned in the range of $\{5, 10\}$ for MSCOCO and $\{1, 2, 3\}$ for Yelp Medium and Yelp Large.[1] c is tuned in the range of $\{5, 6, 8\}$. To obtain the reported results in Section 7.4, m is set as 5 for MSCOCO, 2 for Yelp Medium and 3 for Yelp Large. c is set as 5 for MSCOCO, 6 for Yelp Medium and 8 for Yelp Large. We use Adam (Kingma and Ba, 2015) to optimize the training objective with learning rate 0.001.

.3 Qualitative Results on Interpolation

As in Section 7.4.3, here we show more generated examples through interpolation on MSCOCO dataset.

[1] It is unreasonable to set a large m on Yelp dataset since there are tens of thousands update steps per epoch on Yelp, the annealinig process would be too slow if m is large which usually hurts the language modeling performance (He et al., 2019).

Table 4: Qualitative examples from the MSCOCO dataset on interpolated sentence generation given the prototype. For each example, the first row is the given prototype, the second-row and the last-row sentences are obtained by sampling edit vectors from the prior, the rest three sentences are generated by interpolating between the two edit vectors.

Prototype: a horse drawn carriage on the side of a city street	Prototype: A baseball pitcher on the mound having just threw a pitch
Two horses drawn carriage on a city street	a baseball player swinging a bat at home plate
Two horses standing next to each other on a city street	a man about to hit a ball with his bat
Two horses on the side of a city street	a man swinging a bat at the ball during a game
Two horses on the side of a city street	a person swinging a bat at the ball during a game
A brown and white horse drawn carriage on a city street	A person swinging a bat during a baseball game
Prototype: A man walking on the beach carrying a surfboard	**Prototype: A group of people are raising an umbrella on a beach**
Two people standing next to each other on a beach	A group of people are walking on the beach with umbrellas
A person standing on the beach holding a surfboard	A group of people are walking on the beach next to each other
A man walking along the beach with a surfboard	A group of people are walking on the beach with umbrellas
A man walking on the beach with a surfboard	A group of people are holding umbrellas on the beach
A young man walking on the beach with a surfboard	A group of people are walking on the beach
Prototype: there is a white truck that is driving on the road	**Prototype: A couple of bags of luggage sitting up against a wall**
there are many cows that are standing in the dirt	A large pile of luggage sitting on top of a wall
there are many cows that are standing in the dirt	A pile of luggage sitting on top of a wall
the truck is driving down the road in the rain	Two bags of luggage sitting on the ground
this truck is driving down the road in the rain	Two bags of luggage sitting in a room
This truck is pulled up to the side of the road	A couple of bags of luggage on a wooden floor
Prototype: A man riding a sailboat in the ocean next to a shore	**Prototype: A beer bottle sitting on a bathroom sink next to a mirror**
A man on a boat in a body of water	A white cell phone sitting next to a toilet in a bathroom
A man riding a boat on a body of water	A white bottle of wine sitting next to a toilet
A man riding a boat in a body of water	A glass of wine sitting next to a toilet in a bathroom
A man riding a small boat on a body of water	A pair of scissors is placed next to a toilet
A man riding a wave on top of a boat	A pair of scissors sitting next to each other on a toilet
Prototype: A little boy sitting on a mattress holding a stuffed animal	**Prototype: A giraffe has its nose pressed against the trunk of a tree**
A little girl playing with a stuffed animal	Two giraffes look at a wire fence to eat
A little girl playing with a stuffed animal	Two giraffes look at a fence to eat
A little boy holding a stuffed animal in his mouth	a couple of giraffes are standing by a fence
A little girl sitting on a bed with stuffed animals	a close up of a giraffe is eating a carrot
A little girl sitting on a bed with stuffed animals	a close up of a giraffe has its mouth open

Bibliography

Daniel Adiwardana, Minh-Thang Luong, David R So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, et al. 2020. Towards a human-like open-domain chatbot. *arXiv preprint arXiv:2001.09977*. 2.1.1

Roee Aharoni and Yoav Goldberg. 2020. Unsupervised domain clusters in pretrained language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7747–7763, Online. Association for Computational Linguistics. 8.3.1

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3874–3884, Minneapolis, Minnesota. Association for Computational Linguistics. 2.2.1

Farhad Akhbardeh, Arkady Arkhangorodsky, Magdalena Biesialska, Ondřej Bojar, Rajen Chatterjee, Vishrav Chaudhary, Marta R. Costa-jussa, Cristina España-Bonet, Angela Fan, Christian Federmann, Markus Freitag, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Leonie Harter, Kenneth Heafield, Christopher Homan, Matthias Huck, Kwabena Amponsah-Kaakyire, Jungo Kasai, Daniel Khashabi, Kevin Knight, Tom Kocmi, Philipp Koehn, Nicholas Lourie, Christof Monz, Makoto Morishita, Masaaki Nagata, Ajay Nagesh, Toshiaki Nakazawa, Matteo Negri, Santanu Pal, Allahsera Auguste Tapo, Marco Turchi, Valentin Vydrin, and Marcos Zampieri. 2021. Findings of the 2021 conference on machine translation (WMT21). In *Proceedings of the Sixth Conference on Machine Translation*, pages 1–88, Online. Association for Computational Linguistics. 1

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2019. Character-level language modeling with deeper self-attention. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in*

Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pages 3159–3166. AAAI Press. 7.1, 8.1

Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2019. Quicker ADC: Unlocking the hidden potential of product quantization with simd. *IEEE transactions on pattern analysis and machine intelligence*. 8.2

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566, Online. Association for Computational Linguistics. 9

