I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dan Jurafsky, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Percy Liang

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Christopher Manning

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.
Abstract

Neural language models (LMs) have become the workhorse of most natural language processing tasks and systems today. Yet, they are not perfect, and the two most important challenges in improving them further are (1) their lack of interpretability, and (2) their inability to generalize consistently, both in- and out-of-distribution. In this dissertation, I first describe my work on studying these LMs via black-box analysis, in order to understand how their predictions change in response to strategic changes in inputs. This makes model predictions more transparent by highlighting the features of the input that the model relies on. Then, I describe my work on Generalization through Memorization – exploiting the notion of similarity between examples by using data saved in an external memory and retrieving nearest neighbors from it. This approach improves existing LM and machine translation models in terms of both in- and out-of-domain generalization, without any added training costs. Beyond improving generalization, memorization also makes model predictions more interpretable.
Acknowledgments

I have had the wonderful honor of calling Stanford my home for the last six years. During this time, I have faced many ups and downs, which have left me humbled and with an immense sense of gratitude towards all the people who have helped and supported me through this journey. Foremost among them is my PhD advisor, Dan Jurafsky. He is a brilliant researcher and an amazing and kind human being, and I absolutely would not have been able to get this far without his guidance and support. I want to thank him for all that he has taught me about NLP, machine learning, becoming a better researcher and striving to be a better person. I also really want to thank him for helping me through some of my toughest failures, and for his unwavering belief in me, even when I had lost faith in myself. Dan’s brilliance, kindness and optimism have motivated and inspired me everyday since I’ve known him, and his values have helped to shape the person I have become. It has been my greatest honor and privilege to know Dan, and work with him, and here’s hoping for a lifetime of more fun conversations and collaborations!

I also want to thank Chris Manning and Percy Liang who, together with Dan, have been at the helm of the Stanford NLP Group since I started. I am immensely grateful to Chris for his support and kindness throughout this journey. His brilliance as well as his perceptiveness about how the field will grow and change have been deeply inspirational, and I have learned so much from our discussions. I am also immensely grateful to Percy, who has always encouraged me and taught me to ponder the bigger questions in the field. He is a remarkable researcher with a propensity for technical finesse unlike any other I have known, and he inspires me to be better, everyday. Dan, Chris and Percy have cultivated an amazing lab culture based on kindness and humility, and I am extremely grateful to them for allowing me to be a part of the Stanford NLP Group—I could not have asked for a better PhD family.

I would not have survived graduate school without the unyielding support of my friends from the lab. I want to thank Peng Qi and Yuhao Zhang for having my back all throughout. We were office mates for five years—proud residents of Gates 232—and have survived countless deadlines, rebuttals and existential crises together. Peng is an incredibly kind, patient and compassionate person, who was always very easy to talk to. Working with him was extremely fun and he taught me a great deal about the careful and rigorous framing and execution of ideas. Yuhao is extremely kind and his lightheartedness has brightened my toughest days. His comments on my drafts have made my papers better and his advice, especially this past year as I tried to graduate, has been invaluable. I am the last member of our office to graduate, and, even in these last
few weeks, Peng and Yuhao have encouraged and championed my efforts. I am extremely grateful to them for their unrelenting support. I also thank He He, Kevin Clark, Robin Jia, Arun Chaganty, Siva Reddy, Yulia Tsvetkov, Antoine Bosselut, Dallas Card, Peter Henderson and Kyle Mahowald, as well as Dan’s entire group and the broader Stanford NLP group, for their friendship, advice and encouragement, and for some extremely fun collaborations.

Apart from doing research at Stanford, I also spent two wonderful summers interning with Luke Zettlemoyer’s NLP group at Facebook. My discussions with Luke have always been extremely thought-provoking, and his optimism has always left me feeling excited and encouraged. Luke warmly welcomed me into his group and supported me far beyond the projects that we worked on together, and for that I am immensely grateful. I also want to thank Mike Lewis and Omer Levy for their mentorship, friendship and support. We collaborated during my internships, but I have turned to them for advice on numerous occasions and they have always helped me to find my way. Mike is extremely knowledgeable and working with him has been absolutely wonderful and fun; he has especially taught me to take a step back and enjoy the process. Brainstorming sessions with Omer are my favorite and I always walk away having learned something new about how to approach a problem. Both of them have supported my growth in far more ways than just discussing research, and for that I am extremely grateful. Opportunities to collaborate with members of the industry have been exceedingly helpful for me; I am grateful to Facebook AI and Google Brain for supporting my work through internships, and to Microsoft Research for supporting my work with a Dissertation Grant.

During my fourth year at Stanford, rather unexpectedly, I signed up for a course with Dan Ho at the Stanford Law School. Little did I know, that this would turn into an extremely fun and exciting collaboration. Dan’s endless supply of knowledge, which cuts across vastly different fields, will never cease to amaze and inspire me. I have learned so much from our discussions and want to sincerely thank Dan for allowing me to be a part of his wonderful team. I also thank Alex Yu for working on the PTO report with me and making it very fun.

Prior to coming to Stanford, I was a bright-eyed undergraduate student at the University of Illinois Urbana-Champaign. In my freshman year, Professor Jiawei Han took a chance on me by allowing me to join one of his projects, and encouraged me to pursue research throughout my time there. I owe Dr. Han, as well as my mentors Xiao Yu and Xiang Ren, a great debt of gratitude for fostering my growth as a researcher in those early years and setting me up to pursue my goals. I also thank Cinda Heeren and Chandra Chekuri for helping me fall in love with the field of computer science, and Julia Hockenmaier for first introducing me to NLP. I thank Sean Massung and Chase Geigle for their friendship, and for always including me in study groups and Scrubs-watching sessions. I thank Halie and Michael Robson-Rando and Eric and Ariel Mikida for always being there for me and showing me the lighter side of life. I thank Yonatan Bisk for all his sage advice through the years.

Mackenzie Leake is my closest friend, really she’s more like a sister, and she has been an incredible source of support for me throughout this journey. Mackenzie is kind, compassionate and remarkably creative, and her impeccable work ethic has always inspired me to be better. We started the program together and have
spent countless days since at the Palo Alto Creamery, wondering about that light at the end of the tunnel over warm slices of pie. She’s heard every one of my rants and celebrated all the milestones with me. I could not imagine getting through this program without her help and am very grateful for her constant support. I also want to thank my roommates, Amy Han, Victoria Cadiz and Alexandria Gilbert, who hugged me when I cried, tolerated me when I was frustrated and celebrated with me even when things were just slightly less bad. Their friendship and support has really meant a lot to me over the years.

Last and most importantly, I express my deepest gratitude to the three people in my life who have earned this PhD with me, and on some days, more than even I have. My older brother, Kartikay, has been looking out for me all my life. He has been my teacher, my therapist, my best friend, all wrapped in one, for almost three decades now. And I still cannot believe how lucky I have been to have him in my corner. He started working in NLP and machine learning long before me. I simply did what I knew best, I followed in his footsteps. And today, his help and support at every step is what got me through it. My brother’s patience and kindness have empowered me to shoot for the stars, and for this I will remain eternally grateful. In June of 2011, we attended Kartikay’s Stanford graduation. Now, ten years later, I have finally caught up to him. My parents, Jai and Nita, are brilliant and humble people, and they have waited for this day longer than even I have. Their unconditional love for me has been the most powerful driving force in my life. Even from 8,000 miles away, they have looked after me and supported me every single day. I love them so very much and could not have done any of it without them always by my side. The four of us have weathered this six-year storm together, and now, our PhD is complete.

Around the world, living through this pandemic has caused pain that is insurmountable. My heart breaks as I write these words, remembering my wonderful grandfather who always supported me, and cheerfully celebrated my defense mere weeks ago. We love you, and we will miss you...
To my parents, Jai and Nita, and to my brother, Kartikay.
# Contents

Abstract iv

Acknowledgments v

1 Introduction 1
   1.1 Key Goals for Neural Language Models ....................... 2
      1.1.1 Interpretability .................................... 2
      1.1.2 Generalization .................................... 4
   1.2 Thesis Outline ........................................ 6

2 An Overview of Language Models 9
   2.1 Task Definition and Evaluation .......................... 10
   2.2 n-gram Language Models ................................. 12
   2.3 Neural Language Models ................................ 13
      2.3.1 The Self-Supervised Learning Approach ............ 14
      2.3.2 Training Neural Language Models .................. 15
      2.3.3 Generation using Language Models .................. 17
   2.4 Model Architectures .................................... 17
      2.4.1 Long Short-Term Memory Networks .................. 17
      2.4.2 Transformer Networks ............................... 20
   2.5 The Impact of Scale .................................... 23

3 How Neural Language Models Use Context 24
   3.1 Approach .............................................. 25
   3.2 How much context is used? ................................ 27
      3.2.1 Do different types of words need different amounts of context? .......................... 28
   3.3 Nearby vs. long-range context ........................... 30
      3.3.1 Does word order matter? ................................ 30
      3.3.2 Types of words and the region of context .................. 32
3.4 To cache or not to cache? ........................................... 33
  3.4.1 Can LSTMs copy words without caches? ...................... 33
  3.4.2 How does the cache help? ..................................... 36
3.5 Discussion ................................................................. 37
3.6 Transformer Language Models ...................................... 38
3.7 Conclusion ................................................................. 40

4 Generalization through Memorization ....................................... 41
  4.1 Motivation .................................................................. 41
  4.2 Exemplar Models ......................................................... 43
  4.3 The Memorization Framework for Neural Models .......... 45

5 Nearest Neighbor Language Models ......................................... 48
  5.1 kNN-LM Model .......................................................... 49
  5.2 Experimental Setup ...................................................... 50
  5.3 Experiments ................................................................. 52
    5.3.1 Using the Training Data as the Datastore ............ 52
    5.3.2 More Data without Training ................................. 53
  5.4 Tuning Nearest Neighbor Search .................................... 54
  5.5 Analysis .................................................................... 55
  5.6 Related Work ............................................................. 57
  5.7 Conclusion ................................................................. 59

6 Nearest Neighbor Machine Translation ....................................... 60
  6.1 kNN-MT Model .......................................................... 61
  6.2 Experimental Setup ...................................................... 62
  6.3 Experiments ................................................................. 64
    6.3.1 Single Language-Pair Translation ....................... 64
    6.3.2 Task Specialization for Multilingual Machine Translation .... 64
  6.4 Tuning kNN-MT .......................................................... 66
  6.5 Hyperparameter Choices ................................................ 67
  6.6 Qualitative Analysis ...................................................... 67
  6.7 Related Work ............................................................. 68
  6.8 Conclusion ................................................................. 71

7 Domain Adaptation .............................................................. 72
  7.1 Task Definition .......................................................... 73
  7.2 Experiments ............................................................... 74
    7.2.1 Domain Adaptation for Language Modeling ........... 74
### List of Tables

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>PTB and Wiki dataset statistics and perplexity scores.</td>
</tr>
<tr>
<td>3.2</td>
<td>Hyperparameter settings for our LSTM LM.</td>
</tr>
<tr>
<td>5.1</td>
<td>Performance of $k$NN-LM on WIKITEXT-103.</td>
</tr>
<tr>
<td>5.2</td>
<td>Performance of $k$NN-LM on BOOKS.</td>
</tr>
<tr>
<td>5.3</td>
<td>Experimental results for using $k$NN-LM to larger datasets.</td>
</tr>
<tr>
<td>5.4</td>
<td>Using different model representations for retrieval.</td>
</tr>
<tr>
<td>6.1</td>
<td>Multilingual machine translation with $k$NN-MT.</td>
</tr>
<tr>
<td>6.2</td>
<td>Retrieving neighbors using English as the source language.</td>
</tr>
<tr>
<td>6.3</td>
<td>Multilingual machine translation with $k$NN-MT on the validation set.</td>
</tr>
<tr>
<td>7.1</td>
<td>Domain adaptation for language modeling with $k$NN-LM.</td>
</tr>
<tr>
<td>7.2</td>
<td>Domain adaptation for machine translation with $k$NN-MT.</td>
</tr>
<tr>
<td>7.3</td>
<td>Hyperparameters for domain adaptation with $k$NN-MT.</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The Transformer model architecture.</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>Effective context size of LSTM LMs.</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>Changing model hyperparameters does not change the trend for context use.</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Effect of local word order.</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>Effect of global word order.</td>
<td>31</td>
</tr>
<tr>
<td>3.5</td>
<td>Effect of dropping content and function words from the context.</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>Effects of perturbing the target word in the context.</td>
<td>34</td>
</tr>
<tr>
<td>3.7</td>
<td>Successes of neural caching model.</td>
<td>35</td>
</tr>
<tr>
<td>3.8</td>
<td>Failures of neural caching model.</td>
<td>36</td>
</tr>
<tr>
<td>3.9</td>
<td>Model performance relative to using the neural caching model.</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>An illustration of kNN-LM.</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>Varying the size of the kNN-LM datastore.</td>
<td>54</td>
</tr>
<tr>
<td>5.3</td>
<td>Transformer LM layer.</td>
<td>55</td>
</tr>
<tr>
<td>5.4</td>
<td>Effect of the number of nearest neighbors retrieved.</td>
<td>55</td>
</tr>
<tr>
<td>5.5</td>
<td>Interpolating the Transformer LM with n-gram LMs.</td>
<td>56</td>
</tr>
<tr>
<td>5.6</td>
<td>Memorizing by overfitting a Transformer LM.</td>
<td>56</td>
</tr>
<tr>
<td>5.7</td>
<td>First example of kNN-LM retrieved neighbors.</td>
<td>57</td>
</tr>
<tr>
<td>5.8</td>
<td>Second example of kNN-LM retrieved neighbors.</td>
<td>58</td>
</tr>
<tr>
<td>6.1</td>
<td>An illustration of kNN-MT.</td>
<td>61</td>
</tr>
<tr>
<td>6.2</td>
<td>Effect of number of neighbors and softamx temperature.</td>
<td>67</td>
</tr>
<tr>
<td>6.3</td>
<td>Effect of datastore size.</td>
<td>67</td>
</tr>
<tr>
<td>6.4</td>
<td>First example of kNN-MT retrieved neighbors.</td>
<td>69</td>
</tr>
<tr>
<td>6.5</td>
<td>Second example of kNN-MT retrieved neighbors.</td>
<td>70</td>
</tr>
<tr>
<td>7.1</td>
<td>Effect of interpolation parameter in-domain vs. out-of-domain.</td>
<td>75</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

A long-standing goal of artificial intelligence research is to build models that can understand and generate language in the same way as humans do. With decades of progress in the field of natural language processing (NLP), our day-to-day lives are now steadily becoming supplemented by smart language tools such as online translators like Google Translate, writing features like SmartCompose and Grammarly, virtual assistants like Alexa and Siri, and many more. And with the widespread adoption of deep learning over the last several years, the NLP models that power these systems have never been better. Moreover, in recent years neural language models have been at the heart of most state-of-the-art NLP models (Devlin et al., 2019; Radford et al., 2019; Conneau et al., 2020; Lepikhin et al., 2020; Brown et al., 2020, inter alia) and hence, are the primary objects of study in this dissertation.

Historically, language models (LM) defined a distribution over strings, assigning higher probabilities to fluent and grammatical sequences, and lower probabilities to nonsensical ones. Language modeling can also be interpreted as the task of predicting the next word in a sequence by conditioning on the leftward prior context. More recently, masked language modeling was proposed, which is the task of predicting the missing word in a sequence by conditioning on context on both the left and right sides of the missing word. Consequently, neural LMs are multi-layer neural networks trained on large amounts of unlabeled text using a self-supervised learning approach, that is, learning to predict masked words given the surrounding context words. For instance, given the sequence I’ll meet you at the ______, the model should assign higher probabilities to the more plausible next words like restaurant or airport, as compared to words like eating which are far less likely given the context. Such LMs have now become the foundation on which models that perform other tasks are built. At present, the most common broad class of approaches for using LMs include:

- **Representation learning**: LMs learn representations of text which have, in the past, been directly used for a broad range of tasks like detecting entailment or classifying sentiment (Bowman et al., 2015; Wang et al., 2018). However, in recent years they are most commonly used as part of the transfer learning paradigm because large LMs trained on billions of tokens learn very effective representations
which serve as an excellent starting point for training models that perform other tasks. In this transfer learning paradigm, the LM functions as the pre-trained model which is fine-tuned to learn the desired target task.

- **Generation**: LMs form the basis of text generation systems like machine translation models, chatbots and conversational agents, speech recognition systems etc., where they are used to generate sequences of text word by word, in an autoregressive fashion (Sutskever et al., 2014; Cho et al., 2014).

- **Few-shot learning**: More recently, with the release of GPT-3 (Brown et al., 2020), it has been shown that large language models can non-trivially learn to perform new tasks by simply processing examples as leftward prior context, without any additional training. This approach of providing examples as demonstrations, called in-context learning, has also sparked a lot of interest in prompt-based fine-tuning and adaptation of LMs using a very small number of examples and parameter updates to learn a range of downstream tasks (Schick and Schütze, 2020; Gao et al., 2020; Li and Liang, 2021).

A key ingredient that is contributing to the recent success of neural LMs is scale. Given the self-supervised learning approach, these models can be trained on very large amounts of data and tend to contain on the order of hundreds of millions or even billions of parameters. While this approach incurs tremendous amounts of computational costs during training and while we have a relatively poor understanding of the behavior of these massive neural LMs, they have exhibited unprecedented capabilities through a range of tasks including question answering, machine translation, dialogue and many more.

And yet, in spite of becoming the cornerstone of most modern NLP systems, neural LMs are far from perfect. These models can sometimes exhibit unpredictable and inconsistent behavior when given inputs that they have not seen before, and this inability to generalize can even cause harm when the predictions are biased. Moreover, since we have a very poor understanding of how these models make predictions, we cannot anticipate failures or explain what is causing them. Hence, the lack of interpretability and the inability to generalize consistently across a range of settings are the two most important challenges in improving neural LMs today. In the rest of this chapter, we first describe why interpretability and generalization of neural LMs are important goals for the NLP community to pursue. And then, we provide an outline for the work presented in this dissertation in making progress towards these goals.

### 1.1 Key Goals for Neural Language Models

#### 1.1.1 Interpretability

As machine learning-based technologies are being developed and deployed for tasks in fields ranging from medicine and security to finance and the law, it has become absolutely imperative that humans gain more insight into the models’ decision-making processes and predictions. This insight is important because when any aspects of the decision-making process are automated, understanding the attributes that the model relies
on for making predictions and understanding how confident or how error-prone the model predictions are can be useful in assessing the reliability of these models. One example of this is the fact that many of our existing models tend to exhibit biases of various forms, like gender or racial bias (Buolamwini and Gebru, 2018; Nadeem et al., 2020), even though they haven’t been explicitly trained on those attributes, making them highly unreliable. For instance, in 2015 it was shown that an automated résumé filtering system used by some technology companies appeared to be biased against women for certain technical roles, even though it was never explicitly trained to use gender as a feature for making those classification decisions (Dastin, 2018). This system had demonstrated biased and unfair behavior towards a sub-population within the data, making its predictions unreliable and untrustworthy. For this reason, as the quality of our models improves, developing state-of-the-art systems that are also interpretable has become vital.

The lack of model interpretability is a particularly important challenge facing modern systems across many sub-fields of artificial intelligence because deep learning models, which power most of these systems, are black-box in nature and thus are inherently not interpretable. Neural networks map inputs to high-dimensional numerical vectors and it is these opaque representations that are manipulated through several layers of mathematical operations, before the final prediction is made. Consequently, neural LMs are also black-box in nature and given how ubiquitous they have become in the field of NLP, understanding and interpreting their predictions is an important open problem that has drawn a lot of interest in recent years.