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48. 9

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2019. An effective approach to unsupervised machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 194–203, Florence, Italy. Association for Computational Linguistics. 3.1, 3.4

Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. 2017. Unsupervised neural machine translation. *arXiv preprint arXiv:1710.11041*. 2.2.1

Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. 2018. Unsupervised neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. 3.4

Mikel Artetxe and Holger Schwenk. 2019. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610. 2.2.1

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer. 2.2.3

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *ArXiv preprint*, abs/1607.06450. 6.2.1

Stephen H. Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey,

Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged S. Al-shaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Xiangru Tang, Mike Tian-Jian Jiang, and Alexander M. Rush. 2022. Promptsource: An integrated development environment and repository for natural language prompts. 5.1, 5.3.1

Philip Bachman, Ouais Alsharif, and Doina Precup. 2014. Learning with pseudo-ensembles. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3365–3373. 5.1

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. 8.3.1

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 1, 2.1.1, 2.1.2, 3.1, 3.2, 3.3.1, 7.1, 8.1

Colin Bannard and Chris Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 597–604, Ann Arbor, Michigan. Association for Computational Linguistics. 3.1

Ankur Bapna and Orhan Firat. 2019. Non-parametric adaptation for neural machine translation. In *Proceedings of NAACL*. 7.1

Lisa Bauer, Yicheng Wang, and Mohit Bansal. 2018. Commonsense for generative multi-hop question answering tasks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4220–4230, Brussels, Belgium. Association for Computational Linguistics. 2.2.3

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv e-prints*, pages arXiv–2106. 2.2.2, 6.2.2, 6.4.2, 6.7

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155. 7.1

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradi-

ents through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*. 2.2.2

Lucas Beyer, Xiaohua Zhai, Avital Oliver, and Alexander Kolesnikov. 2019. S4L: self-supervised semi-supervised learning. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1476–1485. IEEE. 5.1

Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer. .1.1

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, page 92–100, New York, NY, USA. Association for Computing Machinery. 4.6

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, Berlin, Germany. Association for Computational Linguistics. 6.4.1

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. 2.2.3

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2021. Improving language models by retrieving from trillions of tokens. *arXiv preprint arXiv:2112.04426*. 1, 2.2.3, 2.3, 9

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, Berlin, Germany. Association for Computational Linguistics. 2.1.2, 2, 7.3.1, 7.3.2

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz

Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 1, 2.1, 2.1.2, 2.2.1, 2.2.2, 5.1, 5.2, 5.4.1, 6.1, 8.1

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. 2015. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*. 7.4.1

Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. 2010. *Semi-Supervised Learning*, 1st edition. The MIT Press. 4.1

Olivier Chapelle and Alexander Zien. 2005. Semi-supervised classification by low density separation. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, AISTATS 2005, Bridgetown, Barbados, January 6-8, 2005*. Society for Artificial Intelligence and Statistics. 4.2

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*. 2.1.1

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics. 2.2.3

Jiaao Chen, Zichao Yang, and Diyi Yang. 2020a. MixText: Linguistically-informed interpolation of hidden space for semi-supervised text classification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2147–2157, Online. Association for Computational Linguistics. 2.2.1

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020b. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846. 2.2.2

Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. 2021. EarlyBERT: Efficient BERT training via early-bird lottery tickets. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2195–2207, Online. Association for Computational Linguistics. 2.2.2

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. 2015. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*. 2.1.1

Xinlei Chen and Kaiming He. 2021. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758. 5.3.2

Zhiyu Chen, Harini Eavani, Wenhu Chen, Yinyin Liu, and William Yang Wang. 2020c. Few-shot NLG with pre-trained language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 183–190, Online. Association for Computational Linguistics. 2.2.1

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics. 2.1.2

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*. 2.2.2

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*. 2.2

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*. 2.2.2

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*. 2.1.2

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018a. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium. Association for Computational Linguistics. 4.6

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018b. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on*

Empirical Methods in Natural Language Processing, pages 1914–1925, Brussels, Belgium. Association for Computational Linguistics. 5.3.2, 5.3.2

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics. 2.2.1

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, Dublin, Ireland. Association for Computational Linguistics. 9

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *Advances in neural information processing systems*, 33:4271–4282. 2.2

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics. 7.1

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. 2020. Autoregressive entity retrieval. *arXiv preprint arXiv:2010.00904*. 2.2.3, 9

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*, volume 23, pages 107–124. 5.4.1

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*. 7.1, 7.3.2

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019a. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics. 1, 1.1, 2.1.2, 2.2.1, 2.2.2, 6.1, 8.1

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019b. BERT: Pre-

training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics. 5.1

Kaustubh D Dhole, Varun Gangal, Sebastian Gehrmann, Aadesh Gupta, Zhenhao Li, Saad Mahamood, Abinaya Mahendiran, Simon Mille, Ashish Srivastava, Samson Tan, et al. 2021. Nl-augmenter: A framework for task-sensitive natural language augmentation. *arXiv preprint arXiv:2112.02721*. 2.2.1

Emily Dinan, Stephen Roller, Kurt Shuster, Angela Fan, Michael Auli, and Jason Weston. 2019. Wizard of wikipedia: Knowledge-powered conversational agents. In *International Conference on Learning Representations*. 2.2.3

Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. 2015. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1723–1732, Beijing, China. Association for Computational Linguistics. 2.2.1

Qing Dou and Kevin Knight. 2013. Dependency-based decipherment for resource-limited machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1668–1676, Seattle, Washington, USA. Association for Computational Linguistics. 3.5.3