As a consequence of this growing interest, varying motivations have caused interpretability research to vacillate between many different goals and approaches, without much consensus on how we, as a community, rigorously define model interpretability and the right ways to evaluate the faithfulness of interpretations. Discussions on these questions are vital, and on-going (Doshi-Velez and Kim, 2017; Lipton, 2018; Jacovi and Goldberg, 2020), as work in this area proliferates. In recent years, an incredibly broad range of approaches have been investigated for understanding and interpreting the predictions of black-box neural network models. A small subset of these approaches includes: post-hoc black-box analyses which investigate how model predictions change in response to changes in inputs (eg: Linzen et al., 2016; Khandelwal et al., 2018; Adler et al., 2018), probing specific components of the model to study them in isolation (eg: Adi et al., 2017; Hewitt and Manning, 2019; Clark et al., 2019), influence functions to identify which training data points are responsible for the given predictions (eg: Koh and Liang, 2017), saliency maps and integrated gradients which are feature attribution methods (eg: Simonyan et al., 2013; Sundararajan et al., 2017) and neural pathways which do this for co-occurring sets of features (Fiacco et al., 2019), visualization methods for neural network features (eg: Strobelt et al., 2017; Olah et al., 2018; Vig, 2019), and generating post-hoc explanations (eg: Ribeiro et al., 2016). However, in light of such broad explorations, a number of follow-up discussions have emerged which highlight technical shortcomings in some of the aforementioned approaches such as over-reliance on unverified intuitions, drawing conclusions without quantifying across data samples, hyperparameter settings or random seeds, not considering the falsifiability of the hypotheses being verified, drawing spurious conclusions due to unverified assumptions to name a few (eg: Jain and Wallace, 2020; Pruthi et al., 2020; Leavitt and Morcos, 2020). This on-going scientific discourse highlights the new and active nature of interpretability
research which is key to making progress towards the goal of building state-of-the-art interpretable systems.

In this dissertation, we employ black-box model analysis to investigate neural LMs. Post-hoc model analysis serves two key purposes in relation to interpretability. First, it adds transparency to model decisions, which is an important step in making these systems reliable and trustworthy. Second, it helps to understand model behavior in a way that can highlight shortcomings and suggest ways to improve performance. With these goals in mind, we analyze neural LMs via black-box analysis to investigate a collection of hypotheses related to how these models use the provided prior context and which contextual features contribute to model predictions (Khandelwal et al., 2018). More specifically, we quantify how model predictions change in response to changes in inputs by applying strategic perturbations like shuffling the words in the prior context. This study aids us in building a better understanding of which aspects of the inputs these models rely on when making predictions, for instance the fact that they use limited prior context and ignore words that occur beyond a certain window. It also highlights limitations in the models which can be addressed to improve performance, such as the need of hierarchical contexts for long-form generation.

1.1.2 Generalization

One of the most basic tenets of machine learning is that a model trained on some given data should be able to generalize to examples not seen during training. These examples can either belong to the same distribution as the training data, which is the most commonly explored setting to date, or the examples could belong to a different distribution which is less common throughout existing benchmarks. Regardless, even our state-of-the-art models exhibit inconsistent generalization capabilities across a variety of settings. We discuss the problem of inconsistent generalization, for both in- and out-of-distribution settings, below.

**In-distribution** The dominant paradigm for building NLP models is to train them to perform a single task, learned from a fixed and often single-domain dataset. For instance, training a machine translation model that only translates news articles from German into English. Then, when we want to evaluate the model, we measure its performance on test data that comes from the same or a very similar data distribution, and this provides a measure of model generalization. In recent years, we have found that our models generalize well and that in-distribution performance improvements on traditional benchmarks have been unprecedented.

However, in spite of the rapid progress, in some cases even when the train and test data are drawn from similar distributions, model performance can be inconsistent. For instance, while we can seemingly learn strong translation models for the German to English language pair, performance of models trained on other language pairs, like translating from German to Japanese, remains low (Schwenk et al., 2019). This shows that we still stand to improve model generalization. As discussed earlier, while scaling the model size and training on additional data has proved to be effective in improving in-distribution performance to a certain extent, it tends to be a very costly approach.

In this dissertation, we propose to improve model generalization through memorization. More specifically, we propose the explicit memorization of data samples in an external memory called the datastore,
and show large improvements in language modeling and machine translation performance by querying this memory at test time (Khandelwal et al., 2020, 2021). Memorization improves model performance without incurring any additional training costs; it is not prohibitive, since it does not require tremendous amounts of compute, and can be used with any existing model. At a high level, the idea of memorization relies on the intuition that neural models are good at identifying whether a test input is similar to an example that was seen during training, and perhaps the prediction for the test sample is the same as for the similar previously seen examples. Memorization also makes model predictions more interpretable because when solely retrieving example from the datastore, each prediction can be traced back to the specific examples that were responsible for the final decision. Our work on memorization draws inspiration from the widely studied exemplar models, proposed in the fields of Cognitive Science and Psychology (Medin and Schaffer, 1978; Nosofsky, 1987), and more specifically, the related non-neural memory-based models that were explored in NLP (Daelemans and van den Bosch, 2005). See Section 4.2 for a discussion on exemplar models.

Out-of-distribution The world around us is constantly changing, and as humans, we are very skilled at adapting to those changes. However, our language tools—virtual assistants like Alexa and Siri, or autocompletion tools like SmartCompose—are unable to adapt to new and changing environments. For instance, in November 2020 after we had been living in lockdown due to the coronavirus pandemic for almost ten months, when asked what do people do for fun these days, virtual assistants responded with answers from the web like travel and spend time with others. This is extremely misleading because these systems completely disregarded the pandemic and its crippling effect on the world. Thus, it is extremely important for NLP systems to be able to adapt.

Most existing NLP models are not designed to adapt to new environments and changing data distributions. One of the most extensively studied model adaptation settings is that of domain adaptation, where a model is evaluated on target-domain data from a different but related distribution to the source-domain training data. Studies on domain adaptation have included the supervised setting where labeled data in the target domain is available, as well as the semi-supervised case where we only have access to unlabeled data in the target domain (Roark and Bacchiani, 2003; Blitzer et al., 2006; Daumé III, 2009). In recent years since the adoption of neural networks for NLP, another widely explored setting for changing data distributions is the study of model robustness to adversarially constructed examples. Numerous studies have highlighted a lack of robustness in existing state-of-the-art models in the context of adversarial examples where strategic alterations to inputs cause models to fail catastrophically without affecting human performance on those benchmarks (Goodfellow et al., 2014). Another example of adaptation is continual learning for building systems that continuously adapt to new tasks and new inputs without forgetting previously acquired skills and knowledge (Hadsell et al., 2020).

In this work, we study the aforementioned domain adaptation setting where the underlying task remains the same but the train and test samples come from different domains, which is why domain adaptation measures a model’s out-of-distribution generalization. An example of this is using a translation model that translates German news articles into English to now translate German medical texts, which are different from
news in terms of both content and structure. Since they belong to different domains, samples for medical data versus news are drawn from different underlying distributions. In this case, we have shown that the machine translation model trained on news shows poor and unreliable performance on the medical texts (Khandelwal et al., 2021). It is not able to adapt to different domains and suffers from poor out-of-distribution generalization.

To address this lack of model adaptation, we apply our proposed approach of memorization to the domain adaptation setting (Khandelwal et al., 2020, 2021). More specifically, given a model trained on the source domain (news in our example) and given a dataset of samples from the target domain (medical in our example), we create a datastore containing the examples from the target domain and query it when making predictions for test samples from this domain. This allows a single model to adapt to many different domains by simply building and querying a domain-specific datastore, without incurring any additional training costs. This is a valuable approach because it is unclear to what extent fine-tuning a model on a new domain would preserve performance on previous domains, and whether collective training on all domains is robust when met with new and previously unanticipated domains. Memorization allows an existing model to be effective in new domains without training on any in-domain data.

1.2 Thesis Outline

Based on the goals described in Section 1.1, the work presented in this thesis discusses approaches for making progress towards improved interpretability and generalization of neural LMs.

After providing an overview of language models in Chapter 2, we first show, via black-box analysis, how neural LMs use the given leftward prior context by perturbing it in strategic ways, like shuffling the words, and quantifying changes in model performance. Then, we describe the proposed generalization through memorization framework and show that memorizing prior context improves language model and machine translation generalization, both in- and out-of-distribution.

**How neural LMs use context.** In Chapter 3, we study the role of context in LSTM-based LMs. The goal of this study is to complement the prior work that studied LSTMs at the sentence level by providing a richer understanding of the role of context, and more specifically, the long-range context beyond a sentence. We aim to answer the following questions: (i) How much context is used by NLMs, in terms of the number of tokens? (ii) Within this range, are nearby and long-range contexts represented differently? (iii) How do copy mechanisms help the model use different regions of context?

We investigate these questions via black-box analysis of a standard LSTM LM (Merity et al., 2018) on two benchmark language modeling datasets: Penn Treebank and WikiText-2. Given a pretrained LM, we perturb the prior context in various ways at test time, to study how much the perturbed information affects model performance. Specifically, we alter the context length to study how many tokens are used, permute tokens to see if LSTMs care about word order in both local and global contexts, and drop and replace target words to
test the copying abilities of LSTMs with and without an external copy mechanism, such as the neural cache (Grave et al., 2017c). The cache operates by first recording target words and their context representations seen in the history, and then encouraging the model to copy a word from the past when the current context representation matches that word’s recorded context vector.

We find that the LSTM is capable of using about 200 tokens of context on average, with no observable differences from changing the hyperparameter settings. Within this context range, word order is only relevant within the 20 most recent tokens or about a sentence. In the long-range context, order has almost no effect on performance, suggesting that the model maintains a high-level, rough semantic representation of faraway words. Finally, we find that LSTMs can regenerate some words seen in the nearby context, but heavily rely on the cache to help them copy words from the long-range context.

**Generalization through Memorization** In Chapter 4, we provide an overview of the generalization through memorization framework. This includes motivating the approach by highlighting limitations of existing large-scale LMs. Then, we draw connections with the widely studied exemplar models that were first proposed in the field of cognitive science and psychology, and have been extensively explored in the field of NLP as well. And finally, we provide a high-level description of the proposed neural network-based memorization framework in a task-agnostic fashion.

**Nearest neighbor language models.** In Chapter 5, we introduce $k$NN-LM, an approach that extends a pre-trained LM by linearly interpolating its next word distribution with a $k$-nearest neighbors ($k$NN) model. The nearest neighbors are computed according to distance in the pre-trained embedding space and can be drawn from any text collection, including the original LM training data. This approach allows rare patterns to be memorized explicitly, rather than implicitly in model parameters. It also improves performance when the same training data is used for learning the prefix representations and the $k$NN model, strongly suggesting that the prediction problem is more challenging than previously appreciated.

To better measure these effects, we conduct an extensive empirical evaluation. Applying our $k$NN augmentation to a strong WIKITEXT-103 LM using only the original dataset achieves a new state-of-the-art perplexity of 15.79 – a 2.86 point improvement over the base model (Baevski and Auli, 2019) – with no additional training. We also show that the approach has implications for efficiently scaling up to larger training sets and allows for effective domain adaptation, by simply varying the nearest neighbor datastore. Training a model on 100-million tokens and using $k$NN search over a 3-billion token dataset can outperform training the same model on all 3-billion tokens, opening a new path for efficiently using large datasets in language models. Qualitatively, we find the model is particularly helpful for long-tail patterns, such as factual knowledge, which might be easier to access via explicit memory.
Nearest neighbor machine translation. In Chapter 6, we introduce $k$NN-MT, a simple non-parametric method for machine translation (MT) using nearest neighbor retrieval. $k$NN-MT can be added to any pre-trained neural translation model without further training, and significantly improves performance for in-domain, out-of-domain, and multi-lingual evaluations.

More specifically, $k$NN-MT interpolates the target-token softmax distribution from a neural MT model with a multinomial generated using nearest neighbor search over examples cached in a data store. The cache is over translation contexts (i.e. the complete source and prefix of the target), and is indexed by hidden states computed from the base MT model. We hypothesize that contexts which are close in representation space are more likely to be followed by the same target word. We show this is not only true for the original training data, thereby improving base model performance, but across a range of different bi-text corpora, allowing for simple and effective model adaptation.

Our work builds upon results showing the effectiveness of nearest neighbor methods in unconditional language models (Khandelwal et al., 2020). We generalize to conditional language models, by using both source and target context, and show nearest neighbour models can be effective for generation in addition to density estimation. Compared to prior work on non-parametric methods for MT, our approach is arguably simpler (in that it requires no training, as compared to Gu et al. (2018)) and more expressive (in that it provides access to billions of key-value pairs during inference, as compared to Zhang et al. (2018); Gu et al. (2018)).

Extensive experiments show that $k$NN-MT scales to datastores containing billions of tokens, improving results across a range of settings. For example, it improves a state-of-the-art German-English translation model by 1.5 BLEU. In addition, language-pair-specific datastores are used to adapt a multilingual model to particular language pairs, with improvements of 3 BLEU for translating English into German and Chinese. We find that retrievals from $k$NN-MT are typically highly contextually relevant.

Domain Adaptation. In Chapter 7, we motivate and describe the problem of domain adaptation. Then, we demonstrate that beyond improving in-distribution performance, memorization is also extremely effective in improving out-of-distribution generalization via experimental results for domain adaptation. For language modeling, we show that adding out-of-domain data to the $k$NN-LM datastore makes a single LM useful across multiple domains, without additional training of the model’s parameters. Specifically, an LM trained on Wikipedia and evaluated on a test set from the Books domain is improved by over 14 points of perplexity when using a domain-specific datastore.

Similarly, $k$NN-MT can also be used to adapt a single machine translation model to diverse domains by simply adding a domain-specific datastore—improving results by an average of 9.2 BLEU over the base model out-of-domain, and even outperforming existing models that train on these domains.
Chapter 2

An Overview of Language Models

The task of language modeling involves assigning probabilities to sequences of text, so as to be able to distinguish likely sequences from unlikely ones. More commonly, language modeling is framed as the task of predicting missing words given the surrounding context words, and in recent years language models have tended to be classified into one of two categories. The more traditional autoregressive language model (LM) defines a probability distribution over next words, conditioned on the leftward context up to that point. For instance, *I enjoy reading _____* is an example of the autoregressive language modeling task. On the other hand, the recently proposed masked language model (MLM) defines a probability distribution over the missing words conditioned on both the leftward prior context as well as rightward future context (Devlin et al., 2019), similar to the cloze test.¹ An example of the masked language modeling task is: *I will _____ you at the airport.* Given input sequences, LMs compute probability distributions over a vocabulary of words, and this distribution reflects the likelihood of each word completing that provided sequence. For instance, given the context *I like reading*, is the word *books* more likely to be generated next, or the word *bread*? This dictates the probability mass assigned to each word of the vocabulary.

Language modeling is one of the most fundamental tasks in the field of NLP, and recently it has become the basis of most models that perform other tasks. Long before deep learning was widely adopted in the NLP community, LMs had become a core part of the statistical modeling approaches for a range of tasks including automatic speech recognition, machine translation and handwriting recognition (Brown et al., 1990; Kernighan et al., 1990; Zue et al., 1991; Srihari and Baltus, 1992; Chen and Goodman, 1999), used mainly for scoring the predicted sequences or hypotheses. Today, most modern text generation systems include a neural LM as the key component that is used to generate sequences of text word by word (see Section 2.3.2 for a brief discussion on text generation models). Apart from being used to generate sequences, neural LMs have also been the key to recent success of the transfer learning paradigm in NLP. Transfer learning involves

---

¹Note that when the rightward context is empty, the masked language modeling task becomes an autoregressive one. When we discuss autoregressive language modeling, we specifically consider models that have been trained to predict next word distributions conditioned on the prior context alone.
using a model that has been trained on some task and adapting it for a different task and/or dataset, where
the hope is that pre-training allows the model to learn knowledge that can be transferred to the target task.
Language modeling is particularly well-suited for the transfer learning setup for two main reasons. First,
novel LMs learn very general context representations that are useful for a broad range of downstream tasks.
And second, LMs are trained using the self-supervised learning approach, so training datasets contain large
amounts of unlabeled text which is far more easily available compared to labeled examples typically required
for most other tasks. Large pre-trained LMs have brought about unprecedented improvements across a very
broad range of downstream tasks and thus, have become a core component of almost every existing NLP
system. This demonstrates the broad scope of impact of the task of language modeling.

In this dissertation, we primarily study autoregressive language modeling as an intrinsic task rather than
through the lens of transfer learning, as well as one application for text generation, namely machine trans-
lation. This chapter will provide the background material that will be relevant for the ideas discussed in the
following chapters. We start by formalizing the task definition and establishing evaluation metrics. This is
followed by a brief discussion on the dominant non-neural language modeling approach, \( n \)-gram LMs as well
as some of their limitations. Then we discuss neural LMs, the value of context-based representations and how
they are learned, as well as the specific neural architectures that have been most commonly used. Finally, we
end with a brief discussion on the impact of scale on progress in language modeling research.

2.1 Task Definition and Evaluation

In this section, we define autoregressive language modeling which is the primary task investigated in this
thesis. Henceforth, all references to language modeling refer to the autoregressive version.

A language model defines a probability distribution over sequences of tokens (Shannon, 1951). In prac-
tice, the probability for a given sequence \( P(w_1, \ldots, w_T) \) is factorized using the chain rule, such that

\[
P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t|w_{t-1}, \ldots, w_1),
\]

and so the joint probability is computed as a product of separate conditional probability computations for
each token in the sequence. This allows us to frame the task as a next word prediction task, that is, given
as input some prior context tokens \( c_t = (w_1, \ldots, w_{t-1}) \), LMs estimate a conditional probability distribution
\( p(w_t|c_t) \) for the target word \( w_t \), where the distribution is defined over items in the vocabulary \( V \). Framing
the task in this way also allows us to see that LMs can be used to generate sequences by drawing samples
from the distribution at the given time step \( t \), before moving on to estimating a distribution for the next time
step \( t + 1 \).

Language modeling is a density estimation task learned from events, context-target pairs, observed in the
training corpus \( \mathcal{D} = \{ (c_j, w_j) \} \) for \( j = 1, \ldots, m \), such that \( m \) denotes the total number of context-target
pair examples in the dataset, which is generally also the length of the corpus in terms of the number of
tokens. LMs are trained via maximum likelihood estimation where the training objective used to estimate
the parameters $\theta$ is minimizing the negative log-likelihood of the training set, where scaling by $\frac{1}{m}$ does not
change the solution:

$$\theta_{\text{ML}} = \arg \min_{\theta} -\frac{1}{m} \sum_{j=1}^{m} \ln p_{\text{LM}}(w_j|c_j; \theta)$$  \hspace{1cm} (2.2)

Then, we evaluate LM performance using the information theoretic metric called *perplexity*, which measures
how well a model predicts test data $D_{\text{test}} = \{(x_i, y_i)\}$ for $i = 1, \ldots, n$, where $(x_i, y_i)$ are the context-target
pairs, and the length of the test corpus is $n$. Perplexity (PP) is simply the exponentiated average negative
log-likelihood of this held-out test data, computed as follows:

$$\text{PP}(D_{\text{test}}) = \exp \left(-\frac{1}{n} \sum_{i=1}^{n} \ln p_{\text{LM}}(y_i|x_i; \theta) \right)$$  \hspace{1cm} (2.3)

where lower values of perplexity indicate a better model.

Another perhaps more intuitive way to think about perplexity is as the weighted average branching factor
of a language (Jurafsky and Martin, 2000), or the number of words the model is confused between when
predicting the next word. This makes it easier to see that the perplexity for a corpus when using an LM that
simply outputs a uniform distribution over items in the vocabulary is $|V|$, the size of the vocabulary (assuming
no out-of-vocabulary tokens).

Since perplexity is measured at the corpus level, two LMs are only comparable when the underlying test
corpus and the vocabulary used are identical. Beyond intrinsic evaluations, LMs can also be compared via
extrinsic application-specific evaluations where performance is compared on a downstream task like translation.

Before concluding this section and moving on to specific approaches to the language modeling task, we
briefly discuss how the corpora and vocabularies are set up. For word-level language modeling, the vocab-
ulary consists of words that occur in the training corpus with some minimum frequency. Historically,
most LMs have been word-level and most language modeling benchmarks provided tokenized datasets and
pre-defined vocabularies. However, these word-level models are unable to predict unseen or rare words that
are not present in the vocabulary and typically require smoothing mechanisms to avoid infinite perplexities
(see Jurafsky and Martin (2000) for a detailed discussion on smoothing). Rather than incorporating complex
smoothing strategies or growing the size of the vocabularies to improve coverage, which can be computa-
tionally expensive, character-level models have been explored and, more recently, subword-level models
(Sennrich et al., 2016) have become the norm.
2.2 \( n \)-gram Language Models

Before deep learning was adopted for NLP research, the \( n \)-gram LM was the dominant approach for language modeling. An \( n \)-gram is a sequence of \( n \) tokens. For instance, \emph{I} is a unigram or 1-gram, \emph{I like} is a bigram or 2-gram, \emph{I like reading} is a trigram or 3-gram, and so on. \( n \)-gram LMs are non-parametric models that condition on truncated contexts of length \( n - 1 \) when estimating next word distributions. Their non-parametric nature means these models make few assumptions about the data distribution and model complexity grows with the size of the data. These models rely on the \( k \)-th order Markov assumption which assumes that the next word in a sequence only depends on the previous \( k \) words. And so, using maximum likelihood estimation, the next word probability conditioned on the previous \( n - 1 \) tokens of context \( P(w_t|c_t=(w_{t-n}, \ldots, w_{t-1})) \) is the fraction of occurrences of the input context \( c_t=(w_{t-n}, \ldots, w_{t-1}) \) in the training corpus that are followed by the target word \( w_t \). This is known as relative frequency estimation. The frequency of contexts and context-target pairs in the training set are recorded offline and coverage of \( n \)-grams depends on the size and diversity of the training corpus. Then, given a test example \((x_i, y_i)\), estimating a conditional probability distribution \( p(y_i|x_i) \) involves a series of look-up operations combined with some post-processing steps such as smoothing.

Given the Markov assumption, the size of contexts being modeled is a key hyperparameter for \( n \)-gram LMs and, in fact, most popular toolkits settled on using \( n = 5 \) (Brants et al., 2007; Pauls and Klein, 2011; Heafield, 2011). In spite of modeling such short contexts, these models proved to be quite effective for language modeling as well as downstream tasks. In addition, the shorter sequences made it easier to know how much context was used and the precise tokens that impacted predictions. However, intuitively the Markov assumption for language, with such small spans of dependence, seems extreme. Often, far longer contexts are required to model the long-range dependencies that are inherent in language. For instance, given the 4-gram context \emph{is a popular romantic _____}, a 5-gram LM might assign a high probability mass to the next word \emph{gesture}, based on corpus statistics. However, the model needs to be able to incorporate long-range dependencies in order to recognize that the context \emph{In the 1800s, novels from many genres were . . . . . . . Pride and Prejudice which is a popular romantic _____} should be followed by the word \emph{novel}, rather than words like \emph{gesture} or \emph{film}.

Why do \( n \)-gram LMs rely on such short contexts? The primary reason for this is sparsity—the longer the sequence that the model conditions on, the rarer it becomes and higher the chance that it was not included in the training data. Hence, the model is unlikely to be able to carry out a simple look-up to estimate the next word distribution and would either incorrectly output a 0 probability for the true next word which would lead to infinite perplexity, or back-off to using a model that uses shorter contexts. Thus, sparsity makes \( n \)-gram LMs unable to model long-range dependencies and limits their ability to generalize.

Another limitation of \( n \)-gram LMs is their inability to generalize across contexts that are similar in meaning but lexically different. For instance, even though we know that two contexts like \emph{Pride and Prejudice was written by} and \emph{The author of Pride and Prejudice is} mean the same, an \( n \)-gram model is unable to recognize and capture this relationship due to its reliance on precise \( n \)-gram statistics. Thus, \( n \)-gram models do not
encode a notion of semantic similarity across many different contexts and rely on an ever-growing training corpus for extensive coverage of context-target pairs. Even though \( n \)-gram models are relatively easy to scale to larger corpora, this limitation makes them inefficient.

With the adoption of deep learning in NLP, language modeling performance has vastly improved, especially via modeling long-range dependencies and generalizing across semantically similar contexts. We extensively discuss neural LMs and their benefits in the rest of this chapter.

### 2.3 Neural Language Models

Deep learning is a form of machine learning that relies on the use of artificial neural networks to learn various kinds of tasks. They typically contain upwards of millions of parameters that are learned via training on large amounts of data. The key benefit of using neural networks lies in representation learning—rather than having to manually design features that the model can use to learn a mapping from inputs to outputs, neural networks automatically extract a range of features most relevant for learning the underlying task. Deep learning models typically consist of several layers of parameters such that feature representations in higher layers are computed by combining simpler feature representations found in lower layers. Given some input, like the prior context in the case of language modeling, these models map it to high-dimensional representations and then apply a series of linear and non-linear transformations to those representations, before making the final prediction. This is a forward pass through the network which constitutes the learned mapping from inputs to outputs. During training, the model parameters must be updated by minimizing the value of an objective function, known as the loss. This is done via a process known as backpropagation. After the forward pass on a batch of examples, the loss is computed and gradient-based updates for each parameter with respect to the loss function are propagated during the backward pass. This learned network is then used to make predictions on test data. For a detailed primer on deep learning and how neural networks are trained, see Goodfellow et al. (2016).

Neural LMs are multi-layer neural networks used for language modeling (Bengio et al., 2003; Mikolov et al., 2010). As opposed to \( n \)-gram models which are non-parametric and grow with the size of the training set to estimate probabilities using \( n \)-gram frequencies, neural LMs are parametric models trained via self-supervised learning (more on this in Section 2.3.1). Given the input contexts, these models learn continuous representations of the prior context which are then used to estimate next word probabilities. These continuous representations have been instrumental for learning generalizable functions and encode contextual information for sequences that are much longer than the 5-grams typically used by \( n \)-gram LMs. Neural LMs rely on context sizes of hundreds, or even thousands, of tokens when computing context representations and estimating next word distributions. In our example from Section 2.2, given the sequence *In the 1800s, novels from many genres were . . . . . . . . Pride and Prejudice which is a popular romantic*, a neural LM can encode long-range dependencies and thus use the information that the distant context is discussing *novels*, while estimating the next word distributions (as shown in Chapter 3).
Apart from being able to model long-range dependencies, unlike $n$-gram LMs, neural LMs can also generalize across semantically similar contexts given their parametric nature. For instance, in the case of the two contexts *Pride and Prejudice was written by* and *The author of Pride and Prejudice is*, neural LMs are highly adept at mapping these similar sequences to similar representations in feature space. This property makes them extremely effective at generalizing to unseen contexts that are semantically similar to previously seen examples, yet lexically very different—another key limitation of $n$-gram LMs. This is, in fact, an important idea in our work on generalization through memorization presented in Chapters 4-7 of this thesis, where we show that LM generalization can be vastly improved by explicitly exploiting this property. It has also been shown, via analysis, that most modern large-scale LMs learn representations which encode various notions of linguistic knowledge such as parts-of-speech, syntactic structure, coreference and more (Tenney et al., 2019; Hewitt and Manning, 2019; Clark et al., 2019).\(^2\)

In spite of their widespread adoption, a drawback of neural LMs as compared to $n$-gram models is that their predictions are fundamentally not interpretable. For example, with $n$-gram LMs it is quite clear how much context is used and which words impact predictions, whereas for neural LMs which use much longer contexts it is quite unclear which contextual features are useful. This is primarily due to the inherent black-box nature of neural networks which makes them non-interpretable. We investigate a set of hypotheses in relation to how neural LMs use context in Chapter 3 of this dissertation, on the road to making these models more interpretable over time.

Now we describe the self-supervised learning approach, which has been key to the success of neural LMs and provide an overview of how these models are structured and trained, before providing details on specific model architectures considered in this work.

2.3.1 The Self-Supervised Learning Approach

Supervised learning is the most popular approach used in machine learning where models learn functions that map inputs to outputs given labeled examples. Neural networks, in particular, have been extremely effective for supervised learning problems where large amounts of labeled data is available. For instance, the ImageNet dataset (Deng et al., 2009) for visual object recognition, which was instrumental in spurring the deep learning revolution for the computer vision research community, contains over 14 million labeled images. However, such massive datasets can be extremely expensive to curate. In addition, the NLP community has found that artificial data collection for textual tasks like entailment and question answering can be prone to artifacts that allow training models which achieve strong results by exploiting dataset artifacts rather than learning and generalizing based on the underlying task (Gururangan et al., 2018; Lewis et al., 2021). But, the task of language modeling is not susceptible to these challenges of data collection because it involves using naturally occurring text as inputs and training models via self-supervised learning.

In self-supervised learning, we are given raw, unlabeled data and a labeled dataset for supervised learning is created by choosing to interpret some of the values in the data as a label to be predicted. In the case\(^2\)Note that these analyses have been mainly conducted on masked language models, specifically BERT (Devlin et al., 2019).
of language modeling, this is a corpus of raw text where each word can be the label or prediction for its preceding context. Hence, the training set consists of labeled examples that are context-target pairs. Given this self-supervised nature of language modeling as well as the fact that we have access to massive amounts of naturally occurring text online, the scale of neural LMs has grown by many orders of magnitude within the last couple of years, which we further discuss in Section 2.5.

Before the use of language modeling for learning contextualized representations was popularized, self-supervised learning aided the first NLP revolution in the deep learning era through word2vec (Mikolov et al., 2013). Word2vec is a self-supervised approach for learning distributed representations of words where in the Skip-gram model, the training objective is to learn context-independent representations for words that are useful for predicting the surrounding context words. Thus, over the course of the last decade, self-supervised learning has become an integral step for representation learning in the typical NLP pipeline.

### 2.3.2 Training Neural Language Models

While in the case of the non-parametric \( n \)-gram LMs the probabilities are estimated based on \( n \)-gram frequencies, for parametric, multi-layer neural LMs this process consists of many steps. All neural LMs share some key components regardless of the underlying model architecture. We now discuss these components and the general training procedure for these models.

**Embedding Layer** Neural LMs take the prior context tokens \( x = (x_1, x_2, \ldots, x_t) \) as input and estimate the probability distribution for the next token \( y \). The first step is to map the input to context-independent embeddings which are high-dimensional representations, one for each word that appears in the input.\(^3\) Each embedding is meant to represent the meaning of that word based on the aggregate of the contexts in which it appears. Following the distributional hypothesis which states that similar words appear in similar contexts (Firth, 1957), words that appear in similar contexts lie close together in the vector space of embeddings, which is particularly beneficial for generalization to words appearing in previously unseen contexts.

Models typically use an embedding lookup table \( E \) which is shared across all examples and contains size of the vocabulary \( |V| \) number of vectors, each of \( d_{emb} \) dimensions. Thus, the first step is to map inputs to embedding representations \( E^T x \). In earlier models, the embedding layer had been randomly initialized and jointly trained with the network layers in an end-to-end fashion—error-based gradient updates were backpropagated all the way to the embedding layer and applied to words that appeared in the input examples (Bengio et al., 2003). But when pre-trained word embeddings such as such as word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) were proposed, they were used to initialize the word embedding layer and fine-tuned while training the rest of the network, leading to improved performance on a range of tasks beyond just language modeling. However, in recent years, the community has reverted back to learning the embedding layer with the rest of the network as the effects of pre-trained word embeddings have

---

\(^3\)Note that tokens refer to instances of words (or subwords) that appear in the vocabulary. The phrase *the cat sat on the mat* has 6 tokens but only 5 words since *the* appears twice.
faded in the presence of growing model and dataset sizes, which allow the model to learn high quality word representations from scratch.

Typically, the embeddings for all words have the same dimensionality but recently, Baevski and Auli (2019) introduced adaptive input representations where frequent words have embeddings with orders of magnitude more parameters than rare words. This helps to reduce the overall number of trainable parameters in the model, which is particularly useful when using large vocabularies.

**Network Layers** These are a series of linear and non-linear transformations applied to the input representations. We denote the composition of these functions as $f$. They vary across different model architectures like Recurrent Neural Networks, Convolutional Neural Networks, Transformers and further variants of each. The architectures studied in this dissertation are described in Section 2.4. The number of hidden layers $L$ signifies the depth of the network, and the parameters at each layer $W^l$ denote the linear transformation applied to the input at that layer. Then, a non-linearity, such as the commonly used ReLU function, is applied to the linearly transformed representations before being fed to the next layer as its input. These layers are seen as learning features of the prior context where lower layers learn simpler features and higher layers compose them to learn more complex features.

**Softmax Layer** Language modeling is a multinomial classification task where the number of output classes is the size of the vocabulary $|V|$. Thus, after mapping inputs to representations using the embedding layer $E$ and the $L$ hidden layers, the model output $f(x)$ must be mapped to a probability distribution over items of the vocabulary. This consists of two steps, (1) projecting $f(x)$ into a $|V|$-dimensional space, and (2) converting these projected representations into probabilities.

For projection, the softmax layer consists of weights $E^s \in \mathbb{R}^{d_{emb} \times |V|}$ that project $f(x)$ into a representation space that has the same size as the vocabulary. These weights can be thought of as a collection of word embeddings $\{e_1, e_2, \ldots, e_{|V|}\}$ similar to the embedding layer’s vectors. In fact, in recent years, sharing the word embedding layer and the softmax layer parameters has become the norm (Press and Wolf, 2017; Inan et al., 2017), such that $E^s = E$. The softmax function converts $f(x)$ into a probability distribution as follows:

$$P(y = w_i | x) = \frac{\exp(f(x)^T e_i)}{\sum_{j=1}^{|V|} \exp(f(x)^T e_j)}$$  \hspace{1cm} (2.4)

where $w_i$ is the $i$-th word in the vocabulary.

As the size of the vocabulary grows, the softmax layer can become a computational bottleneck for the neural model. In light of this, several approaches have been proposed to improve the efficiency of the softmax layer such as the hierarchical softmax (Morin and Bengio, 2005), noise contrastive estimation-based approaches (Mnih and Teh, 2012), adaptive softmax similar to the adaptive input representations described earlier (Grave et al., 2017b), mixture of softmaxes inspired by the mixture-of-experts model (Yang et al., 2018) and more.
Cross-Entropy Loss. During training, after generating the probability distribution, a loss term is further computed in order to calculate the error in the model’s prediction and use this to update the model parameters via backpropagation. For language modeling, we use the cross-entropy loss which is the same as the negative log-likelihood term provided in Equation 2.2. Then, based on the loss gradient-based updates are applied to every parameter in the network using the stochastic gradient descent (SGD) algorithm, or a variant that uses momentum and/or adaptive gradients like Adam.

Thus, we have described each of the components of our general neural LM. This model is trained until convergence such that the loss on a held-out validation set stops decreasing after enough iterations over the training data. For newer large-scale LMs that are trained on trillions of tokens of data, iterating over the entire training set may not be feasible and training ends after a fixed number of updates. Before providing details on specific model architectures that have been studied in this work, we briefly discuss using neural LMs for generation.

2.3.3 Generation using Language Models

When we want to use a neural LM for generation, it is trained in exactly the same way as the vanilla LM. This is known as teacher forcing. During inference, we can use the model to generate sequences by sampling tokens from the estimated conditional probability distributions, instead of relying on true prior context. We can use greedy decoding which samples the token with the maximum probability, or beam search decoding which maintains and expands the top $b$ sequences with the highest likelihood at every step. We can also use random sampling, or variants that attempt to improve the quality of generations for this approach such as top-$k$ sampling (Fan et al., 2018) or nucleus sampling (Holtzman et al., 2020). In this thesis, the neural machine translation models in Chapters 6 and 7 are the only ones that deal in text generation, and we use beam search decoding for all our models.

2.4 Model Architectures

In the previous section, we described the components of a general neural LM. While these basic building blocks are present in most models, the network layers can vary and their structure controls the capabilities of the underlying model. Here, we describe the two architectures, Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) and Transformer networks (Vaswani et al., 2017), that are investigated in this dissertation.

2.4.1 Long Short-Term Memory Networks

Recurrent Neural Networks. The LSTM network is a type of Recurrent Neural Network (RNN). RNN layers contain a recurrent connection such that information can be passed from one time step to the next.
This recurrent connection means that RNNs process information, like text, sequentially—one token at a time. At every time step, each layer combines the new input representation $e_t = E^T x_t$ with information from the previous time step $h_{t-1}$:

$$h_t = g(W_{hh} h_{t-1} + W_{hx} e_t + b_h)$$

(2.5)

where $W_{hh}$ and $W_{hx}$ are the layer weights for processing previous hidden states and new inputs, respectively, $b$ is the bias parameter and $g$ is a non-linearity. The recurrence means that these weights are tied across time steps and the same weights are used at every position. This is the general form of an RNN that can contain many stacked layers, as described in the note on network layers in Section 2.3.2. Due to the recurrent connection, training this model involves a slight modification to backpropagation, called backpropagation through time (BPTT). In BPTT, the gradients of the loss with respect to $W_{hh}$ and $W_{hx}$ are computed and aggregated at every time step of the sequence. Since sequences in language modeling tend to be very long, in practice a truncated version of BPTT is used where gradients are backpropagated to a fixed number of time steps.

An appealing property of RNNs is that by feeding information to future time steps, these models should capture long-range dependencies in sequential data. This property would be particularly useful for natural language, as shown by the examples in Section 2.3. However, while in theory RNNs should model dependencies that exist in the distant context, in practice they do not. They suffer from the exploding and vanishing gradients problem (Bengio et al., 1994; Pascanu et al., 2013) such that the gradients that are being backpropagated to time steps further in the history can often either grow very large resulting in large updates to weights and unstable model behavior, or shrink to very small values and have no impact on updating the weights. An intuitive explanation for this is that when computing the gradient updates, repeated multiplications with the same weights matrix where the largest eigenvalue is less than (or greater than) 1 causes the gradients to rapidly shrink (or grow) in the direction of that largest eigenvalue. Thus, exploding and vanishing gradients prevent RNNs from learning long-range dependencies. LSTMs, however, do not suffer from the vanishing gradients problem. They make structural changes to the vanilla RNN which we discuss next.

**Long Short-Term Memory Networks** LSTMs, like vanilla RNNs, contain recurrent connections. However, the structure of the recurrence is more complex than Equation 2.5. Specifically, LSTMs rely on a new cell state which persists across time steps with gated updates for what information is preserved and what is forgotten over time. The cell state also informs which information is relevant for the hidden state at the given time step.

First, we describe how new information is written to the cell state. This is governed by the input gate, as shown in Equation 2.6. The input gate, which owing to the sigmoid $\sigma$ contains values between 0 and 1, is used to control which elements of the cell state will be updated. $\tilde{c}_t$ in Equation 2.7 denotes the new candidate values that will be written to the cell state using the input gate.
\[ i_t = \sigma(W_{ih}h_{t-1} + W_{ix}e_t + b_i) \quad (2.6) \]
\[ \tilde{c}_t = \tanh(W_{ch}h_{t-1} + W_{cx}e_t + b_c) \quad (2.7) \]

Next, we describe how information from the cell state is forgotten, using the forget gate in Equation 2.8. This gate, which also consists of values between 0 and 1, controls which of the previous cell state elements will be masked out and thus, forgotten. Given the forget gate, the input gate and the new candidate values, the cell state can be updated as shown in Equation 2.9.

\[ f_t = \sigma(W_{fh}h_{t-1} + W_{fx}e_t + b_f) \quad (2.8) \]
\[ c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \quad (2.9) \]

Finally, we must determine what information is to be included in the hidden state \( h_t \) which serves as the LSTM layer’s output at the given time step. Synonymous to the input and forget gates, an output gate is computed (Equation 2.10) to determine which values from the cell should be included in the hidden state. Then, as the last step shown in Equation 2.11, the output gate and newly update cell state are used to determine the output hidden state for the current time step, which will also be fed to the next time step, as per the recurrence.

\[ o_t = \sigma(W_{oh}h_{t-1} + W_{ox}e_t + b_o) \quad (2.10) \]
\[ h_t = o_t \odot \tanh(c_t) \quad (2.11) \]

After \( L \) stacked LSTM layers, the output from the model \( f(x) \) is fed to a softmax layer, described in Section 2.3.2, which outputs the model’s final prediction. Depending on the underlying task, \( f(x) \) may be the output from the last time step or a combination of outputs from all the time steps.