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2021. All NLP tasks are generation tasks: A general pretraining framework. *ArXiv preprint*, abs/2103.10360. 1

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer. 9

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium. Association for Computational Linguistics. 4.2, 4.5.1, 4.5.7

Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. 2021. Measuring and improving consistency in pretrained

language models. *Transactions of the Association for Computational Linguistics*, 9:1012–1031. 5.1, 5.3.2, 5.3.2

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. 2020. Learned step size quantization. In *International Conference on Learning Representations*. 2.2.2

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press. 8.4.2

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*. 2.2.2

A Farooq. 2000. Biologically inspired modular neural networks. *Unpublished doctoral dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA*. 9

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *ArXiv preprint*, abs/2101.03961. (document), 1, 2.2.2, 6.1

Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. 2021. A survey of data augmentation approaches for NLP. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 968–988, Online. Association for Computational Linguistics. 2.2.1

Yufei Feng, Xiaoyu Yang, Xiaodan Zhu, and Michael Greenspan. 2022. Neuro-symbolic natural logic with introspective revision for natural language inference. *Transactions of the Association for Computational Linguistics*, 10:240–256. 9

Thibault Févry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski. 2020. Entities as experts: Sparse memory access with entity supervision. *arXiv preprint arXiv:2004.07202*. 2.2.3

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. 2016. Multi-way, multilingual neural machine translation with a shared attention mechanism. *arXiv preprint arXiv:1601.01073*. 2.2.1

Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378. 5.3.3

Yao Fu and Yansong Feng. 2018. Natural answer generation with heterogeneous memory. 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 185–195, New Orleans, Louisiana. Association for Computational Linguistics. 2.2.3

Luyu Gao and Jamie Callan. 2021. Condenser: a pre-training architecture for dense retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 981–993, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. 2.2.3

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021a. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3816–3830, Online. Association for Computational Linguistics. 2.2.1

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021b. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. 2.2.3

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. Creating training corpora for NLG micro-planners. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 179–188, Vancouver, Canada. Association for Computational Linguistics. 2.1.1

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. 6.4.4

Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, page 518–529, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. 8.2

Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4852–4861. 2.2.2

Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative quantiza-

tion: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929. 10

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680. 2.1.2

Yves Grandvalet and Yoshua Bengio. 2004. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 529–536. 4.2

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2016. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*. 2.2.3

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018a. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*. 2.1.2, 2.2

Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. 2018b. Search engine guided neural machine translation. In *Proceedings of AAAI*. 2.2.3, 7.1, 7.2

Demi Guo, Yoon Kim, and Alexander Rush. 2020a. Sequence-level mixed sample data augmentation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5547–5552, Online. Association for Computational Linguistics. 2.2.1

Demi Guo, Alexander Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4884–4896, Online. Association for Computational Linguistics. 6.2.2, 6.4.2

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020b. Accelerating large-scale inference with anisotropic vector quantization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3887–3896. PMLR. 8.2

Kelvin Guu, Tatsunori B. Hashimoto, Yonatan Oren, and Percy Liang. 2018. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450. 1.1, 2.2.3, 7, 7.1, 7.2, 7.2, 7.3.1, 7.3.2, 7.3.2, 7.3.2, 7.4.1, 7.1, 7.4.2, 5, 6, 7.4.3, 8.1, .2

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. *ArXiv preprint*, abs/2002.08909. 2.2.3

Francisco Guzmán, Peng-Jen Chen, Myle Ott, Juan Pino, Guillaume Lample, Philipp Koehn, Vishrav Chaudhary, and Marc'Aurelio Ranzato. 2019. The FLORES evaluation datasets for low-resource machine translation: Nepali–English and Sinhala–English. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6098–6111, Hong Kong, China. Association for Computational Linguistics. 4.5.2

Thanh-Le Ha, Jan Niehues, and Alexander Waibel. 2016. Toward multilingual neural machine translation with universal encoder and decoder. *arXiv preprint arXiv:1611.04798*. 2.2.1

Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy S Liang. 2018. A retrieve-and-edit framework for predicting structured outputs. In *Proceedings of NeurIPS*. 7.1

Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. 2018. Retrieval-based neural code generation. In *Proceedings EMNLP*. 7.1

Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. 2016. Dual learning for machine translation. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 820–828. 3.4

Junxian He, Taylor Berg-Kirkpatrick, and Graham Neubig. 2020a. Learning sparse prototypes for text generation. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 8.1

Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. 2020b. Revisiting self-training for neural sequence generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. 5.4.2

Junxian He, Daniel Spokoyny, Graham Neubig, and Taylor Berg-Kirkpatrick. 2019. Lagging inference networks and posterior collapse in variational autoencoders. In *Proceedings of ICLR*.

7.3.1, 1

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*. 5.1, 5.3.3

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28. 1

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *ArXiv preprint*, abs/1503.02531. 2.2.2, 5.3.2

Geoffrey E Hinton. 2002. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800. 2.1.2

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv preprint*, abs/1207.0580. 4.1, 4.3.3

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780. 1, 2.1.2, 7.3.1, 8.4.1

Micah Hodosh, Peter Young, and Julia Hockenmaier. 2013. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899. 7.2

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. 2013. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347. 7.3.2

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR. 1.1, 2.2.2, 5.1, 5.3.3, 6.1, 6.2.2, 6.2.2, 6.4.4

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*. 2.2.2, 5.3.3, 5.4.1, 6.1, 6.2.2, 3

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. 2017. Toward

controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR. 3.1, 3.1, 3.5.4

Shaoyi Huang, Dongkuan Xu, Ian EH Yen, Sung-en Chang, Bingbing Li, Shiyang Chen, Mimi Xie, Hang Liu, and Caiwen Ding. 2021. Sparse progressive distillation: Resolving overfitting under pretrain-and-finetune paradigm. *arXiv preprint arXiv:2110.08190*. 2.2.2

Zhongqiang Huang and Mary Harper. 2009. Self-training PCFG grammars with latent annotations across languages. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 832–841, Singapore. Association for Computational Linguistics. 4.1

Drew Hudson and Christopher D Manning. 2019. Learning by abstraction: The neural state machine. *Advances in Neural Information Processing Systems*, 32. 9

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Towards unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*. 2.2.3

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics. 1, 2.2.3

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. 3.3.2, 3.5.6

Harsh Jhamtani, Varun Gangal, Eduard Hovy, and Eric Nyberg. 2017. Shakespearizing modern language using copy-enriched sequence to sequence models. In *Proceedings of the Workshop on Stylistic Variation*, pages 10–19, Copenhagen, Denmark. Association for Computational Linguistics. 3.5.3

Haozhe Ji, Pei Ke, Shaohan Huang, Furu Wei, Xiaoyan Zhu, and Minlie Huang. 2020. Language generation with multi-hop reasoning on commonsense knowledge graph. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 725–736, Online. Association for Computational Linguistics. 2.2.3

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*,

8:423–438. 5.1, 5.3.2, 5.4.2

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics. 2.2.2

Alexander Johansen and Richard Socher. 2017. Learning when to skim and when to read. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 257–264, Vancouver, Canada. Association for Computational Linguistics. 8.4.1

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547. 8.2

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017a. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351. 2.2.1

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017b. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351. 3.3.2

Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128. 8.2

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*. 2.1.2

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*. 2.3

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics. 2.2.3

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*. 2.1.1

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2021. Nearest neighbor machine translation. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. 1, 2.2.3, 7.1, 8.1, 8.2, 8.4.2

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. 1, 1.1, 2.2.3, 7.1, 8, 8.1, 8.2, 2, 8.2, 8.3.1, 8.3.2, 7, 8.1

Byeongchang Kim, Jaewoo Ahn, and Gunhee Kim. 2020. Sequential latent knowledge selection for knowledge-grounded dialogue. In *International Conference on Learning Representations*. 2.2.3

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics. 3.5.3

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics. 2.2.2, 5.3.2

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 4.5.1, 5.4.1, 8.4.1, .2

Diederik P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. 2014. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3581–3589. 2.2.1, 4.1

Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2.1.2, 2.2.1, 3.1, 3.3.2, 3.3.2, 7.1, 7.3.2

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. 2016. Improved variational inference with inverse autoregressive flow. In *Proceedings of NeurIPS*. 7.3.2

Kevin Knight, Anish Nair, Nishit Rathod, and Kenji Yamada. 2006. Unsupervised analysis for decipherment problems. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 499–506, Sydney, Australia. Association for Computational Linguistics. 3.1, 3.2

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver. Association for Computational Linguistics. 8.3.1

Ashutosh Kumar, Satwik Bhattamishra, Manik Bhandari, and Partha Talukdar. 2019. Submodular optimization-based diverse paraphrasing and its effectiveness in data augmentation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3609–3619, Minneapolis, Minnesota. Association for Computational Linguistics. 2.2.1

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of EACL*. 7.1

Samuli Laine and Timo Aila. 2017. Temporal ensembling for semi-supervised learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. 4.6

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*. 2.2.1

Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018a. Unsuperunsupervised neural machine translationvised machine translation using monolingual corpora only. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. 2.2.1, 3.1, 3.5.1

Guillaume Lample, Myle Ott, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018b. Phrase-based & neural unsupervised machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5039–5049, Brussels, Belgium. Association for Computational Linguistics. 3.1, 3.3.2, 3.4, 5, 3.5.2, 4.4.3, 4.5.4

Guillaume Lample, Sandeep Subramanian, Eric Michael Smith, Ludovic Denoyer, Marc'Aurelio Ranzato, and Y-Lan Boureau. 2019. Multiple-attribute text rewriting. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

OpenReview.net. 3.1, 3.4, 3.5, 3.5.1, 3.5.3

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR. 2.2.1

Teven Le Scao and Alexander Rush. 2021. How many data points is a prompt worth? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2627–2636, Online. Association for Computational Linguistics. 2.2.1

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. 2006. A tutorial on energy-based learning. *Predicting structured data*, 1(0). 2.1.2

Dong-Hyun Lee. 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*. 4.2

Jinhyuk Lee, Mujeen Sung, Jaewoo Kang, and Danqi Chen. 2021. Learning dense representations of phrases at scale. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6634–6647, Online. Association for Computational Linguistics. 2.2.3

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, Dublin, Ireland. Association for Computational Linguistics. 9

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096, Florence, Italy. Association for Computational Linguistics. 2.2.3

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. 1.1, 2.2.2, 6.1, 6.2.2, 6.7

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. 5.4.1

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020a. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics. 1, 2.1.2, 6.1, 6.4.1, 6.7

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020b. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 2.2.3

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. 5.4.1

Bohan Li, Junxian He, Graham Neubig, Taylor Berg-Kirkpatrick, and Yiming Yang. 2019. A surprisingly effective fix for deep latent variable modeling of text. In *Proceedings of EMNLP*. 7.3.2, .2

Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. 2020. On the sentence embeddings from pre-trained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9119–9130, Online. Association for Computational Linguistics. 2.2.3