This updated structure did help to improve the model’s ability to capture long-range dependencies and was widely used in the NLP community for a few years. However, as shown in our analysis in Chapter 3, the LSTM is limited and can only model the sequential relationships for a few hundred tokens in the prior context (Khandelwal et al., 2018). In addition, for the representation learning task, the sequential nature of the model made it extremely slow to train and deploy. In 2017, the Transformer network was introduced by Vaswani et al. (2017) which removed recurrent connections entirely and used self-attention instead to model information across long sequences. We discuss this model architecture next.
20

2.4.2 Transformer Networks

In this section, we describe the Transformer network which succeeded the LSTM as the most widely used model architecture in the NLP community. Transformers were originally introduced as an encoder-decoder architecture (Sutskever et al., 2014; Cho et al., 2014), as depicted in Figure 2.1. Although we only use the decoder for language modeling as proposed by Liu et al. (2018), we describe the complete encoder-decoder architecture, component-by-component, which is used for our neural machine translation models in Chapter 6.

Position Embeddings: Although the Transformer layers replace the Network Layers described in Section 2.3.2, they do add an additional component to the embedding layer. Since Transformers do not include recurrent connections, they need to track position information of tokens in the sequence in order to distinguish recent local context from distant long-range context. For this reason, the proposed model includes position embeddings \( P \) which encode information about the position of that token in the input sequence. Similar to the word embedding look-up table \( E \), the position embeddings are shared across examples and can either be learned or set according to a pre-defined function such as sinusoids, as proposed by Vaswani et al. (2017) who claim that they are useful for generalizing to longer sequences than those seen during training. The word and position embeddings are then combined to produce the final input representation \( x_e = E^T x + P^T x \) which is fed to the network layers.
CHAPTER 2. AN OVERVIEW OF LANGUAGE MODELS

Each Transformer network layer consists of two sub-layers: (1) a multi-head attention layer, and (2) position-wise feed-forward layer, which we describe below. Note that in the encoder-decoder architecture, the decoder also includes a separate multi-head cross-attention layer for incorporating encoder representations, as shown in Figure 2.1.

**Multi-Head Attention** As stated previously, the Transformer eliminates recurrent connection entirely and relies on self-attention connections to model pairwise relationships between tokens in the input sequence. Attention between two sequences, which was first introduced in the context of neural machine translation (Bahdanau et al., 2015), assigns a weight from every token in the target sequence to every token in the source sequence. These probability distributions govern how much each token in the source sequence contributes to the representation of each token in the target sequence.

More concretely, given the representation of the token in the target sequence \( q \) which we call the query, and a sequence of representations from the source sequence \( k_1, k_2, \ldots, k_n \) which we call the keys, we compute attention weights using dot-product attention as follows:

\[
\alpha_i = \frac{\exp(q^T k_i)}{\sum_{j=1}^{n} \exp(q^T k_j)} \quad (2.12)
\]

Then, based on these weights, the final representation is a linear combination of the representations of the sequence of values \( v_1, v_2, \ldots, v_n \), which may or may not be the same as the keys:

\[
h_{\text{attn}} = \sum_{i=1}^{n} \alpha_i v_i \quad (2.13)
\]

Self-attention, which is used in the Transformers, similarly assigns weights between tokens within the same sequence (Parikh et al., 2016; Lin et al., 2017). And this is done many times within the same layer, using several different attention heads. Given the sequence of input representations for the current layer \( x_{\text{inp}} \), they are first projected using different weights matrices for the different attention heads \( i = 1, \ldots, H \) to compute the queries \( Q \), keys \( K \) and values \( V \):

\[
Q_i = W_i^Q x_{\text{inp}}
\]

\[
K_i = W_i^K x_{\text{inp}}
\]

\[
V_i = W_i^V x_{\text{inp}}
\]

Then, the output representations \( h_i \) for each head are computed similarly to Equations 2.12 and 2.13, where \( \sigma \) denotes the softmax operation:

\[
h_i = \sigma \left( \frac{Q_i K_i^T}{\sqrt{d}} \right) V_i \quad (2.15)
\]

Given the set of output representations for all the heads \( h_1, h_2, \ldots, h_H \), we concatenate them and apply
one more projection to compute the final layer’s output:

\[ h_{\text{MHA}} = W_O [h_1, h_2, \ldots, h_H] \]  \hspace{1cm} (2.16)

The purpose of using multiple heads of attention is to project inputs into different vector spaces and model pairwise relations based on a variable set of features (Clark et al., 2019).

While the Transformer encoder uses self-attention as described above, the decoder uses a masked version of self-attention to prevent the network from conditioning on the future when learning to generate tokens autoregressively from left to right. Encoder-decoder cross-attention models relations between the source and target sequence tokens, as was originally intended by Bahdanau et al. (2015).

**Feed-Forward Network** Apart from the attention-based sub-layers, the Transformer also includes a fully-connected feed-forward network sub-layer, which consists of two affine projections separated by a ReLU non-linearity. Given the input representation to this sub-layer \( x_{\text{inp}} \), we compute the layer output as follows:

\[ h_{\text{FNN}} = W_2 \text{ReLU}(W_1 x_{\text{inp}} + b_1) + b_2 \]  \hspace{1cm} (2.17)

 Typically, \( W_1 \) projects the input to a much larger representation space which makes the underlying model more expressive. Within a layer, the same feed-forward network is applied to each position in the sequence.

Each sub-layer of the Transformer also makes use of residual connections (He et al., 2016) and layer normalization (Ba et al., 2016). A residual connection is a skip-connection—the Transformer feeds the input to every sub-layer as an input to the following sub-layer as well, by combining it with the current sub-layer output. For instance, the input of the multi-head attention layer is added into its output \( x_{\text{inp}} + h_{\text{MHA}} \) before passing it on to the next sub-layer. Layer normalization transforms representations such that they have zero mean and unit variance, which can speed up training time by addressing the instability caused by internal covariate shift (Ioffe and Szegedy, 2015). After \( L \) Transformer layers, the output of the model \( f(x) \) is fed to a softmax layer, described in Section 2.3.2, which outputs the model’s final prediction.

This concludes our description of the Transformer model. Due to the direct self-attention connections, we find that these models are able to model longer-range dependencies than LSTMs (see Section 3.6). In addition, due to the absence of recurrent connections this model is highly parallelizable and thus much faster to train, which makes it easier to scale to a larger number of parameters, as we see in the next section.

**Transformer LM** Before concluding this section, we note that the Transformer LM simply consists of the decoder without the encoder-decoder cross-attention sub-layer.
2.5 The Impact of Scale

In this section, we briefly discuss the impact of scale on progress in neural language modeling research. Given the highly parallelizable nature of Transformers, the adoption of this model architecture has led to the extensive exploration of scaling LMs to very large numbers of parameters. For instance, recent work on GPT-3 involved training an autoregressive Transformer LM that contained 175 billion parameters (Brown et al., 2020). Similarly, GShard is a Transformer-based multilingual translation model which employs sparsely-gated mixture-of-experts to train a 600 billion parameter model (Lepikhin et al., 2020). Given present-day hardware, such large models require specialized efforts to parallelize and tremendous amounts of compute to train.

While it has been shown that representation learning with large LMs has brought unprecedented improvements in generalization and demonstrated novel modeling behaviors such as few-shot learning capabilities, scaling models has also introduced additional challenges. For instance, training every single instance of these large models brings with it a huge environmental cost that could be detrimental in the long-run (Strubell et al., 2019; Schwartz et al., 2020; Henderson et al., 2020). Furthermore, training such large models can also cause privacy-related issues. For instance, recent work has shown that large LMs are able to memorize user-specific information such as names and addresses, which are then susceptible to adversarial attacks meant to extract that information from the model directly (Carlini et al., 2020). This would be particularly harmful for models that are trained on private and proprietary data that then stands to be stolen if the model is deployed. It is not yet clear how these models memorize data, and how much control we have over what is being memorized. Due to their black-box nature, these models are inherently not interpretable which was discussed in Section 1.1.1. This makes it more difficult to trust model predictions given that they have been shown to produce biased behavior (Nadeem et al., 2020). Beyond not being able to trust model predictions, it has also been recently shown that the improving quality of LM-based generations can sometimes make them indistinguishable from human-written text, and that this poses a serious threat of malicious actors using these models to generate and propagate fake news (Zellers et al., 2019). See Bender et al. (2021) for a more extensive discussion of the emerging challenges from scaling LMs.

In this dissertation, we explore an alternate approach to model scaling for improving generalization. Our approach, Generalization through Memorization, involves explicitly memorizing data samples, either from within the training data or from some other text collection, in an external memory and then during inference, retrieving examples that are similar to the test input to make similar predictions. This form of explicit memorization gives us control over the data that the model has access to and the memory, which adds interpretability to the model, can then be updated as needed. In addition, since our approach can be applied to any existing model it does not incur additional training costs. Note that memorization is orthogonal to model scaling and the two approaches can be combined. For more details on Generalization through Memorization, see Chapters 4-7 of this thesis.
How Neural Language Models Use Context

Neural language models (LM) have become ubiquitous and are used in a variety of NLP systems. And yet, we know very little about how they work in practice. In the past, there have been a number of hypotheses regarding how these models use the provided input context. For instance, it was believed that their ability to model long-range dependencies in faraway context was one of the key reasons for their success over non-neural approaches like \( n \)-gram LMs. However, in spite of the growing interest in these models, how they use the input prior context and what contextual features they rely on remained largely unexplored, and the hypotheses mostly unverified.

These hypotheses have been non-trivially challenging to verify because neural models are not interpretable. As discussed in Chapter 2, most \( n \)-gram models were used for short context windows spanning about 2-5 tokens and given their symbolic nature, it was always clear how much context was used and which tokens impacted the predictions. However, most neural LMs rely on hundreds or even thousands of tokens of prior context when estimating next word distributions. While these longer contexts allow the model to capture long-range dependencies when making predictions, they also contribute to obscuring which specific contextual features the model relies on for making decisions—the model is a black-box and thus, not interpretable. Understanding how these models use the context would not only add transparency to the model’s decisions but could also highlight their shortcomings and suggest ways to improve them.

Prior to the work presented here, analytic studies had begun to shed some light on the information encoded by Long Short-Term Memory (LSTM) networks. For instance, Adi et al. (2017) showed that these models could remember lengths of the sentences that were encoded, identities of words that appeared in those sentences, and the relative ordering of two words that appeared in those sentences, all by simply training a neural network that learned to map from input representations to each of these classification decisions. Linzen et al. (2016) showed that these models could capture some syntactic structures such as subject-verb
agreement via the number prediction task which involved comparing the probabilities the models assigned to the correct and incorrect number of the present-tense verb in the short sequence. Li et al. (2016) showed that these networks could model certain kinds of semantic compositionality such as negation and intensification via visualization. Karpathy et al. (2016) investigated long-range dependencies in character-level recurrent models to find interpretable cells within the networks that could capture, for instance, pairs of open-close brackets and quote marks. However, all of the previous work studied LSTMs at the sentence level, with the exception of Karpathy et al. (2016) who studied character-level models instead of word-level models, even though word-level LSTMs can potentially encode much longer contexts. Our goal is to complement the prior work to provide a richer understanding of the role of context, in particular, long-range context beyond a sentence.

In this chapter, we employ black-box analysis and empirically test a collection of hypotheses regarding how neural LMs, specifically LSTM-based models, use the provided prior context. The three key hypotheses that we investigate are: (1) the amount of prior context that neural LMs rely on when estimating next word distributions is limited, (2) neural LMs represent the different regions of the context that they use, differently, and (3) LSTMs can copy words from the prior context both with and without external caching mechanisms. In order to test these hypotheses, we design ablation tests which involve perturbing the prior context during evaluation to remove certain contextual features, and quantifying how the model’s performance deteriorates in the absence of that contextual information. Details for our experimental design and findings are presented in the rest of this chapter.

This work was first presented in the publication *Sharp Nearby Fuzzy Faraway: How Neural Language Models Use Context* at the Association for Computational Linguistics conference held in 2018 (Khandelwal et al., 2018).

## 3.1 Approach

Our goal is to investigate the effect of contextual features such as the length of context, word order and more, on LSTM performance. Thus, we use ablation analysis, during evaluation, to measure changes in model performance in the absence of certain contextual information.

Typically, when testing the language model on a held-out sequence of words, all tokens prior to the target
word are fed to the model; we call this the infinite-context setting. In this study, we observe the change in perplexity or when the model is fed a perturbed context $\delta(w_{t-1}, \ldots, w_1)$, at test time. $\delta$ refers to the perturbation function, and we experiment with perturbations such as dropping tokens, shuffling/reversing tokens, and replacing tokens with other words from the vocabulary.\footnote{It is important to note that we do not train the model with these perturbations. This is because the aim is to start with an LSTM that has been trained in the standard fashion, and discover how much context it uses and which features in nearby vs. long-range context are important. Hence, the mismatch in training and test is a necessary part of experiment design, and all measured losses are upper bounds which would likely be lower, were the model also trained to handle such perturbations.}

We use a standard LSTM language model, trained and finetuned using the Averaging SGD optimizer (Merity et al., 2018).\footnote{We analyze two datasets commonly used for language modeling, Penn Treebank (PTB)\textsuperscript{3} (Marcus et al., 1993; Mikolov et al., 2010) and Wikitext-2 (Wiki) (Merity et al., 2017). PTB consists of Wall Street Journal Code for our experiments available at \url{https://github.com/urvashik/lm-context-analysis} } The vanilla LSTM is augmented with dropout on recurrent connections, embedding weights, and all input and output connections (Wan et al., 2013; Gal and Ghahramani, 2016), weight tying between the word embedding and softmax layers (Inan et al., 2017; Press and Wolf, 2017) and variable length backpropagation sequences. We provide the key hyperparameter settings for the model in Table 3.2. These are the default settings suggested by Merity et al. (2018). We also augment the model with a cache only for Section 3.4.2, in order to investigate why an external copy mechanism is helpful.

![Table 3.2: Hyperparameter settings for the LSTM model used in our experiments (Merity et al., 2018).](image)

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>PTB</th>
<th>Wiki</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word Emb. Size</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>Hidden State Dim</td>
<td>1150</td>
<td>1150</td>
</tr>
<tr>
<td>Layers</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Optimizer</td>
<td>ASGD</td>
<td>ASGD</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Gradient clip</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Epochs (train)</td>
<td>500</td>
<td>750</td>
</tr>
<tr>
<td>Epochs (finetune)</td>
<td>500 (max)</td>
<td>750 (max)</td>
</tr>
<tr>
<td>Batch Size</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>Sequence Length</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>LSTM Layer Dropout</td>
<td>0.25</td>
<td>0.2</td>
</tr>
<tr>
<td>Recurrent Dropout</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Word Emb. Dropout</td>
<td>0.4</td>
<td>0.65</td>
</tr>
<tr>
<td>Word Dropout</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>FF Layers Dropout</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>$1.2 \times 10^{-6}$</td>
<td>$1.2 \times 10^{-6}$</td>
</tr>
</tbody>
</table>

\footnote{We analyze two datasets commonly used for language modeling, Penn Treebank (PTB)\textsuperscript{3} (Marcus et al., 1993; Mikolov et al., 2010) and Wikitext-2 (Wiki) (Merity et al., 2017). PTB consists of Wall Street Journal Code for our experiments available at \url{https://github.com/urvashik/lm-context-analysis} } The vanilla LSTM is augmented with dropout on recurrent connections, embedding weights, and all input and output connections (Wan et al., 2013; Gal and Ghahramani, 2016), weight tying between the word embedding and softmax layers (Inan et al., 2017; Press and Wolf, 2017) and variable length backpropagation sequences. We provide the key hyperparameter settings for the model in Table 3.2. These are the default settings suggested by Merity et al. (2018). We also augment the model with a cache only for Section 3.4.2, in order to investigate why an external copy mechanism is helpful.

We analyze two datasets commonly used for language modeling, Penn Treebank (PTB)\textsuperscript{3} (Marcus et al., 1993; Mikolov et al., 2010) and Wikitext-2 (Wiki) (Merity et al., 2017). PTB consists of Wall Street Journal Note that the version of the PTB dataset used for language modeling is different from the one originally proposed by Marcus et al. (1993). Words were lower-cased, numbers were replaced with a canonical token, punctuation was removed and newlines replaced with *eos* tokens. In addition, the vocabulary only consisted of the 10K most frequent words, the rest were replaced with the *unk*. \footnote{Public release of their code at \url{https://github.com/salesforce/awd-lstm-lm} }
news articles with 0.9M tokens for training and a 10K vocabulary. Wiki is a larger and more diverse dataset, containing Wikipedia articles across many topics with 2.1M tokens for training and a 33K vocabulary. Additional dataset statistics are provided in Table 3.1.

In this study, we present results only on the dev sets, in order to avoid revealing details about the test sets. However, we have confirmed that all results are consistent with those on the test sets. When testing the effect of ablations, we focus on comparing differences in the language model’s losses on the dev set, which is equivalent to relative improvements in perplexity. In addition, for all experiments we report averaged results from three models trained with different random seeds.

3.2 How much context is used?

LSTMs are designed to capture long-range dependencies in sequences (Hochreiter and Schmidhuber, 1997). In practice, LSTM language models are provided an infinite amount of prior context, which is as long as the test sequence goes. However, it is unclear how much of this history has a direct impact on model performance. In this section, we investigate how many tokens of context achieve a similar loss (or 1-2% difference in model perplexity) to providing the model infinite context. We consider this the effective context size.

LSTM language models have an effective context size of about 200 tokens on average. We determine the effective context size by varying the number of tokens fed to the model. In particular, at test time, we feed the model the most recent $n$ tokens:

$$\delta_{\text{truncate}}(w_{t-1}, \ldots, w_1) = (w_{t-1}, \ldots, w_{t-n}),$$

where $n > 0$ and all tokens farther away from the target $w_t$ are dropped.\footnote{Words at the beginning of the test sequence with fewer than $n$ tokens in the context are ignored for loss computation.} We compare the dev loss, which is the negative log-likelihood on the validation set, from truncated context, to that of the infinite-context setting where all previous words are fed to the model. The resulting increase in loss indicates how important the dropped tokens are for the model.

Figure 3.1a shows that the difference in dev loss, between truncated- and infinite-context variants of the test setting, gradually diminishes as we increase $n$ from 5 tokens to 1000 tokens. In particular, we only see a 1% increase in perplexity as we move beyond a context of 150 tokens on PTB and 250 tokens on Wiki. Hence, we provide empirical evidence to show that LSTM language models do, in fact, model long-range dependencies, without help from extra context vectors or caches.

Changing hyperparameters does not change the effective context size. NLM performance has been shown to be sensitive to hyperparameters such as the dropout rate and model size (Melis et al., 2018). To investigate if these hyperparameters affect the effective context size as well, we train separate models by varying the following hyperparameters one at a time: (1) number of timesteps for truncated back-propogation
CHAPTER 3. HOW NEURAL LANGUAGE MODELS USE CONTEXT

(a) Varying context size. (b) Frequent vs. infrequent words. (c) Different parts-of-speech for PTB. (d) Different parts-of-speech for Wiki.