Juncen Li, Robin Jia, He He, and Percy Liang. 2018. Delete, retrieve, generate: a simple approach to sentiment and style transfer. 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 1865–1874, New Orleans, Louisiana. Association for Computational Linguistics. 3.5.3

Junyi Li, Tianyi Tang, Wayne Xin Zhao, and Ji-Rong Wen. 2021a. Pretrained language models for text generation: A survey. *arXiv preprint arXiv:2105.10311*. 2.1.1

Xiang Lisa Li and Percy Liang. 2021a. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics. 1.1, 6.1, 6.2.2, 6.4.4, 6.7

Xiang Lisa Li and Percy Liang. 2021b. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics. 2.2.2, 5.3.3

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. 2021b. {BRECQ}: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*. 2.2.2

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics. 4.5.3, 6.4.1

Chin-Yew Lin and Franz Josef Och. 2004. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of COLING*. 7.4.1

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *Proceedings of ECCV*. 7.1, 7.4.1

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535. 7.1

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021a. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ArXiv preprint*, abs/2107.13586. 1, 1.1, 2.1, 5.1, 5.2, 6.1, 6.2.2

Qi Liu, Dani Yogatama, and Phil Blunsom. 2022. Relational memory augmented language models. *arXiv preprint arXiv:2201.09680*. 2.2.3

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021b. Gpt understands, too. *ArXiv preprint*, abs/2103.10385. 2.2.1, 6.2.2

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020a. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742. 2.1.2

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020b. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742. 2.2.1, 6.4.1, 6.7

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019a. RoBERTa: A robustly optimized bert pretraining approach. *ArXiv preprint*, abs/1907.11692. 1, 2.1.2, 5.1, 6.4.1, 7.1

Zhibin Liu, Zheng-Yu Niu, Hua Wu, and Haifeng Wang. 2019b. Knowledge aware conversation generation with explainable reasoning over augmented graphs. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1782–1792, Hong Kong, China. Association for Computational Linguistics. 2.2.3

Shayne Longpre, Yi Lu, Zhucheng Tu, and Chris DuBois. 2019. An exploration of data augmentation and sampling techniques for domain-agnostic question answering. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 220–227, Hong Kong, China. Association for Computational Linguistics. 2.2.1

Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. 2021. Luna: Linear unified nested attention. *Advances in Neural Information Processing Systems*, 34. 2.2

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. 2019. FlowSeq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4282–4292, Hong Kong, China. Association for Computational Linguistics. 2.1.2, 2.2

Andrea Madotto, Chien-Sheng Wu, and Pascale Fung. 2018. Mem2Seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*,

pages 1468–1478, Melbourne, Australia. Association for Computational Linguistics. 2.2.3

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *Proceedings of NeurIPS*. 6.3.2, 6.4.2

Rabeeh Karimi Mahabadi, Luke Zettlemoyer, James Henderson, Marzieh Saeidi, Lambert Mathias, Veselin Stoyanov, and Majid Yazdani. 2022. Perfect: Prompt-free and efficient few-shot learning with language models. *arXiv preprint arXiv:2204.01172*. 2.2.1

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. 2019. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations*. 9

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330. 2.1.1

Pedro Henrique Martins, Zita Marinho, and André FT Martins. 2022. Efficient machine translation domain adaptation. *arXiv preprint arXiv:2204.12608*. 2.2.3

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. *Advances in neural information processing systems*, 30. 2.2.1

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159, New York City, USA. Association for Computational Linguistics. 4.1

Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. 2022. Generating training data with language models: Towards zero-shot language understanding. *arXiv preprint arXiv:2202.04538*. 9

Yuxian Meng, Xiaoya Li, Xiayu Zheng, Fei Wu, Xiaofei Sun, Tianwei Zhang, and Jiwei Li. 2021. Fast nearest neighbor machine translation. *arXiv preprint arXiv:2105.14528*. 2.2.3

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *Proceedings of ICLR*. 7.1

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. 2.1.1, 8.1, 8.3.1

Yishu Miao and Phil Blunsom. 2016. Language as a latent variable: Discrete generative models for sentence compression. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 319–328, Austin, Texas. Association for Computational Linguistics. 2.2.1, 3.1, 3.3.2, 2, 3.5.6, 4.1

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32. 2.2.2

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. 2.2.1

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*. 7.1, 8.1

Sewon Min, Jordan Boyd-Graber, Chris Alberti, Danqi Chen, Eunsol Choi, Michael Collins, Kelvin Guu, Hannaneh Hajishirzi, Kenton Lee, Jennimaria Palomaki, et al. 2021. Neurips 2020 efficientqa competition: Systems, analyses and lessons learned. *ArXiv preprint*, abs/2101.00133. 2.2.3, 8.4.2

Remi Mir, Bjarke Felbo, Nick Obradovich, and Iyad Rahwan. 2019. Evaluating style transfer for text. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 495–504, Minneapolis, Minnesota. Association for Computational Linguistics. 3.5.3

Takeru Miyato, Andrew M. Dai, and Ian J. Goodfellow. 2017. Adversarial training methods for semi-supervised text classification. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. 4.6

Takeru Miyato, Shin-Ichi Maeda, Masanori Koyama, and Shin Ishii. 2019. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1979–1993. 4.6, 5.1, 5.3.2

Masahiro Mizukami, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Linguistic individuality transformation for spoken language. In *Natural Language Dialog Systems and Intelligent Assistants*. Springer. 3.1

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for

deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California. Association for Computational Linguistics. 5.4.1

Marius Muja and David G Lowe. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of the International Conference on Computer Vision Theory and Applications*. 8.2