Figure 3.1: Effects of varying the number of tokens provided in the context, as compared to the same model provided with infinite context. Increase in loss represents an absolute increase in negative log-likelihood over the entire corpus, due to restricted context. All curves are averaged over three random seeds. (a) The model has an effective context size of 150 on PTB and 250 on Wiki. (b) Infrequent words need more context than frequent words. (c), (d) Content words need more context than function words.

(2) dropout rate, (3) model size (hidden state size, number of layers, and word embedding size). In Figure 3.2, we show that while different hyperparameter settings result in different perplexities in the infinite-context setting, the trend of how perplexity changes as we reduce the context size remains the same.

3.2.1 Do different types of words need different amounts of context?

The effective context size determined in the previous section is aggregated over the entire corpus, which ignores the type of the upcoming word. Boyd-Graber and Blei (2009) have previously investigated the differences in context used by different types of words and found that function words rely on less context than...
Infrequent words need more context than frequent words. We categorize words that appear at least 800 times in the training set as frequent, and the rest as infrequent. Figure 3.1b shows that the loss of frequent words is insensitive to missing context beyond the 50 most recent tokens, which holds across the two datasets. Infrequent words, on the other hand, require more than 200 tokens.

Content words need more context than function words. Given the parts-of-speech of each word, we define content words as nouns, verbs and adjectives, and function words as prepositions and determiners. Figures 3.1c and 3.1d show that the loss of nouns and verbs is affected by distant context, whereas when the target word is a determiner, the model only relies on words within the last 10 tokens.

Discussion. Overall, we find that the model’s effective context size is dynamic. It depends on the target word, which is consistent with what we know about language, e.g., determiners require less context than nouns (Boyd-Graber and Blei, 2009). In addition, these findings are consistent with those previously reported for different language models and datasets (Hill et al., 2016; Wang and Cho, 2016).
CHAPTER 3. HOW NEURAL LANGUAGE MODELS USE CONTEXT

3.3 Nearby vs. long-range context

An effective context size of 200 tokens allows for representing linguistic information at many levels of abstraction, such as words, sentences, topics, etc. In this section, we investigate the importance of contextual information such as word order and word identity. Unlike prior work that studies LSTM embeddings at the sentence level, we look at both nearby and faraway context, and analyze how the language model treats contextual information presented in different regions of the context.

3.3.1 Does word order matter?

Adi et al. (2017) have shown that LSTMs are aware of word order within a sentence. We investigate whether LSTM language models are sensitive to word order within a larger context window. To determine the range in which word order affects model performance, we permute substrings in the context to observe their effect on dev loss compared to the unperturbed baseline. In particular, we perturb the context as follows,

$$\delta_{\text{permute}}(w_{t-1}, \ldots, w_{t-n}) = (w_{t-1}, \ldots, \rho(w_{t-s_1-1}, \ldots, w_{t-s_2}), \ldots, w_{t-n}) \quad (3.2)$$

where $\rho \in \{\text{shuffle}, \text{reverse}\}$ and $(s_1, s_2]$ denotes the range of the substring to be permuted. We refer to this substring as the permutable span. For the following analysis, we distinguish local word order, within 20-token permutable spans which are the length of an average sentence, from global word order, which extends

\footnotetext{We obtain part-of-speech tags using Stanford CoreNLP (Manning et al., 2014).}
(a) Perturb global order, i.e. all tokens in the context before a given point, in PTB. 
(b) Perturb global order, i.e. all tokens in the context before a given point, in Wiki. 

Figure 3.4: Shuffling and reversing the order of all context words beyond a certain distance from the target but within 300 tokens of context, relative to an unperturbed baseline, does not affect loss beyond 50 tokens. All curves are averages from three random seeds, where error bars represent the standard deviation.

beyond local spans to include all the farthest tokens in the history. We consider selecting permutable spans within a context of $n = 300$ tokens, which is greater than the effective context size.

**Local word order only matters for the most recent 20 tokens.** We can locate the region of context beyond which the local word order has no relevance, by permuting word order locally at various points within the context. We accomplish this by varying $s_1$ and setting $s_2 = s_1 + 20$. Figure 3.3 shows that local word order matters very much within the most recent 20 tokens, and far less beyond that.

**Global order of words only matters for the most recent 50 tokens.** Similar to the local word order experiment, we locate the point beyond which the general location of words within the context is irrelevant, by permuting global word order. We achieve this by varying $s_1$ and fixing $s_2 = n$. Figure 3.4 demonstrates that after 50 tokens, shuffling or reversing the remaining words in the context has no effect on the model performance.

In order to determine whether this is due to insensitivity to word order or whether the language model is simply not sensitive to any changes in the long-range context, we further replace words in the permutable span with a randomly sampled sequence of the same length from the training set. The gap between the permutation and replacement curves in Figure 3.4 illustrates that the identity of words in the far away context is still relevant, and only the order of the words is not.

**Discussion.** These results suggest that word order matters only within the most recent sentence, beyond which the order of sentences matters for 2-3 sentences (determined by our experiments on global word order).
After 50 tokens, word order has almost no effect, but the identity of those words is still relevant, suggesting a high-level, rough semantic representation for these faraway words. In light of these observations, we define 50 tokens as the boundary between nearby and long-range context, for the rest of this study. Next, we investigate the importance of different word types in the different regions of context.

![Effect of dropping content and function words on PTB](image1.png)  ![Effect of dropping content and function words on Wiki](image2.png)

Figure 3.5: Effect of dropping content and function words from 300 tokens of context relative to an unperturbed baseline. Dropping both content and function words 5 tokens away from the target results in a nontrivial increase in loss, whereas beyond 20 tokens, content words are far more relevant.

### 3.3.2 Types of words and the region of context

Open-class or content words such as nouns, verbs, adjectives and adverbs, contribute more to the semantic context of natural language than function words such as determiners and prepositions. Given our observation that the language model represents long-range context as a rough semantic representation, a natural question to ask is how important are function words in the long-range context? Below, we study the effect of these two classes of words on the model’s performance. Function words are defined as all words that are not nouns, verbs, adjectives or adverbs.

**Content words matter more than function words.** To study the effect of content and function words on model perplexity, we drop them from different regions of the context and compare the resulting change in loss. Specifically, we perturb the context as follows,

$$
\delta_{\text{drop}}(w_{t-1}, \ldots, w_{t-n}) = (w_{t-1}, \ldots, w_{t-s_1}, f_{\text{pos}}(y, (w_{t-s_1+1}, \ldots, w_{t-n})))
$$

(3.3)

where $f_{\text{pos}}(y, \text{span})$ is a function that drops all words with POS tag $y$ in a given span. $s_1$ denotes the starting offset of the perturbed subsequence. For these experiments, we set $s_1 \in \{5, 20, 100\}$. On average, there are slightly more content words than function words in any given text. As shown in Section 3.2, dropping
more words results in higher loss. To eliminate the effect of dropping different fractions of words, for each experiment where we drop a specific word type, we add a control experiment where the same number of tokens are sampled randomly from the context, and dropped.

Figure 3.5 shows that dropping content words as close as 5 tokens from the target word increases model perplexity by about 65%, whereas dropping the same proportion of tokens at random, results in a much smaller 17% increase. Dropping all function words, on the other hand, is not very different from dropping the same proportion of words at random, but still increases loss by about 15%. This suggests that within the most recent sentence, content words are extremely important but function words are also relevant since they help maintain grammaticality and syntactic structure. On the other hand, beyond a sentence, only content words have a sizeable influence on model performance.

3.4 To cache or not to cache?

As shown in Section 3.3.1, LSTM language models use a high-level, rough semantic representation for long-range context, suggesting that they might not be using information from any specific words located far away. Adi et al. (2017) have also shown that while LSTMs are aware of which words appear in their context, this awareness degrades with increasing length of the sequence. However, the success of copy mechanisms such as attention and caching (Bahdanau et al., 2015; Hill et al., 2016; Merity et al., 2017; Grave et al., 2017a,c) suggests that information in the distant context is very useful. Given this fact, can LSTMs copy any words from context without relying on external copy mechanisms? Do they copy words from nearby and long-range context equally? How does the caching model help? In this section, we investigate these questions by studying how LSTMs copy words from different regions of context. More specifically, we look at two regions of context, nearby (within 50 most recent tokens) and long-range (beyond 50 tokens), and study three categories of target words: those that can be copied from nearby context ($C_{near}$), those that can only be copied from long-range context ($C_{far}$), and those that cannot be copied at all given a limited context ($C_{none}$).

3.4.1 Can LSTMs copy words without caches?

Even without a cache, LSTMs often regenerate words that have already appeared in prior context. We investigate how much the model relies on the previous occurrences of the upcoming target word, by analyzing the change in loss after dropping and replacing this target word in the context.

**LSTMs can regenerate words seen in nearby context.** In order to demonstrate the usefulness of target word occurrences in context, we experiment with dropping all the distant context versus dropping only occurrences of the target word from the context. In particular, we compare removing all tokens after the 50 most recent tokens, (Equation 3.1 with $n = 50$), versus removing only the target word, in context of size $n = 300$:

$$
\delta_{\text{drop}}(w_{t-1}, \ldots, w_{t-n}) = f_{\text{word}}(w_{t}, (w_{t-1}, \ldots, w_{t-n})),
$$

(3.4)
Figure 3.6: Effects of perturbing the target word in the context compared to dropping long-range context altogether, on PTB. Error bars represent 95% confidence intervals. (a), (c) Words that can only be copied from long-range context are more sensitive to dropping all the distant words than to dropping the target. For words that can be copied from nearby context, dropping only the target has a much larger effect on loss compared to dropping the long-range context. (b), (d) Replacing the target word with other tokens from vocabulary hurts more than dropping it from the context, for words that can be copied from nearby context, but has no effect on words that can only be copied from far away.

where \( f_{\text{word}}(w, \text{span}) \) drops words equal to \( w \) in a given span. We compare applying both perturbations to a baseline model with unperturbed context restricted to \( n = 300 \). We also include the target words that never appear in the context (\( C_{\text{none}} \)) as a control set for this experiment.

The results show that LSTMs rely on the rough semantic representation of the faraway context to generate \( C_{\text{far}} \), but directly copy \( C_{\text{near}} \) from the nearby context. In Figures 3.6a and 3.6c, the long-range context bars show that for words that can only be copied from long-range context (\( C_{\text{far}} \)), removing all distant context is far more disruptive than removing only occurrences of the target word (12% and 2% increase in perplexity,
Currently displaying: attn_vis_data.json

**Calloway** said <unk> excluding the British snack-food business acquired in July snack-food international UNK jumped NUM UNM with sales strong in Spain Mexico and Brazil <unk> total snack-food profit rose NUM UNM <unk> led by Pizza Hut and UNK Bell restaurant earnings increased about NUM UNM in the third quarter on a NUM UNM sales increase <unk> UNK sales for pizza hut rose about NUM UNM while UNK bell’s increased NUM UNM as the chain continues to benefit from its UNK strategy <unk> UNK Bell has turned around declining customer counts by permanently lowering the price of its UNK <unk> same UNK for Kentucky fried chicken which has struggled with increased competition in the fast-food chicken market and a lack of new products rose only NUM UNM <unk> the operation which has been slow to respond to consumers’ shifting UNK away from fried foods has been developing a UNK product that may be introduced nationally at the end of next year <unk> the new product has performed well in a market test in Las Vegas Nev. **Calloway**

(a) Success of neural cache on PTB.

La **Fortuna** Mexico. UNK just off the coast of Mexico, the system interacted with land and began weakening. UNK later, convection rapidly diminished as dry air became entrained in the circulation. In response to quick degradation of the system’s structure, the NHC downgraded UNK to a tropical storm. Rapid weakening continued throughout the day and by the evening hours, the storm no longer had a defined circulation. Lacking an organized center and deep convection, the final advisory was issued on UNK. The storm’s remnants persisted for several more hours before dissipating roughly 175 mi (280 km) southwest of Cabo Corrientes, Mexico. <unk> <unk> = = Preparations and impact = = <unk> <unk> Following the classification of Tropical Depression Two <unk> E on June 19, the Government of Mexico issued a tropical storm warning for coastal areas between UNK and Manzanillo. A hurricane watch was also put in place from UNK de UNK to Punta San UNK. Later that day, the tropical storm warning was upgraded to a hurricane warning and the watch was extended westward to La **Fortuna**

(b) Success of neural cache on Wiki.

![Figure 3.7: Successes of neural caching model where brightly shaded region shows peaky distribution.](image)

respectively). This suggests that the model relies more on the rough semantic representation of faraway context to predict these $C_{far}$ tokens, rather than directly copying them from the distant context. On the other hand, for words that can be copied from nearby context ($C_{near}$), removing all long-range context has a smaller effect (about 3.5% increase in perplexity) as seen in Figures 3.6a and 3.6c, compared to removing the target word which increases perplexity by almost 9%. This suggests that these $C_{near}$ tokens are more often copied from nearby context, than inferred from information found in the rough semantic representation of long-range context.

However, is it possible that dropping the target tokens altogether, hurts the model too much by adversely affecting grammaticality of the context? We test this theory by replacing target words in the context with other words from the vocabulary. This perturbation is similar to Equation 3.4, except instead of dropping the token, we replace it with a different one. In particular, we experiment with replacing the target with <unk>, to see if having the generic word is better than not having any word. We also replace it with a word that has the same part-of-speech tag and a similar frequency in the dataset, to observe how much this change confuses the model. Figures 3.6b and 3.6d shows that replacing the target with other words results in up to a 14% increase in perplexity for $C_{near}$, which suggests that the replacement token seems to confuse the model far more than when the token is simply dropped. However, the words that rely on the long-range context, $C_{far}$, are largely unaffected by these changes, which confirms our conclusion from dropping the target tokens: $C_{far}$ words are predicted from the rough representation of faraway context instead of specific occurrences of certain words.
offerings outside the u.s. & co. will manage the offering & co. macmillan said berlitz intends to pay quarterly dividends on the stock & co. the company said it expects to pay the first dividend of num cents a share in the num first quarter & co. berlitz will borrow an amount equal to its expected net proceeds from the offerings plus $ num million in connection with a credit agreement with lenders & co. the total borrowing will be about $ num million the company said & co. proceeds from the borrowings under the credit agreement will be used to pay an $ num million cash dividend to macmillan and to lend the remainder of about $ num million to maxwell communications in connection with a unk note & co. proceeds from the offering will be used to repay borrowings under the short-term parts of a credit agreement & co. berlitz which is based in princeton n.j. provides language instruction and translation services through more than num language centers in num countries & i in the past five years more than num num of its sales have been outside the u.s. & co. caliper has owned berlitz since num & co. in the first six months

(a) Failure of neural cache on ptb.

). Standing roughly 15 metres ( 49 ft ) away , the cadres now raised their weapons . " you have taken our land , " one of them said . " please don ’ t shoot us ! " one of the passengers cried , just before they were killed by a sustained burst of automatic gunfire . & co. having collected water from the nearby village , unk and his companions were almost back at the crash site when they heard the shots . unk it was personal ammunition in the luggage exploding in the heat , they continued on their way , and called out to the other passengers , who they thought were still alive . this alerted the insurgents to the presence of more survivors ; one of the guerrillas told unk ’ s group to " come here . " the insurgents then opened fire on their general location , prompting unk and the others to flee . hill and the unk also ran ; they revealed their positions to the fighters in their unk , but successfully hid themselves behind a ridge . after hill and the others had hidden there for about two hours

(b) Failure of neural cache on wiki.

Figure 3.8: Failures of neural caching model where lightly shaded regions show flat distribution.

3.4.2 How does the cache help?

If lstms can already regenerate words from nearby context, how are copy mechanisms helping the model? We answer this question by analyzing how the neural cache model (Grave et al., 2017c) helps with improving model performance. The cache records the hidden state \( h_t \) at each timestep \( t \), and computes a cache distribution over the words in the history as follows:

\[
P_{cache}(w_t | w_{t-1}, \ldots, w_1; h_t, \ldots, h_1) \propto \sum_{i=1}^{t-1} \mathbb{1}[w_i = w_t] \exp(\theta h_i^T h_t),
\]

where \( \theta \) controls the flatness of the distribution. This cache distribution is then interpolated with the model’s output distribution over the vocabulary. Consequently, certain words from the history are upweighted, encouraging the model to copy them.

Caches help words that can be copied from long-range context the most. In order to study the effectiveness of the cache for the three classes of words \( (C_{near}, C_{far}, C_{none}) \), we evaluate an lstm language model with and without a cache, and measure the difference in perplexity for these words. In both settings, the model is provided all prior context (not just 300 tokens) in order to replicate the Grave et al. (2017c) setup. The amount of history recorded, known as the cache size, is a hyperparameter set to 500 past timesteps for ptb and 3,875 for wiki, both values very similar to the average document lengths in the respective datasets.

We find that the cache helps words that can only be copied from long-range context (\( C_{far} \)) more than words that can be copied from nearby (\( C_{near} \)). This is illustrated by Figure 3.9 where without caching, \( C_{near} \) words see a 22% increase in perplexity for ptb, and a 32% increase for wiki, whereas \( C_{far} \) see a 28% increase in perplexity for ptb, and a whopping 53% increase for wiki. Thus, the cache is, in a sense, complementary
Figure 3.9: Model performance relative to using a cache. Error bars represent 95% confidence intervals. Words that can only be copied from the distant context benefit the most from using a cache.

to the standard model, since it especially helps regenerate words from the long-range context where the latter falls short.

However, the cache also hurts about 36% of the words in PTB and 20% in Wiki, which are words that cannot be copied from context ($C_{\text{none}}$), as illustrated by bars for “none” in Figure 3.9. We also provide some case studies showing success (Fig. 3.7) and failure (Fig. 3.8) modes for the cache. We find that for the successful case, the cache distribution is concentrated on a single word that it wants to copy. However, when the target is not present in the history, the cache distribution is more flat, illustrating the model’s confusion, shown in Figure 3.8a. This suggests that the neural cache model might benefit from having the option to ignore the cache when it cannot make a confident choice.

### 3.5 Discussion

The presented findings provide a great deal of insight into how LSTMs model context. This information can prove extremely useful for improving language models. For instance, the discovery that some word types are more important than others can help refine word dropout strategies by making them adaptive to the different word types. Results on the cache also show that we can further improve performance by allowing the model to ignore the cache distribution when it is extremely uncertain, such as in Figure 3.8a. Differences in nearby vs. long-range context suggest that memory models, which feed explicit context representations to the LSTM (Ghosh et al., 2016; Lau et al., 2017), could benefit from representations that specifically capture information orthogonal to that modeled by the LSTM.

In addition, the empirical methods used in this study are model-agnostic and can generalize to models other than the standard LSTM. This opens the path to generating a stronger understanding of model classes beyond test set perplexities, by comparing them across additional axes of information such as how much context they use on average, or how robust they are to shuffled contexts.
Chelba et al. (2017), in proposing a new model, showed that on PTB, an LSTM language model with 13 tokens of context is similar to the infinite-context LSTM performance, with close to an 8% \(^6\) increase in perplexity. This is compared to a 25% increase at 13 tokens of context in our setup. We believe this difference is attributed to the fact that their model was trained with restricted context and a different error propagation scheme, while ours is not. Further investigation would be an interesting direction for future work.

Given the empirical nature of this study and the fact that the model and data are tightly coupled, separating model behavior from language characteristics, has proved challenging. More specifically, a number of confounding factors such as vocabulary size, dataset size etc. make this separation difficult. In an attempt to address this, we have chosen PTB and Wiki - two standard language modeling datasets which are diverse in content (news vs. factual articles) and writing style, and are structured differently (eg: Wiki articles are 4-6x longer on average and contain extra information such as titles and paragraph/section markers). Making the data sources diverse in nature, has provided the opportunity to somewhat isolate effects of the model, while ensuring consistency in results. An interesting extension to further study this separation would lie in experimenting with different model classes and even different languages.