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR. 2.2.2

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*. 2.2.3

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics. 1, 6.2, 6.4.1

Graham Neubig, Zi-Yi Dou, Junjie Hu, Paul Michel, Danish Pruthi, and Xinyi Wang. 2019. compare-mt: A tool for holistic comparison of language generation systems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 35–41, Minneapolis, Minnesota. Association for Computational Linguistics. 3.5.5

Nathan Ng, Kyunghyun Cho, and Marzyeh Ghassemi. 2020. SSMBA: Self-supervised manifold based data augmentation for improving out-of-domain robustness. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1268–1283, Online. Association for Computational Linguistics. 2.2.1

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR's WMT19 news translation task submission. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 314–319, Florence, Italy. Association for Computational Linguistics. 8.3.1

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of*

the 58th Annual Meeting of the Association for Computational Linguistics, pages 4885–4901, Online. Association for Computational Linguistics. 5.4.1

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206, Saarbrücken, Germany. Association for Computational Linguistics. 1, 2.1.1

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics. 4.3.1, 4.5.1, 7.4.1

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE. 2.1.1

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 of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics. 4.3.1, 6.4.1

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186, Online. Association for Computational Linguistics. 2.1.1

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*. 2.1.2

Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujun Li, Jinchao Li, Michael Zeng, and Jianfeng Gao. 2020. Few-shot natural language generation for task-oriented dialog. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 172–182, Online. Association for Computational Linguistics. 2.2.1

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of EMNLP*. 2.2.1, .2

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. 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 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics. 1, 1.1, 2.1.2, 2.2.1, 6.1, 8.1

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. 2021. KILT: a benchmark for knowledge intensive language tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2523–2544, Online. Association for Computational Linguistics. 2.2.3

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics. 1, 7.1

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics. 6.1, 6.2.2, 6.3.2, 6.7

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1267–1273, Minneapolis, Minnesota. Association for Computational Linguistics. 5.4.1

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics. 2.2.1

Nima Pourdamghani and Kevin Knight. 2017. Deciphering related languages. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2513–2518, Copenhagen, Denmark. Association for Computational Linguistics. 3.1, 3.5.3

Sai Prasanna, Anna Rogers, and Anna Rumshisky. 2020. When BERT Plays the Lottery, All Tickets Are Winning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural*

Language Processing (EMNLP), pages 3208–3229, Online. Association for Computational Linguistics. 2.2.2

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of ACL (Demo Track)*. 4

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying LMs with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212, Online. Association for Computational Linguistics. 5.4.2

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*. 6.1

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. 2017. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*. 9

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9. 1, 2.1.2, 2.2.1, 6.1, 8.1

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*. 1

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*. 1, 2.1.2, 5.1, 5.4.1, 6.4.2

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*. 2.1.2

Sudha Rao and Joel Tetreault. 2018. Dear sir or madam, may I introduce the GYAFC dataset: Corpus, benchmarks and metrics for formality style transfer. 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 129–140, New Orleans, Louisiana. Association for Computational Linguistics. 3.1, 3.2, 3.5.3

Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. 2015. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3546–3554. 4.6

Sujith Ravi and Kevin Knight. 2011. Deciphering foreign language. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 12–21, Portland, Oregon, USA. Association for Computational Linguistics. 3.1

Roi Reichart and Ari Rappoport. 2007. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 616–623, Prague, Czech Republic. Association for Computational Linguistics. 4.1

Nils Reimers and Iryna Gurevych. 2019a. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2.2.3

Nils Reimers and Iryna Gurevych. 2019b. Sentence-BERT: Sentence embeddings using siamese bert-networks. In *Proceedings of EMNLP*. 7.4.3

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87. 2.1

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. {ZeRO-Offload}: Democratizing {Billion-Scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 551–564. 5.4.1

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. 9

Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc. 2.2.3

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*. 5.4.1

Ronald Rosenfeld, Stanley F Chen, and Xiaojin Zhu. 2001. Whole-sentence exponential language models: a vehicle for linguistic-statistical integration. *Computer Speech & Language*, 15(1):55–73. 2.1.2

Sebastian Ruder, Ivan Vulić, and Anders Søgaard. 2019. A survey of cross-lingual word embedding models. *Journal of Artificial Intelligence Research*, 65:569–631. 2.2.1

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal. Association for Computational Linguistics. 1, 2.1.1, 3.1, 3.3.1, 4.5.3, 7.1, 8.1

Gözde Gül Şahin and Mark Steedman. 2018. Data augmentation via dependency tree morphing for low-resource languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5004–5009, Brussels, Belgium. Association for Computational Linguistics. 2.2.1

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. Poor man's bert: Smaller and faster transformer models. *arXiv preprint arXiv:2004.03844*. 2.2.2

Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. 2016. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1163–1171. 5.1

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*. 2.2.2

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*. 1, 2.2.1, 5.1, 5.1, 5.2, 5.4.1, 5.4.1, 6, 5.4.3, 5.5

Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389. 2.2.2

Timo Schick and Hinrich Schütze. 2021a. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics. 2.2.1

Timo Schick and Hinrich Schütze. 2021b. Generating datasets with pretrained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6943–6951, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. 2.2.1

Timo Schick and Hinrich Schütze. 2021c. It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352, Online. Association for Computational Linguistics. 1, 2.2.1

Tal Schuster, Ori Ram, Regina Barzilay, and Amir Globerson. 2019. Cross-lingual alignment of contextual word embeddings, with applications to zero-shot dependency parsing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1599–1613, Minneapolis, Minnesota. Association for Computational Linguistics. 2.2.1

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. 2020. Green ai. *Communications of the ACM*, 63(12):54–63. 2.2

Henry Scudder. 1965. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3):363–371. 1.1, 2.2.1, 4.1

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics. 2.2.1

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics. 3.4, 4.1, 4.5.1

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016c. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany.