Thus, carefully designed black-box analyses can be extremely useful for studying aggregate model behaviors measured across datasets and model configurations. This approach to analysis provides a general framework which makes it easier to standardize analyses and the axes of comparison. For instance, our study employs the use of different datasets, different model configurations and even different random seeds which allows us to eliminate confounds and compare solely on the basis of which contextual features correlate with better performance. Newer analyses conducted on the Transformer LM, presented in Section 3.6, are congruous with our earlier work because the black-box being investigated can easily be switched out for another and still be compared across the same contextual features. Thus, black-box testing plays a vital role in the model analysis methodologies toolkit, complementary to the vast majority of approaches that aim to interpret individual internal components of neural models. The benefits of comparing architectures on varied datasets, model configurations and random seeds extends to approaches beyond black-box analysis as well. They contribute to making analyses more comprehensive and immune to the pitfalls of spurious conclusions drawn based on anecdotal observations which are often untenable (Leavitt and Morcos, 2020). With the growing interest in analyzing neural LMs, and deep models more generally, it is crucial that studies are conducted in a scientifically sound manner to avoid noise and allow steady progress towards building better models that are more interpretable, a key goal for progress in NLP research. We hope that work presented in this chapter contributes to and inspires a larger collection of robust model analyses.

### 3.6 Transformer Language Models

As noted in Chapter 2, at the time of writing, Transformer models (Vaswani et al., 2017) have become the state-of-the-art for most NLP tasks, including language modeling. Transformer decoders, which are the

\(^6\)Table 3, 91 perplexity for the 13-gram vs. 84 for the infinite context model.
variant of the original network used for language modeling, were only proposed following the work presented in this chapter (Liu et al., 2018). Since similar experiments and findings have been reported for Transformer LMs in recent years, we briefly discuss them here.

**How much context is used?** Recall that Transformer decoders rely on self-attention connections which allow them to model long-range dependencies in the prior context. As we have seen in Section 3.2, LSTM LMs are provided infinite prior context during evaluation, even though their effective context size is only 200 tokens. In contrast, Transformer LMs make predictions using finite prior context windows. Thus, the training context size is a key hyperparameter that influences the investigation of how much context the model relies on when making predictions.

Baevski and Auli (2019) have shown when training a Transformer LM on the WIKITEXT-103 corpus\(^7\) that longer sequence lengths result in better performance—the longest training context window used in their experiments was 3,072 tokens. In addition, they have shown that providing all tokens with a minimum context size during evaluation results in small performance improvements—for the training context size of 3,072 tokens, they provided a minimum extra context of 2,560 while evaluating a sequence of 512 tokens simultaneously. Note this is different from our setting where we evaluate a single token at a time. More recently, Press et al. (2020) have shown that when evaluating a single token at a time, the improvement from increasing the training context size from 1,024 and 3,072 tokens is only 0.05 perplexity.

These results show that while longer contexts improve performance, there are diminishing returns. Further experiments that control the training context size and evaluate a range of minimum context windows could provide a clearer and more precise understanding of how much context Transformer LMs rely on. In addition, confirming results across different datasets could be helpful to decouple the effects of Wikipedia articles from the capabilities of the model itself.

**Nearby vs. Distant Context** Given that Transformer LMs tend to rely on thousands of tokens of prior context, do they represent nearby and distant contexts differently, similar to LSTM LMs? Given the direct self-attention connections to the distant prior context, it is unclear if these models also estimate next word distributions by ignoring the structure of sequences that appear further away in the history.

Recently, Subramanian et al. (2020) have analyzed the behavior of Transformer LMs using the approach proposed in this chapter. More specifically, they measure the effect of global word order on the Transformer LM performance for a training context size of 512 tokens. Similar to our setting in Section 3.3, they perturb the prior context during evaluation by shuffling the words beyond a certain distance from the target, and measure the change model perplexity. In their experiments, they find that similar to LSTMs, word order for Transformers as evaluated on the WIKITEXT-103 and BOOKCORPUS (Zhu et al., 2015) datasets does not alter performance significantly beyond 50 tokens, defined as the local context in our study.

These results show that Transformers and LSTMs appear to similarly rely on word order in the local context and ignore it in the long-range context. However, it remains unclear whether these observations are

---

7This corpus (Merity et al., 2017) contains 103M tokens in the training set, and uses the same dev and test sets as WIKITEXT-2
due to properties of the underlying language or of the model architectures. Comparing local word order effects between different models, as well as running experiments on other languages could help to decouple the two and provide a clearer understanding of how these models use the prior context when estimating next word distributions.

**Continuous Caching Copy Mechanism**  In Section 3.4, we studied the effect of the continuous caching mechanism proposed by Grave et al. (2017c) and found that it is particularly helpful in allowing LSTM LMs to copy words from the distant context. Is the caching mechanism similarly helpful for Transformer LMs? Given that Transformer LMs can capture long-range dependencies via their self-attention connections, the hypothesis is that adding an extra mechanism to attend to tokens in the history would not provide much improvement in performance. This hypothesis has been verified in our work on memorization (Khandelwal et al., 2020), discussed in Chapter 5. Using the Transformer LM from Baevski and Auli (2019) which was trained using a context size of 3,072 tokens, we augment the model with a continuous cache that attends over the entire prior context. Experiments show that on the WIKITEXT-103 benchmark, the continuous cache improves model perplexity by only 0.3 points.

### 3.7 Conclusion

In the analytic study presented in this chapter, we have empirically shown that a standard LSTM language model can effectively use about 200 tokens of context on two benchmark datasets, regardless of hyperparameter settings such as model size. It is sensitive to word order in the nearby context, but less so in the long-range context. In addition, the model is able to regenerate words from nearby context, but heavily relies on caches to copy words from far away.

These findings demonstrate that analysis of existing models makes them more interpretable in two ways: (1) it adds transparency to model predictions, for instance, by demonstrating which contextual features contribute to the estimation of next word distributions by neural LMs, and (2) it suggests ways in which we can improve our models, for instance, by highlighting the need for hierarchical contexts in cases where the distant history needs to contribute to model predictions such as for long-form document understanding and generation. Given the empirical evidence that these models rely on local and long-range contexts at different granularities, the analysis also suggests that LMs which map similar contexts to similar representations might be encoding a more granular notion of similarity at a local and/or global level. This idea, in fact, plays an important role in Chapters 4-7 of this dissertation where we propose a model to memorize prior contexts and then retrieve them during inference based on similarity in the context representation space.
Chapter 4

Generalization through Memorization

4.1 Motivation

Improving model generalization is a key goal for progress in NLP research. When we train a machine learning model, we want it to be able to generalize to data that it has not seen before. Yet, most existing NLP models generalize inconsistently. Neural language models (LM), in particular, have shown inconsistent generalization in both cases where the train and test data come from the same distribution as well as when they come from different distributions. In this dissertation, we propose an approach based on nearest neighbor retrieval for improving neural LM generalization, both in- and out-of-distribution (See Chapters 5 and 7). We also apply this approach to neural machine translation (MT) models, which rely on LMs for generating translations autoregressively, thus demonstrating the generality of our proposed approach (See Chapters 6 and 7).

In the era of model scaling, where larger neural LMs have consistently demonstrated improving generalization capabilities, some key challenges that remain.

- **Implicit memorization and capacity limitations**: In Chapter 3, we saw that model analysis can improve our understanding of how neural LMs make predictions. Other recent analyses have shown that our state-of-the-art models might be memorizing information in their parameters during training (Petroni et al., 2019; Roberts et al., 2020). This form of implicit memorization is brittle because when the model parameters are updated it is unclear how much of the information is preserved and how much gets overwritten. It is also unclear how much we can control what the model memorizes in its parameters which is especially important because model capacity is limited and hence, the model would be unable to continue memorizing information infinitely. This capacity limitation can be particularly harmful for the long-tail of events where sparsity of data causes a failure to generalize.

- **Failure to adapt**: Recent work has also shown that neural LMs are unable to adapt to distributional shifts in data such as newly emerging facts over time (Lazaridou et al., 2021). This, in turn, hurts the
models’ ability to generalize in changing environments.

- **Computationally expensive**: Another limitation of existing neural LMs is that, given present day model architectures and hardware, these models require tremendous amounts of compute to train (Strubell et al., 2019; Schwartz et al., 2020; Henderson et al., 2020).

- **Lack of interpretability**: As described in Section 1.1.1, the lack of interpretability of neural LMs, and deep learning models more generally, is one of the key challenges facing the research community. The existence of biased and unfair predictions coupled with a very poor understanding of how models make predictions can result in unintended harmful behaviors. Hence, addressing this challenge will be vital on the road to making these models more reliable and trustworthy.

In light of these challenges which make existing models unreliable and expensive, we propose Generalization through Memorization—a framework to improve model generalization via explicit memorization of examples in an external store, without any updates to the model parameters. More specifically, memorization refers to the process of saving examples in an external memory and when given a test sample, retrieving similar examples from that memory in order to make similar predictions for the new input. While at first it may seem like a paradox that memorizing can help the model generalize to test instances, it is quite an intuitive approach to match new data to similar previously seen examples. Intuitively, for instance, any English speaker knows that the phrases *Pride and Prejudice was written by* and *The author of Pride and Prejudice is* mean the same, even if they do not know about *Jane Austen*. We hypothesize that neural LMs, which can be seen as solving two subproblems: (1) mapping prior context to fixed-sized representations, and (2) using these representations to predict the next word, behave similarly by encoding the fact that similar contexts like *Pride and Prejudice was written by* and *The author of Pride and Prejudice is* should have similar next word distributions. Hence, the representation learning problem should be easier than the prediction problem. Our proposed approach of memorization is based on this hypothesis which suggests that being able to retrieve relevant previously seen examples during inference should improve model generalization. Furthermore, access to an external memory containing examples that are used for making predictions is particularly useful in that it allows models that are (1) *expressive*, because they are non-parametric in nature and can use an arbitrary amount of data at test time; (2) *adaptable*, because predictions can be controlled by changing what is stored in the memory, and (3) *interpretable*, because the data used to make the prediction can be directly inspected.

In the rest of this chapter we first draw connections between our proposed memorization model and exemplar models, particularly those applied to NLP systems prior to the adoption of deep learning. Then, we describe a general version of the memorization framework and discuss the contributions made in this part of the dissertation.
4.2 Exemplar Models

Our proposed approach of memorization has evolved from the widely studied exemplar models proposed in the fields of Cognitive Science and Psychology (Medin and Schaffer, 1978; Nosofsky, 1987). Exemplar theory hypothesizes that humans categorize information by storing specific examples for each of the categories. Then, when making new classification decisions they match features of new instances to the exemplars already stored in memory. This is in contrast to abstraction-based models which propose that humans store an abstract representation for each of the categories rather than saving exact examples, and use these representations to generalize to new instances (eg: Rosch, 1973; Anderson, 1991). In this dissertation, we show that this idea of exemplar models or memorization is a particularly interesting approach to study in conjunction with deep learning. But before describing our proposed framework of memorization for neural models in Section 4.3, we look back at how ideas from exemplar theory have influenced research in NLP in the past.

Before deep learning was widely adopted, exemplar models have been explored in artificial intelligence research, sometimes known as memory-based or case-based reasoning (Stanfill and Waltz, 1986) and in machine learning, sometimes known as memory-based or instance-based learning (Mitchell, 1997). A key advantage of such methods, as noted by Mitchell (1997), is that predictions for new instances can be made locally rather than having to solely rely on the most general learned patterns. This type of local neighborhood-based classification using similar instances is a key property of the $k$-nearest neighbors algorithm (Fix and Hodges Jr, 1952; Cover and Hart, 1967) which is very commonly used for memory-based models, including in our study.

Memory-based models have been proposed and extensively explored in the field of NLP as well. Daelemans and van den Bosch (2005) describe decades of progress on memory-based models in NLP, and cite shortcomings of statistical models as one of the primary motivations for exploring these non-parametric approaches. They describe the complex nature of language which consists of an ever-increasing set of subregularities and exceptions as an important source of knowledge that statistical methods are unable to leverage, and that memory-based models are particularly effective for using this low-frequency information.

Memory-based NLP models typically consisted of two parts: (1) storing feature-based representations of examples or experiences in a memory, and (2) reusing solutions from similar previous experiences to solve new problems. Older case-based reasoning systems organized examples into symbolic hierarchical tree structures (see, for instance, the Exemplar Memory Organization Packets proposed by Kolodner (1980)), and look-ups involved searching for similar cases based on the pre-defined notions of similarity. Newer memory-based classifiers used more sophisticated feature-based representations to represent exemplars (Daelemans et al., 2004). For example, the kinds of features used in the case of memory-based part-of-speech (POS) tagging included left- and right-side context words and their POS tags, capitalization, the presence of numerical characters, morphological features etc. Retrieval of similar instances involved using $k$NN based on weighted similarity scores, where each feature was assigned a separate weight. In the case of example-based machine translation, memories were constructed to support phrase-based retrieval using parallel corpora, bilingual lexicons, syntactic features and more (Planas and Furuse, 1999; Way and Gough, 2003; Carl and Rascu, 2006).
and some systems even used them in conjunction with statistical models (Imamura et al., 2004; Groves and Way, 2005). This illustrates the steady evolution of exemplar representations from being symbolic to becoming feature-based and hence, more generalizable.

Such memory-based models have been explored in the context of many NLP tasks. Memory-based classifiers have been applied to a range of language understanding tasks such as morphological tagging, part-of-speech tagging, prepositional phrase attachment and word-sense disambiguation (Daelemans et al., 1996; Zavrel and Daelemans, 1997; Daelemans et al., 1999; van den Bosch and Daelemans, 1999; Decadt et al., 2004). For machine translation, Nagao (1984) proposed example-based MT for translating sequences by analogy. This approach was extended to sub-sentential alignment of retrieved sentence pairs (Brown, 1996), identifying portions of the training data that could be a translation based on edit distance (Doi et al., 2005), matching training examples based on local trigram contexts (van den Bosch et al., 2007b), using phrase-based memories (van Gompel et al., 2010), incorporating syntactic features when retrieving similar examples (Stroppa et al., 2007; Haque et al., 2009), and combining memory-based aligners with memory-based LMs to score generations post-hoc (van Den Bosch and Berck, 2009). See Carl and Way (2003) for a broader discussion on advances that were made in example-based translation models.

Since memory-based models rely on saving and retrieving exemplars, the contents and coverage of the memory require special consideration. Given that the $k$NN algorithm is non-parametric and asymptotically consistent, it is unsurprising that performance of memory-based models, which used $k$NN, seemed to improve with the use of larger memories that provided better coverage of the space of examples. In fact, Daelemans et al. (1999) noted that forgetting examples that were exceptions, and provided low-frequency information, was especially harmful for the generalization of these systems, owing to the highly irregular nature of language. However, larger memories are computationally expensive and thus, ways to compress them were necessarily explored. This included automated editing of memories to remove useless or harmful exemplars which proved challenging (Daelemans et al., 1999), as well as approaches related to clustering for merging similar instances into a single substructure and searching for exemplars hierarchically (van den Bosch, 1999; Brown, 2000).

Limitations In spite of this progress in proving the utility of memory-based models in NLP, this paradigm was not as widely adopted as statistical parametric models. One of the limitations was the substantial dependence of retrieval quality on manually designed feature vectors. When saving examples into external memories, most systems relied on discrete representations which required careful feature engineering, as was earlier demonstrated by the POS tagging example. Such feature spaces were often highly complex—they involved the use of a combination of categorical features, numerical ones, case-based ones etc. Measuring similarity in such complex features spaces posed some serious challenges, such as redundancy, noisy features and missing entries, and involved carefully designing a whole host of similarity metrics. Generation tasks often also required carefully designed post-processing approaches. For instance, in example-based machine translation, a widely known problem was that of boundary friction where after retrieving translation pairs for sub-parts of a sentence, they needed to be carefully stitched together in order to form a coherent output.
Another limitation, as discussed previously, was the size of the memory, which continued to serve as a computational bottleneck and prevented scaling this approach for better coverage and improved generalization. While smaller memories are less likely to capture the long-tail of irregularities in language, saving more exemplars requires more storage space and the index is also more expensive to search. But the use of discrete features made compression of the representations challenging and removing less helpful exemplars was shown to be ineffective.

Moreover, the pure memory-based approaches relied solely on the contents of the memory and the efficacy of the chosen features. For instance, if the memory used for POS tagging did not contain a very similar example to the new test instance, then there was no way for the model to reliably make a prediction. This further underscored the limitations of smaller memories and the challenges in scaling the approach.

In this thesis, we show that the long-standing idea of memorization can be extremely effective for neural NLP models since continuous representations are helpful in addressing some of the aforementioned limitations. First, using neural representations for retrieval can automatically encode an extremely broad set of features useful for the task (as seen in Chapter 3 for language models), and generalize the notion of similarity across a highly diverse set of inputs which can easily be extended, for example, to data from different distributions (see Chapter 7). In addition, these neural representations are amenable to quantization and compression, such that memories can be easily scaled to billions of examples with the use of fast approximate nearest neighbor search methods. In addition, our proposed semi-parametric approach combines memory-based predictions and abstract representations learned by the neural network parameters, in order to effectively generalize to both test instances with neighbors in the memory, as well as to entirely novel constructions. We now describe a general framework for memorization that can be used in conjunction with any class of neural models and applied to a broad range of tasks.

### 4.3 The Memorization Framework for Neural Models

In this section, we describe the proposed memorization model, which uses a $k$-nearest neighbor classifier, at a high level and in a task-agnostic fashion. Suppose we are given a pre-trained neural network-based classification model $f$ parametrized by $\theta$, which we call the base model. Suppose also that we are given a dataset of input-output pairs $D = (x_i, y_i)$ for $i = 1, \ldots, m$ which can be memorized. Note that here, the inputs $x_i$ are not necessarily sequences of text, but an arbitrary input from audio signals or images, to sequences of actions from a reinforcement learning agent. Also note, that this data can be the training data that $f$ is trained on, or any other collection of examples for the same task.

Memorization entails constructing an external memory called a datastore which will contain the provided data that must be memorized. As shown in Eq 4.1, the datastore is a collection of key-value pairs where the keys are high-dimensional representations of the input $f_\theta(x_i)$ and the values are the corresponding outputs.
CHAPTER 4. GENERALIZATION THROUGH MEMORIZATION

\[ (\mathcal{K}, \mathcal{V}) = \{(f_\theta(x_i), y_i) \mid (x_i, y_i) \in \mathcal{D}\} \]

Note that we use \( f_\theta(x_i) \) to loosely refer to any representation from within the neural network, including both intermediate as well as the final representations.

This datastore can then be queried during inference, where the query is the corresponding model representation of the test input \( f_\theta(x') \). When querying the datastore, the model retrieves the set of \( k \)-nearest neighbors denoted by \( (k_j, v_j) \in \mathcal{N} \), that is, the top-\( k \) examples in the datastore that are most similar to the test input in the high-dimensional representation space \(-d(k_j, f_\theta(x'))\), based on the chosen distance metric \( d(\cdot, \cdot) \).\(^1\) Then, the retrieved set \( \mathcal{N} \) is used to output a distribution over output classes \( p_{\text{knn}} \). This distribution can either be directly used as the model prediction, or can be interpolated with the model’s output distribution \( p_\theta \):

\[ p(y' \mid x') = \lambda \, p_{\text{knn}}(y' \mid x') + (1 - \lambda) \, p_\theta(y' \mid x') \]

where \( \lambda \) is the interpolation parameter that can either be learned or manually tuned. This provides a general framework within which memorization can be applied to neural models. We provide instantiations of this framework for language modeling and machine translation in subsequent chapters.

A key point to note is that our memorization model uses a pre-trained model and does not incur any additional training costs. This means memorization is not prohibitive—this framework can be applied to any existing model without having to update its parameters. And since the proposed form of memorization is separate from the model parameters, we can control the type and amount of data that is stored in the datastore. This is an extremely useful property for improving model generalization in a range of settings, which we discuss below in the form of key contributions of this part of the dissertation.

**Contributions** The first key contribution of our work is showing that memorization improves model generalization when train and test data come from the same or a very similar distribution. For this, we propose nearest neighbor language models and nearest neighbor machine translation models in Chapters 5 and 6, respectively. We show that saving data, from the same distribution as the model’s training data, into the datastore can surprisingly improve performance even when the underlying base model is a strong baseline, without incurring any additional training costs. In fact, we show that memorization improves generalization in a variety of settings including scaling to larger datasets and facilitating task specialization.

Then we show that memorization also improves out-of-distribution generalization by applying it to the problem of domain adaptation, shown in Chapter 7. Recall that in this setting, the model’s training data and the test data that it is evaluated on come from different domains generated by different underlying distributions. The base model is adapted to other domains by simply memorizing domain-specific data without any in-domain training. This shows that memorization allows a single model to be effective in multiple domains by simply switching out what is stored in the datastore, without applying any parameter updates.

\(^1\)The distance metric is negated since similar examples are closer together and have smaller distances.
A key benefit of memorization is that querying the datastore of examples adds interpretability to model predictions. This is because the prediction from the model can be traced back to the specific examples that were retrieved and thus influenced the output. Looking at the retrieved examples, as in Figure 6.4, could help us understand the model’s predictions and perhaps how we need them to change, especially in cases where models produce biased behavior. Furthermore, control over adding and removing examples from the datastore provides an interesting avenue to explore changes in model behavior in response to changes in the datastore.

**Memorization vs. Scale** Having described the general memorization framework and the ways in which it can improve model generalization, it is important to note that memorization and scaling models by adding trainable parameters are not mutually exclusive approaches. However, hundreds of billion parameter models like GPT-3 (Brown et al., 2020) are still relatively new and their generalization capabilities are still poorly understood. For this reason, we need to empirically investigate and establish a clear relationship between the size of the model and the use of an external non-parametric memory.
Chapter 5

Nearest Neighbor Language Models

In the previous chapter, we motivated and introduced the concept of Generalization through Memorization as a way to improve model performance in a range of scenarios. In this chapter, in order to verify the hypothesis that memorization improves language modeling performance, we propose the Nearest Neighbor Language Model (kNN-LM) which consists of interpolating a pre-trained language model with a nearest neighbor retrieval component, without any added training costs. Recall the key intuition for using this approach lies in being able to identify when two contexts like *Pride and Prejudice* was written by and *The author of Pride and Prejudice* is mean the same, even if you do not know who the author is. Our findings in Chapter 3 suggest that context representations should also be able to capture this notion of similarity across many different types of features, based on both local and distant contexts, and that this property can be used to improve LM performance.

We begin by first formally introducing the model and its various components including the datastore construction and querying. Next, we provide experiments and results that show kNN-LM improves generalization by memorizing the training data, even when the underlying model is a strong baseline. We also provide results that show memorization can scale models to larger datasets containing data from the same distribution as the training set. Next, we discuss how model behavior varies in response to alternate configurations such as using a different intermediate representation from the neural LM, using a different interpolation parameter and retrieving a different number of neighbors. Finally, we provide analyses which confirm our hypothesis that neural representations are crucial to the effectiveness of memorization (see Section 4.2). Qualitatively, we find the model is particularly helpful for long-tail patterns, such as factual knowledge, which might be easier to access via explicit memory.

This work was first presented in the publication *Generalization through Memorization: Nearest Neighbor Language Models* at the International Conference on Learning Representations held in 2020 (Khandelwal et al., 2020).
CHAPTER 5. NEAREST NEIGHBOR LANGUAGE MODELS

5.1 \(k\text{-NN-LM Model}\)

Recall that LMs assign probabilities to sequences. Given a context sequence of tokens \(c_t = (w_1, \ldots, w_{t-1})\), these models estimate \(p(x|c_t)\), the distribution over the target token \(w_t\).

The \(k\text{-NN-LM}\) involves augmenting such a pre-trained LM with a nearest neighbors retrieval mechanism, without any additional training (the representations learned by the LM remain unchanged). This can be done with a single forward pass over a text collection (potentially including the original LM training set), where the resulting context-target pairs are stored in a key-value datastore that is queried during inference, as illustrated in Figure 5.1.

**Datastore** Let \(f(\cdot)\) be the function that maps a context \(c\) to a fixed-length vector representation computed by the pre-trained LM. For instance, in a Transformer LM, \(f(c)\) could map \(c\) to an intermediate representation that is output by an arbitrary self-attention layer. Then, given the \(i\)-th training example \((c_i, w_i)\) \(\in\mathcal{D}\), we define the key-value pair \((k_i, v_i)\), where the key \(k_i\) is the vector representation of the context \(f(c_i)\) and the value \(v_i\) is the target word \(w_i\). The datastore \((\mathcal{K}, \mathcal{V})\) is thus the set of all key-value pairs constructed from all the training examples in \(\mathcal{D}\):

\[
(\mathcal{K}, \mathcal{V}) = \{(f(c_i), w_i) | (c_i, w_i) \in \mathcal{D}\} \tag{5.1}
\]

**Inference** At test time, given the input context \(x\) the model generates the output distribution over next words \(p_{LM}(y|x)\) and the context representation \(f(x)\). The model queries the datastore with \(f(x)\) to retrieve its \(k\)-nearest neighbors \(\mathcal{N}\) according to a distance function \(d(\cdot, \cdot)\) (squared \(L^2\) distance in our experiments, making the similarity function an RBF kernel). Then, it computes a distribution over neighbors based on a softmax of their negative distances, while aggregating probability mass for each vocabulary item across all

---

Figure 5.1: An illustration of \(k\text{-NN-LM}\). A datastore is constructed with an entry for each training set token, and an encoding of its leftward context. For inference, a test context is encoded, and the \(k\) most similar training contexts are retrieved from the datastore, along with the corresponding targets. A distribution over targets is computed based on the distance of the corresponding context from the test context. This distribution is then interpolated with the original model’s output distribution.
its occurrences in the retrieved targets (items that do not appear in the retrieved targets have zero probability):

\[ p_{k\text{NN}}(y|x) \propto \sum_{(k_i, v_i) \in \mathcal{N}} \mathbb{I}_{y=v_i} \exp(-d(k_i, f(x))) \]  

(5.2)

Finally, we follow Grave et al. (2017a) and interpolate the nearest neighbor distribution \( p_{k\text{NN}} \) with the model distribution \( p_{\text{LM}} \) using a tuned parameter \( \lambda \) to produce the final \( k\text{NN-LM} \) distribution:

\[ p(y|x) = \lambda p_{k\text{NN}}(y|x) + (1 - \lambda) p_{\text{LM}}(y|x) \]  

(5.3)

**Implementation** The datastore contains an entry for each target in the training set, which for LMs can be up to billions of examples. To search over this large datastore, we use FAISS (Johnson et al., 2019), an open source library for fast nearest neighbor retrieval in high dimensional spaces. FAISS speeds up search by clustering the keys and looking up neighbors based on the cluster centroids, while reducing memory usage by storing compressed versions of the vectors. We found in preliminary experiments that using \( L^2 \) distance for FAISS retrieval results in better performance for \( k\text{NN-LM} \), compared to inner product distance.

**Related Cache Models** Prior work (Grave et al., 2017c; Merity et al., 2017) used a similar approach to compute similarity to the previous hidden states of test documents, making it easier to copy rare vocabulary items from the recent past. Such techniques have been less popular since the development of Transformers (Vaswani et al., 2017), which can learn to copy recent words using self-attention; in Section 5.3.1, we observe relatively small gains from caching recent items in the same test document à la Grave et al. (2017c). Most relatedly, Grave et al. (2017a) describe an online language model using nearest neighbor search over all previous hidden states, to improve domain adaptation. In our work, we only save training data, with the goal of explicitly memorizing training examples to better generalize to similar cases at test time.

### 5.2 Experimental Setup

**Data** Experiments in this study use the following English corpora:

- **WIKITEXT-103** is a standard benchmark by Merity et al. (2017) for autoregressive language modeling with a 250K word-level vocabulary. It consists of 103M tokens of Wikipedia in the training set and 250K tokens in each of the development and test sets.

- **BOOKS** is the Toronto Books Corpus (Zhu et al., 2015), containing 0.7B. Complete books are held out for validation/test.

- **WIKI-3B** is English Wikipedia, containing about 2.87B tokens. Whole articles are held out for validation/test.

- **WIKI-100M** is a random 100M token subset of WIKI-3B, consisting of complete articles.
Except for WIKITEXT-103, text is tokenized using the byte-pair encoding (Sennrich et al., 2016) with the 29K subword vocabulary from BERT (Devlin et al., 2019).

Model Architecture  $k$NN-LM is compatible with any model that produces fixed size context representations. We use decoder-only Transformers (Vaswani et al., 2017) for language modeling, which are the current state of the art. Since the $k$NN-LM makes no changes to the underlying LM, we take the exact architecture and optimization described by Baevski and Auli (2019) and use it to create a $k$NN-LM for inference. This model consists of 16 layers, each with 16 self-attention heads, 1024 dimensional hidden states, and 4096 dimensional feedforward layers, amounting to 247M trainable parameters. It processes 3072 tokens of context per example for WIKITEXT-103 and 1024 tokens for the rest of the corpora. Following Baevski and Auli (2019), we use adaptive inputs and an adaptive softmax (Grave et al., 2017b) with tied weights (Press and Wolf, 2017) for the WIKITEXT-103 experiments. On other datasets we do not use adaptive inputs or an adaptive softmax.

Evaluation  LMs are trained to minimize the negative log-likelihood of the training corpus, and evaluated by perplexity (exponentiated negative log-likelihood) on held out data. Following Baevski and Auli (2019), 512 tokens are scored per test example, but up to 2560 tokens of extra prior context is provided for WIKITEXT-103 and up to 512 tokens of extra prior context is provided for the rest of the corpora.

$k$NN-LM  The keys used for $k$NN-LM are the 1024-dimensional representations fed to the feedforward network in the final layer of the Transformer LM (after self-attention and layernorm; see Section 5.4 for further explanation). We perform a single forward pass over the training set with the trained model, in order to save the keys and values. During this forward pass, each target token is provided a minimum of 1536 tokens of prior context for WIKITEXT-103 and a minimum of 512 tokens for the rest of the corpora. A FAISS index is then created using 1M randomly sampled keys to learn 4096 cluster centroids. For efficiency, keys are quantized to 64-bytes. During inference, we retrieve $k = 1024$ neighbors, and the index looks up 32 cluster centroids while searching for the nearest neighbors. For WIKITEXT-103 experiments, we compute squared $L^2$ distances with full precision keys, but for the other datasets we use the FAISS $L^2$ distances (not squared) between quantized keys directly, for faster evaluation. We tune the interpolation parameter $\lambda$ on the validation set.\footnote{Code is available at: \url{https://github.com/urvashik/knnlm}}

Computational Cost  Although the $k$NN-LM requires no training given an existing LM, it does add some other computational overheads. Storing the keys and values requires a single forward pass over the training set, which amounts to a fraction of the cost of training for one epoch on the same examples. Once the keys are saved, for WIKITEXT-103 building the cache with 103M entries takes roughly two hours on a single CPU. Finally, running on the validation set took approximately 25 minutes when retrieving 1024 keys. While the
Table 5.1: Performance on WIKITEXT-103. The $k$NN-LM substantially outperforms existing work. Gains are additive with the related but orthogonal continuous cache, allowing us to improve the base model by almost 3 perplexity points with no additional training. We report the median of three random seeds.

Table 5.2: Performance on BOOKS, showing that $k$NN-LM works well in multiple domains.

5.3 Experiments

5.3.1 Using the Training Data as the Datastore

We first experiment with creating a datastore from the same data used to train the LM. Table 5.1 shows that $k$NN-LM improves perplexity on WIKITEXT-103 from 18.65 (Baevski and Auli, 2019) to a new state-of-the-art of 16.12. We also provide reported perplexities from two other recent models that also build upon Baevski and Auli’s, suggesting that further improvements may be possible by augmenting the $k$NN-LM with these techniques. We compare with models trained only on the standard training set, but recent work has shown performance can be improved by training on additional data, from either the test set (Krause et al., 2019) or large amounts of web text (Shoeybi et al., 2019).

We also experiment with a continuous cache model, a related but orthogonal technique from Grave et al. (2017c), in which the model saves and retrieves neighbors from earlier in the test document, rather than the training set. Gains from interpolating with the continuous cache are smaller than reported in the original setting that used LSTMs, perhaps because self-attentive language models can learn to perform such queries.

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity $(\downarrow)$</th>
<th># Trainable Params</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dev</td>
<td>Test</td>
</tr>
<tr>
<td>Baevski and Auli (2019)</td>
<td>17.96</td>
<td>18.65</td>
</tr>
<tr>
<td>+Transformer-XL (Dai et al., 2019)</td>
<td>-</td>
<td>18.30</td>
</tr>
<tr>
<td>+Phrase Induction (Luo et al., 2019)</td>
<td>-</td>
<td>17.40</td>
</tr>
<tr>
<td>Base LM (Baevski and Auli, 2019)</td>
<td>17.96</td>
<td>18.65</td>
</tr>
<tr>
<td>+$k$NN-LM</td>
<td><strong>16.06</strong></td>
<td><strong>16.12</strong></td>
</tr>
<tr>
<td>+Continuous Cache (Grave et al., 2017c)</td>
<td>17.67</td>
<td>18.27</td>
</tr>
<tr>
<td>+$k$NN-LM + Continuous Cache</td>
<td><strong>15.81</strong></td>
<td><strong>15.79</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity $(\downarrow)$</th>
<th># Trainable Params</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dev</td>
<td>Test</td>
</tr>
<tr>
<td>Base LM (Baevski and Auli, 2019)</td>
<td>14.75</td>
<td>11.89</td>
</tr>
<tr>
<td>+$k$NN-LM</td>
<td><strong>14.20</strong></td>
<td><strong>10.89</strong></td>
</tr>
</tbody>
</table>
Table 5.3: Experimental results on WIKI-3B. The model trained on 100M tokens is augmented with a datastore that contains about 3B training examples, outperforming the vanilla LM trained on the entire WIKI-3B training set.

Improvements from the continuous cache are additive with the $k$NN-LM, pushing our state-of-the-art result to 15.79, a gain of 2.86 over the base model.

Finally, we repeat the experiment using text from a different domain, BOOKS, to control for the possibility that encyclopedic Wikipedia text is somehow uniquely good for caching. Table 5.2 shows an improvement in test set perplexity from 11.89 to 10.89, suggesting that this is not the case.

### More Data without Training

Section 5.3.1 has shown that retrieving neighbors from the training data can significantly improve language modeling performance. This raises the question: can retrieving nearest neighbors from data be a substitute for training on it? To test this, we train a LM on WIKI-100M and use it to build a datastore from WIKI-3B, a corpus 30 times larger than the training set. We then compare this $k$NN-LM to a vanilla LM trained on the entire WIKI-3B corpus. The original LM (Baevski and Auli, 2019) was trained for 286K steps on a corpus of similar size to WIKI-100M. In this experiment, we control for the number of trainable parameters, so both models have the same capacity. When scaling up to WIKI-3B, we tuned only the number of updates on the validation set and found that training for 572K steps (double) produces a slightly stronger baseline.

Table 5.3 shows that, as expected, the model trained on 3B tokens dramatically outperforms the model trained on 100M tokens, improving perplexity from 19.59 to 15.17. However, adding nearest neighbors retrieval over those 3B examples to the model trained on 100M tokens improves perplexity from 19.59 to 13.73; i.e. retrieving nearest neighbors from the corpus outperforms training on it. This result suggests that rather than training language models on ever larger datasets, we can use smaller datasets to learn representations and augment them with $k$NN-LM over a large corpus.

To understand how the amount of data used for $k$NN retrieval affects performance, we use the WIKI-100M model to create datastores using different amounts of randomly sampled data from WIKI-3B. Figure 5.2a shows that using only 1.6B examples for the datastore already surpasses the performance of the model trained on all of WIKI-3B. In addition, performance does not saturate at 3B examples in the datastore, suggesting that growing the datastore more could lead to further gains. Figure 5.2b shows the model relies more on the $k$NN component as the size of the datastore increases.
5.4 Tuning Nearest Neighbor Search

While the \( k \)NN-LM is conceptually straightforward, and requires no additional training, a number of hyperparameters are introduced for nearest neighbor search. We experiment with different choices here.

**Key Function** For similarity search, we extract a representation of context \( c \) using an intermediate state of the LM \( f(c) \). Transformers compute a number of different intermediate states, and we compare several choices depicted in Figure 5.3, with results shown in Table 5.4. While all the instantiations of \( f \) we tried are helpful, we achieved the largest improvement by using the input to the final layer’s feedforward network. We also observe that normalized representations (i.e. taken immediately after the layer norm) perform better. Repeating the experiment on the second-last transformer layer showed similar trends with slightly worse results (not shown), suggesting that the feedforward layer might be focusing more on the prediction problem, while the onus of representing the input falls more on the self-attention layer.

**Number of Neighbors per Query** Each query returns the top-\( k \) neighbors. Figure 5.4 shows that performance monotonically improves as more neighbors are returned, and suggests that even larger improvements may be possible with a higher value of \( k \). Nonetheless, even a small number of neighbors (\( k = 8 \)) is enough to achieve a new state of the art.

**Interpolation Parameter** We use a parameter \( \lambda \) to interpolate between the base model distribution and the distribution from \( k \)NN search over the dataset. This parameter is manually tuned and set to a fixed value for all examples. \( \lambda = 0.25 \) is optimal on \textsc{WikiText-103}. As shown in Figure 5.2b, as the amount of data not
Figure 5.3: Transformer LM layer.

Table 5.4: \textsc{Wikitext-103} validation results using different states from the final layer of the LM as the representation function \(f(\cdot)\) for keys and queries. We retrieve \(k=1024\) neighbors and \(\lambda\) is tuned for each.

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Dev ppl. ((\downarrow))</th>
</tr>
</thead>
<tbody>
<tr>
<td>No datastore</td>
<td>17.96</td>
</tr>
<tr>
<td>Model output</td>
<td>17.07</td>
</tr>
<tr>
<td>Model output layer normalized</td>
<td>17.01</td>
</tr>
<tr>
<td>FFN input after layer norm</td>
<td>\textbf{16.06}</td>
</tr>
<tr>
<td>FFN input before layer norm</td>
<td>17.06</td>
</tr>
<tr>
<td>MHSA input after layer norm</td>
<td>16.76</td>
</tr>
<tr>
<td>MHSA input before layer norm</td>
<td>17.14</td>
</tr>
</tbody>
</table>

Figure 5.4: Effect of the number of nearest neighbors returned per word on \textsc{Wikitext-103} (validation set). Returning more entries from the datastore monotonically improves performance.

contained in the model’s training set grows, so does the model’s dependence on the retrieval. See Figure 7.1 for a trend of how performance on \textsc{Books} varies with \(\lambda\).

**Precision of Similarity Function** In \textsc{FAISS}, the nearest neighbor search computes \(L^2\) distances against quantized keys. We found results were improved from 16.5 perplexity on \textsc{Wikitext-103} to 16.06 by computing squared \(L^2\) distances with full precision keys for Equation 5.2.

### 5.5 Analysis

**Qualitative Analysis** To understand why \(k\text{NN-LM}\) improves performance, we manually examine cases in which \(p_{k\text{NN}}\) was significantly better than \(p_{\text{LM}}\). Figures 5.7 and 5.8 show such examples. Figures 5.7 shows an
CHAPTER 5. NEAREST NEIGHBOR LANGUAGE MODELS

Figure 5.5: Interpolating the Transformer LM with \( n \)-gram LMs on WIKITEXT-103 (validation set). Using \( k \)NN-LM gives a much lower perplexity, suggesting that the representations are learning more than just matching local context.

Figure 5.6: Training curves for the Transformer LM with and without dropout. Turning off dropout allows the training loss to go to 0, indicating that the model has sufficient capacity to memorize the training data.

interesting case where the model matches the trigram impact on the in several retrieved neighbors, but puts almost all weight on the most relevant neighbor, thus adding more value than an \( n \)-gram LM.