Association for Computational Linguistics. 2.2.1

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016d. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics. 4.3.1

Claude Elwood Shannon. 1948. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423. 3.1

Jude W Shavlik. 1994. Combining symbolic and neural learning. *Machine Learning*, 14(3):321–331. 9

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6830–6841. 3.1, 3.2, 3.5.3, 3.1, 3.2, 3.5.4

Josef Sivic and Andrew Zisserman. 2003. Video google: A text retrieval approach to object matching in videos. In *Proceedings of ICCV*. 8.2

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics. 6.4.1

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. MASS: masked sequence to sequence pre-training for language generation. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5926–5936. PMLR. 2.1.2, 4.5, 4.5.3

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015a. A neural network approach to context-sensitive generation of conversational responses. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 196–205, Denver, Colorado. Association for Computational Linguistics. 1

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015b. A neural network approach to context-

sensitive generation of conversational responses. In *Proceedings of NAACL*. 7.1

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics. 2.3

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. 2020. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 9229–9248. PMLR. 5.4.2

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR. 9

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*, pages 194–197. ISCA. 7.1, 8.1, 8.2

Yi Tay, Vinh Q Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, et al. 2022. Transformer memory as a differentiable search index. *arXiv preprint arXiv:2202.06991*. 2.2.3, 9

Neil C Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F Manso. 2020. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*. 9

Nenad Tomasev, Milos Radovanovic, Dunja Mladenic, and Mirjana Ivanovic. 2014. The role of hubness in clustering high-dimensional data. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):739–751. 8.4.2

Chau Tran, Yuqing Tang, Xian Li, and Jiatao Gu. 2020. Cross-lingual retrieval for iterative self-supervised training. *Advances in Neural Information Processing Systems*, 33:2207–2219. 2.2.1

Yi-Lin Tuan, Yun-Nung Chen, and Hung-yi Lee. 2019. DyKgChat: Benchmarking dialogue generation grounding on dynamic knowledge graphs. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1855–1865, Hong Kong, China. Association for Computational Linguistics. 2.2.3

Nicola Ueffing. 2006. Using monolingual source-language data to improve MT performance. In *Proceedings of the Third International Workshop on Spoken Language Translation: Papers*, Kyoto, Japan. 4.1

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008. 1, 1.1, 2.1.2, 3.2, 4.3.1, 4.5.2, 5.3.3, 6.2.1, 7.1, 8.1, 8.2, 8.3.1

Pat Verga, Haitian Sun, Livio Baldini Soares, and William Cohen. 2021. Adaptable and interpretable neural MemoryOver symbolic knowledge. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3678–3691, Online. Association for Computational Linguistics. 2.2.3

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics. 2.2.2

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3261–3275. 5.4.1, 6.4.2

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. 6.4.1

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno A. Olshausen, and Trevor Darrell. 2021. Tent: Fully test-time adaptation by entropy minimization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. 5.4.2

Jian Wang, Junhao Liu, Wei Bi, Xiaojiang Liu, Kejing He, Ruifeng Xu, and Min Yang. 2020a. Improving knowledge-aware dialogue generation via knowledge base question answering. In

Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 9169–9176. 2.2.3

Shuohang Wang, Yichong Xu, Yuwei Fang, Yang Liu, Siqi Sun, Ruochen Xu, Chenguang Zhu, and Michael Zeng. 2022. Training data is more valuable than you think: A simple and effective method by retrieving from training data. *arXiv preprint arXiv:2203.08773*. 2.2.3

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020b. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*. 2.2

Xinyi Wang, Hieu Pham, Zihang Dai, and Graham Neubig. 2018. SwitchOut: an efficient data augmentation algorithm for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 856–861, Brussels, Belgium. Association for Computational Linguistics. 2.2.1

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*. 2.2.1, 5.1, 5.2, 5.5

Jason Wei and Kai Zou. 2019. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388, Hong Kong, China. Association for Computational Linguistics. 2.2.1

Jason Weston, Emily Dinan, and Alexander Miller. 2018. Retrieve and refine: Improved sequence generation models for dialogue. In *Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd International Workshop on Search-Oriented Conversational AI*. 7.1

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. 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 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics. 6.4.1

R Williams. 1987. A class of gradient-estimation algorithms for reinforcement learning in neural networks. In *Proceedings of the International Conference on Neural Networks*, pages II–601. 3.3.2, 3.5.6

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256. 7.3.2

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics. 2.1.1

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771. 3

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics. 6.4.1

Shijie Wu and Mark Dredze. 2019. Beto, bentz, becas: The surprising cross-lingual effectiveness of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 833–844, Hong Kong, China. Association for Computational Linguistics. 2.2.1

Sixing Wu, Ying Li, Dawei Zhang, Yang Zhou, and Zhonghai Wu. 2020. Diverse and informative dialogue generation with context-specific commonsense knowledge awareness. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5811–5820, Online. Association for Computational Linguistics. 2.2.3

Yu Wu, Furu Wei, Shaohan Huang, Yunli Wang, Zhoujun Li, and Ming Zhou. 2019. Response generation by context-aware prototype editing. In *Proceedings of AAAI*. 7.1

Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. Memorizing transformers. In *International Conference on Learning Representations*. 2.2.3

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. 2019. Unsupervised data augmentation. *ArXiv preprint*, abs/1904.12848. 4.6

Qizhe Xie, Zihang Dai, Eduard H. Hovy, Thang Luong, and Quoc Le. 2020a. Unsupervised data augmentation for consistency training. In *Advances in Neural Information Processing*

Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. 2.2.1, 5.1, 5.1, 5.3.2, 5.3.2

Qizhe Xie, Minh-Thang Luong, Eduard H. Hovy, and Quoc V. Le. 2020b. Self-training with noisy student improves imagenet classification. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10684–10695. IEEE. 5.3.2, 5.4.2

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020a. BERT-of-theseus: Compressing BERT by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869, Online. Association for Computational Linguistics. 2.2.2

Jiacheng Xu and Greg Durrett. 2018. Spherical latent spaces for stable variational autoencoders. In *Proceedings of EMNLP*. 7.3.1, 7.3.2, .1.2, .2

Peng Xu, Mostofa Patwary, Mohammad Shoeybi, Raul Puri, Pascale Fung, Anima Anandkumar, and Bryan Catanzaro. 2020b. MEGATRON-CNTRL: Controllable story generation with external knowledge using large-scale language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2831–2845, Online. Association for Computational Linguistics. 2.2.3

Wei Xu, Alan Ritter, Bill Dolan, Ralph Grishman, and Colin Cherry. 2012. Paraphrasing for style. In *Proceedings of COLING 2012*, pages 2899–2914, Mumbai, India. The COLING 2012 Organizing Committee. 3.1, 3.2, 3.5.3

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics. 2.1.2, 2.2.1

Yiben Yang, Chaitanya Malaviya, Jared Fernandez, Swabha Swayamdipta, Ronan Le Bras, Ji-Ping Wang, Chandra Bhagavatula, Yejin Choi, and Doug Downey. 2020. Generative data augmentation for commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1008–1025, Online. Association for Computational Linguistics. 2.2.1

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Informa-*

tion Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 5754–5764. 1, 2.1.2, 7.1, 8.1

Zichao Yang, Zhiting Hu, Chris Dyer, Eric P. Xing, and Taylor Berg-Kirkpatrick. 2018. Unsupervised text style transfer using language models as discriminators. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7298–7309. 3.1, 3.4, 3.5.3, 3.1, 3.2, 3.5.3, 3.5.4

Xingcheng Yao, Yanan Zheng, Xiaocong Yang, and Zhilin Yang. 2021. Nlp from scratch without large-scale pretraining: A simple and efficient framework. *arXiv preprint arXiv:2111.04130*. 2.2

David Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, Massachusetts, USA. Association for Computational Linguistics. 4.1, 4.2

Yelp. 2017. Yelp dataset challenge, round 8. 7.1, 7.4.1

Pengcheng Yin, Graham Neubig, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. 2019. Learning to represent edits. In *Proceedings of ICLR*. 7.3.2

Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. 2018. StructVAE: Tree-structured latent variable models for semi-supervised semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 754–765, Melbourne, Australia. Association for Computational Linguistics. 2.2.1, 3.1, 2, 3.5.6, 4.1

Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. 2021. Adaptive semiparametric language models. *Trans. Assoc. Comput. Linguistics*, 9:362–373. 8.4.1

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31. 2.1.2

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics. 5.4.1

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond

empirical risk minimization. *arXiv preprint arXiv:1710.09412*. 2.2.1

Houyu Zhang, Zhenghao Liu, Chenyan Xiong, and Zhiyuan Liu. 2020a. Grounded conversation generation as guided traverses in commonsense knowledge graphs. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2031–2043, Online. Association for Computational Linguistics. 2.2.3

Jiajun Zhang and Chengqing Zong. 2016. Exploiting source-side monolingual data in neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545, Austin, Texas. Association for Computational Linguistics. 4.1

Yu Zhang, James Qin, Daniel S Park, Wei Han, Chung-Cheng Chiu, Ruoming Pang, Quoc V Le, and Yonghui Wu. 2020b. Pushing the limits of semi-supervised learning for automatic speech recognition. *ArXiv preprint*, abs/2010.10504. 5.4.2

Tiancheng Zhao, Ran Zhao, and Maxine Eskenazi. 2017. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 654–664, Vancouver, Canada. Association for Computational Linguistics. 3.1

Xueliang Zhao, Wei Wu, Can Xu, Chongyang Tao, Dongyan Zhao, and Rui Yan. 2020a. Knowledge-grounded dialogue generation with pre-trained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3377–3390, Online. Association for Computational Linguistics. 2.2.3

Yang Zhao, Lu Xiang, Junnan Zhu, Jiajun Zhang, Yu Zhou, and Chengqing Zong. 2020b. Knowledge graph enhanced neural machine translation via multi-task learning on sub-entity granularity. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4495–4505, Barcelona, Spain (Online). International Committee on Computational Linguistics. 2.2.3

Luowei Zhou, Chenliang Xu, and Jason J Corso. 2018. Towards automatic learning of procedures from web instructional videos. In *Thirty-Second AAAI Conference on Artificial Intelligence*. 2.1.1

Y. Zhou and S. Goldman. 2004. Democratic co-learning. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 594–602. 4.6

Zhi-Hua Zhou and Ming Li. 2005. Tri-training: exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541. 4.6

Xiaojin Zhu and Andrew B Goldberg. 2009. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130. 4.1, 4.2

Yaoming Zhu, Jiangtao Feng, Chengqi Zhao, Mingxuan Wang, and Lei Li. 2021. Serial or parallel? plug-able adapter for multilingual machine translation. *ArXiv preprint*, abs/2104.08154. 6.3.3