In general, we find that examples where \( k \)NN-LM is most helpful typically contain rare patterns. Examples include factual knowledge, names, and near-duplicate sentences from the training set. In these cases, assigning train and test instances similar representations (via \( f(\cdot) \)) appears to be an easier problem than implicitly memorizing the next word in model parameters.

**Simple vs Neural Representation** We observe that many long-tail phenomena manifest as rare \( n \)-grams (e.g. names). Is it therefore possible to interpolate an \( n \)-gram model with a Transformer LM, as an alternative to our \( k \)NN approach? Figure 5.5 shows little improvement from using \( n \)-gram LMs – 0.2 perplexity points (similarly to Bakhtin et al. (2018)). This result highlights the need to use the learned representation function \( f(\cdot) \) to measure similarity between more varied contexts.

**Implicit vs Explicit Memory** If a neural representation function is crucial for \( k \)NN-LM, could implicitly memorizing the training dataset in the neural network parameters replace the explicit memory in the datastore? To test this, we train a Transformer LM with no dropout. Figure 5.6 shows that this model eventually reaches zero training loss, indicating that it can make perfect predictions for all examples in the training set; the model has memorized the dataset. Naturally, the memorizing LM overfits, i.e. the training loss drops to 0 while the best validation perplexity is much higher at 28.59. For comparison, the vanilla Transformer LM (with dropout) has a much higher training loss (shown in Figure 5.6), but also generalizes better with a validation perplexity of 17.96. This result shows that the Transformer has sufficient capacity to memorize the training set.

We consider whether the memorizing LM can be an effective substitute for nearest neighbor search.
Test Context \( (p_{kNN} = 0.998, p_{LM} = 0.124) \)  

Test Target

<table>
<thead>
<tr>
<th>Training Set Context</th>
<th>Training Set Target</th>
<th>Context Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>it was organised by New Zealand international player Joseph Warbrick, promoted by civil servant Thomas Eyton, and managed by James Scott, a publican. The Natives were the first New Zealand team to perform a haka, and also the first to wear all black. They played 107 rugby matches during the tour, as well as a small number of Victorian Rules football and association football matches in Australia. Having made a significant impact on the...</td>
<td>development</td>
<td>0.998</td>
</tr>
<tr>
<td>promoted to a new first grade competition which started in 1900. Glebe immediately made a big impact on the...</td>
<td>development</td>
<td>0.00012</td>
</tr>
<tr>
<td>centuries, few were as large as other players managed. However, others contend that his impact on the...</td>
<td>game</td>
<td>0.000034</td>
</tr>
<tr>
<td>Nearly every game in the main series has either an anime or manga adaptation, or both. The series has had a significant impact on the...</td>
<td>development</td>
<td>0.00000092</td>
</tr>
</tbody>
</table>

Figure 5.7: Example where the \( k \)NN model has much higher confidence in the correct target than the LM. Although there are other training set examples with similar local \( n \)-gram matches, the nearest neighbour search is highly confident of specific and very relevant context.

Interpolating the memorizing LM with the original LM improves validation perplexity by just 0.1 – compared to 1.9 from \( k \)NN-LM. This result suggests that although the Transformer is expressive enough to memorize all training examples, learning to do so does not result in context representations that generalize. In contrast, \( k \)NN-LM memorizes training data while improving generalization.

From these experiments, we conjecture that \( k \)NN-LM improves performance because (1) the Transformer LM is very good at learning a representation function for contexts with an implicit notion of similarity, and (2) while the Transformer has capacity to memorize all training examples, doing so causes its representation to generalize less effectively, but (3) the \( k \)NN-LM allows the model to memorize the training data while retaining an effective similarity function.

### 5.6 Related Work

We discuss related uses of caches for language modeling in Section 6.1.

Similar \( k \)NN models to ours have been proposed for computer vision tasks (Papernot and McDaniel, 2018; Orhan, 2018; Zhao and Cho, 2018), primarily motivated by improving interpretability and robustness to adversarial attacks. We hypothesize that our method may be particularly effective for language modeling, because plentiful unlabeled data allows datastores of billions of tokens, and language modeling often requires world knowledge to be learnt from few examples.
CHAPTER 5. NEAREST NEIGHBOR LANGUAGE MODELS

Test Context \((p_{\text{ANN}} = 0.950, p_{\text{LM}} = 0.503)\)  

U2 do what they’re best at, slipping into epic rock mode, playing music made for the arena”. In two other local newspaper reviews, critics praised the song's inclusion in a sequence of greatest hits. For the PopMart Tour of 1997–...

<table>
<thead>
<tr>
<th>Training Set Context</th>
<th>Training Set Target</th>
<th>Context Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Following their original intent, “Sunday Bloody Sunday” was not played during any of the forty-seven shows on the Lovetown Tour in 1989. The song reappeared for a brief period during the Zoo TV Tour, and late during the second half of PopMart Tour (1997–…).</td>
<td>1998</td>
<td>0.936</td>
</tr>
<tr>
<td>They are 6 times Champions and they won the Challenge Cup in 1938, and have experienced two previous stretches in the Super League, 1997–…</td>
<td>2002</td>
<td>0.0071</td>
</tr>
<tr>
<td>About $40 million ($61.4 million in 2018 dollars) was spent on the property acquisition. After weather-related construction delays due to the El Nino season of the winter of 1997–…</td>
<td>1998</td>
<td>0.0015</td>
</tr>
<tr>
<td>This made it the highest-rated season of The X-Files to air as well as the highest rated Fox program for the 1997–…</td>
<td>98</td>
<td>0.00000048</td>
</tr>
</tbody>
</table>

Figure 5.8: In this example, the desired date pattern appears in many examples. Yet, the nearest neighbors search is able to identify the only training set context which is relevant to the test context and assigns it the highest probability mass.

As discussed in Section 4.2, nearest neighbor models have been applied to a number of NLP problems in the past, such as part of speech tagging (Daelemans et al., 1996) and morphological analysis (van den Bosch et al., 2007a), but the use of learned representations makes the similarity function much more effective in the case of neural models. More recently, Kaiser et al. (2017) have used a similarly differentiable memory that is learned and updated during training, and is applied to one-shot learning tasks.

Several models have also improved language generation by using training examples directly at test time. Guu et al. (2018) propose a model that samples training sentences at random and edits them with a sequence-to-sequence model, but does not use a retrieval mechanism such as \(k\)NN. Gu et al. (2018) introduce a translation model that attends over retrieved training set examples. Weston et al. (2018) improve a dialogue response generation model by refining similar instances from the training set. \(k\)NN-LM differs from these approaches by working at the level of individual tokens instead of whole training sentences, as well as not incorporating the retrieval mechanism into the training pipeline.

A general trend in machine learning, and in language modeling in particular, is that adding more data consistently improves performance (Devlin et al., 2019; Radford et al., 2019; Yang et al., 2019; Liu et al., 2019; Zellers et al., 2019; Shoeybi et al., 2019). Our work offers an alternative method for scaling language models, in which relatively small models learn context representations, and a nearest neighbour search acts as a highly expressive classifier.
5.7 Conclusion

In this chapter, we have introduced $k$NN-LMs, which can significantly outperform standard language models by employing the ideas of memorization and directly querying examples from an external datastore at test time. The approach can be applied to any neural language model without adding training costs and it can also scale the LM to larger text collections. The success of this method suggests that learning similarity functions between contexts may be an easier problem than predicting the next word from some given context. Qualitative analysis suggests that this method is particularly effective for rare patterns such as factual knowledge or named entities.

Given the non-parametric nature of memorization, we have shown that it can scale to very large datastore sizes with relative ease. However, there exists a trade-off between growing the size of the memory to gain access to more relevant contexts and the retrieval speed which can in turn reduce inference speeds. Beyond simply tuning the retrieval hyperparameters which can address some of the latency issues, understanding which examples memorization is helpful for can be extremely useful for removing unnecessary contexts to pare down the datastore. For instance, currently our datastores contain all context-target pairs from any given text collection including frequent words. It may not be necessary to include all of these examples in the datastore. However, naively subsampling examples that contain frequent words as the target, without adjusting the interpolation parameter, causes perplexity to deteriorate. This shows that we need more careful consideration in first understanding when memorization is helpful and then applying this knowledge to better take advantage of the method as well as improve its speed during inference.

Beyond in-distribution performance, we are interested in understanding whether memorization can improve out-of-distribution generalization as well. We investigate this in Chapter 7 by applying memorization to the problem of domain adaptation, and show that $k$NN-LM allows an existing pre-trained model to adapt to new domains without training on data from the domain in which it is being evaluated. Furthermore, we are also interested in understanding whether memorization is a general approach that improves performance beyond density estimation. We investigate this question in the next chapter where we apply this approach to neural machine translation.
Chapter 6

Nearest Neighbor Machine Translation

Having demonstrated the effectiveness of memorization for neural language models, in this chapter, we argue that it is a general approach that is extremely effective beyond the problem of density estimation. In order to verify its generality, we show that memorization can improve language generation performance by applying it to neural machine translation. Thus, inspired by $k$NN-LM, we introduce Nearest Neighbor Machine Translation ($k$NN-MT) which, similar to the LM setting, involves interpolating a pre-trained neural machine translation (MT) model with a nearest neighbor retrieval component, without any further training. This setting is far more complex than LMs because now, apart from conditioning on the prior context in the target language that the model is generating, it must also condition on the input sequence in the source language that the model is translating from. But even so, the representations of the model are sophisticated and are able to encode a notion of similarity that captures both.

In the rest of this chapter, we first formally define the $k$NN-MT model, how its datastore is created and queried, as well as how it differs from $k$NN-LM, since the two models are very related. Then, we provide experiments and results that show $k$NN-MT is extremely effective at improving translation performance when the model’s training data is memorized in the datastore, even when the underlying model is an extremely competitive baseline and its parameters are fixed. We also show that $k$NN-MT can be useful for task specialization of massive multilingual translation models such that the multilingual system adapts to the specific pair of languages invoked by the test sample. Through creative experiment design, we show that not only can $k$NN-MT improve generalization by specializing these models at test time, but it can also help model performance by using cross-lingual retrieval to draw neighbors from related high-resource language pairs. Finally, we note that memorization can make generation models more interpretable because we can inspect the examples being retrieved from the datastore.

This work was first presented in the publication *Nearest Neighbor Machine Translation* at the International Conference on Learning Representations held in 2021 (Khandelwal et al., 2021).
6.1 \(k\)NN-MT Model

\(k\)NN-MT involves augmenting the decoder of a pre-trained machine translation model with a nearest neighbor retrieval mechanism, allowing the model direct access to a datastore of cached examples. The translation is generated word-by-word; at each time step, we find the most similar contexts in the datastore, and compute a distribution over the corresponding target tokens, as shown in Figure 6.1. This distribution is then interpolated with the output distribution from the pre-trained MT model.

More specifically, given an input sequence of tokens in a source language \(s = (s_1, \ldots, s_{M_s})\), a neural MT model outputs a sequence of tokens \(t = (t_1, \ldots, t_{M_t})\) in the target language. When using autoregressive decoders, the output distribution for each token \(t_i\) in the target sequence is conditioned on the entire source sequence as well as the previous target tokens, \(p(t_i | s, t_{1:i-1})\). Let \((s, t_{1:i-1})\) be the translation context and \(t_i\) be the target token.

**Dataverse creation** Our datastore is constructed offline and consists of a set of key-value pairs. The key is a high-dimensional representation of the entire translation context computed by the MT decoder, \(f(s, t_{1:i-1})\), where \(f\) represents a mapping from input to an intermediate representation of the decoder. The value is the corresponding ground truth target token \(t_i\). For a parallel text collection \((\mathcal{S}, \mathcal{T})\), the representations are generated by a single forward pass over each example and the complete datastore is defined as follows:

\[
(\mathcal{K}, \mathcal{V}) = \{(f(s, t_{1:i-1}), t_i), \ \forall t_i \in t \mid (s, t) \in (\mathcal{S}, \mathcal{T})\}
\]  

Tokens from the source language are not stored directly as values in the datastore. Conditioning on the source is implicit via the keys, and the values are only target language tokens.
CHAPTER 6. NEAREST NEIGHBOR MACHINE TRANSLATION

Generation At test time, given a source \( x \), the model outputs a distribution over the vocabulary \( p_{MT}(y_i|x, \hat{y}_{1:i-1}) \) for the target \( y_i \) at every step of generation, where \( \hat{y} \) represents the generated tokens. The model also outputs the representation \( f(x, \hat{y}_{1:i-1}) \), which is used to query the datastore for the \( k \) nearest neighbors \( N \) according to squared-\( L^2 \) distance, \( d \). In practice, the search over billions of key-value pairs is carried out using FAISS (Johnson et al., 2019), a library for fast nearest neighbor search in high-dimensional spaces.

The retrieved set is converted into a distribution over the vocabulary by applying a softmax with temperature \( T \) to the negative distances and aggregating over multiple occurrences of the same vocabulary item. Using a temperature greater than one flattens the distribution, and prevents overfitting to the most similar retrievals.

\[
p_{kNN}(y_i|x, \hat{y}_{1:i-1}) \propto \sum_{(k_j, v_j) \in N} \mathbb{1}_{y_i = v_j} \exp \left( -\frac{-d(k_j, f(x, \hat{y}_{1:i-1}))}{T} \right) \tag{6.2}
\]

While a pure \( k \)NN approach is effective, we improve results by interpolating with the base model distribution, which is more robust in cases without relevant cached examples. The model and \( k \)NN distributions are interpolated with a tuned parameter \( \lambda \), resulting in the final \( kNN-MT \) distribution:

\[
p(y_i|x, \hat{y}_{1:i-1}) = \lambda \ p_{kNN}(y_i|x, \hat{y}_{1:i-1}) + (1 - \lambda) \ p_{MT}(y_i|x, \hat{y}_{1:i-1}) \tag{6.3}
\]

The complete translation is generated using beam search.

\( kNN-MT \) vs. \( kNN-LM \) \( kNN-MT \) is a generalization of \( kNN-LM \) applied to conditional sequence generation, with a few important differences. First, the keys are not only conditioned on prior context, but also on the source sequence (here, in a different language). This means that the representations must encode both source and target context; we show examples in Section 6.6. Second, there is an additional tuned parameter, the softmax temperature. Higher temperatures flatten the distribution and allow for greater diversity without overfitting to the retrieved contexts, as shown in Section 6.6.

6.2 Experimental Setup

In this chapter, we present experiments with \( kNN-MT \) in two settings: (1) single language-pair translation, and (2) multilingual MT.

Data We use the following datasets for training and evaluation.

- WMT’19: For the single language-pair experiments, we use WMT’19 data for German-English.
- CCMATRIX: We train our multilingual model on CCMatrix (Schwenk et al., 2019), containing parallel data for 79 languages and 1,546 language pairs. The parallel sentences are mined from cleaned monolingual commoncrawl data created using the ccNet pipeline (Wenzek et al., 2020).
similar sentences in different languages are aligned using a learned distance measure; we use examples where the distance measure is at least 1.06, resulting in 4 billion sentence-pairs.

- **NEWSTEST**: The newstest2018 and newstest2019 test sets from WMT (Bojar et al., 2018; Barrault et al., 2019) are used as validation and test sets for the multilingual experiments. The same German-English validation and test sets are also used for evaluation in the single language-pair experiments.

- **TED TALKS**: We use the Ted Talks data prepared by Qi et al. (2018) for evaluation in the multilingual setting, particularly to explore performance for language pairs that do not include English.

**Models** For the single language-pair experiments, we use the WMT’19 German-English news translation task winner (Ng et al., 2019), available via the FAIRSEQ library (Ott et al., 2019).¹ It is a Transformer encoder-decoder model (Vaswani et al., 2017) with 6 layers, 1,024 dimensional representations, 8,192 dimensional feedforward layers and 8 attention heads. Apart from WMT’19 training data, this model is trained on over 10 billion tokens of backtranslation data and fine-tuned on newstest test sets from years prior to 2018. In this work, we do not use ensembles or n-best reranking.

For multilingual MT, we trained a 418M parameter Transformer-based encoder-decoder model on the CCMatrix data for 100K updates. The model has embedding dimension 1,024, hidden dimension 4,096, 12 layers in both the encoder and decoder, with 16 attention heads. To balance the training of different language pairs, which have various resource levels, we apply temperature upsampling with $T = 5$ (Arivazhagan et al., 2019). The vocabulary is shared across all languages and consists of 128K subwords extracted using sentencepiece (Kudo and Richardson, 2018).² All results use case-sensitive detokenized BLEU, measured using SACREBLEU (Post, 2018).

$\textit{kNN-MT}$ In this work, we use a FAISS index to represent the datastore and search for nearest neighbors. The keys are stored in clusters to speed up search and quantized to 64-bytes for space efficiency (the full-precision keys are discarded). The index is constructed offline via a single forward pass over every example in the given parallel text collection. We use the 1024-dimensional representation input to the final layer feedforward network as the key. Building the index involves a training phase to learn the cluster centroids. We use 5M keys for learning 131K cluster centroids for the multilingual experiments. During inference, we query the datastore for 64 neighbors while searching 32 clusters. The interpolation and softmax temperature parameters are tuned on the validation sets.

**Computational Cost** While $k$NN-MT does not add trainable model parameters, it does add some computational overhead. The primary cost of building the datastore is a single forward pass over all examples in the datastore, which is a fraction of the cost for training on the same examples for one epoch. During inference,

¹[https://github.com/pytorch/fairseq/tree/master/examples/translation](https://github.com/pytorch/fairseq/tree/master/examples/translation)
²Evaluating our model on the recently released OPUS100 (Zhang et al., 2020) corpus improves upon the result in Zhang et al. (2020) by 0.4 BLEU, suggesting that it is a very strong baseline.
retrieving 64 keys from a datastore containing billions of items results in a generation speed that is two orders of magnitude slower than the base MT system. Generation speed can be improved by searching fewer clusters, using smaller beams, or querying smaller datastores, with relatively minor trade-offs in performance, as we will see in Section 6.4. Developing faster nearest neighbor search tools remains an active area of research (Guo et al., 2020).

6.3 Experiments

6.3.1 Single Language-Pair Translation

To test whether $k$NN-MT can improve a model’s ability to generalize from its training data, we first apply it to a state-of-the-art translation model, using a datastore containing only the original training set. We use a state-of-the-art German-English model as our base MT system, which scores 37.59 BLEU on the newstest2019 test set. This is a highly competitive baseline – apart from the WMT’19 training data, the base model has also been trained on over 10 billion tokens of extra backtranslation data as well as fine-tuned on newstest test sets from previous years. Providing this heavily tuned base model with a datastore containing about 770M tokens of WMT’19 training data improves performance by 1.5 BLEU to 39.08, without any additional training. This result shows that even very strong translation models can be improved with test-time access to training sets.

6.3.2 Task Specialization for Multilingual Machine Translation

Next, we apply $k$NN-MT to multilingual machine translation. A multilingual translation model is trained to translate to and from many different language-pairs. However, when translating a given test instance, the model is typically processing just a single language-pair. Our hypothesis is that $k$NN-MT allows the multilingual model to specialize to the given language pair by retrieving nearest neighbors from language-specific datastores. This can be seen as a form of task specialization where translating between different language pairs can be viewed as separate tasks—a model that is trained to perform many different tasks simultaneously can specialize to a single task during inference, in our case, without updating model parameters. For the following experiments, we create language-specific datastores using subsets of the CCMatrix parallel data that the model has been trained on.

Retrieving neighbors from same source language data Here, we build one datastore per language-pair being tested, using the training examples for that language-pair. Table 6.1 shows performance for the baseline and $k$NN-MT on 17 language-pairs from newstest2019. Retrieving neighbors results in up to 3 BLEU improvements for English-German, English-Chinese and Chinese-English, with an average improvement of 1.4 BLEU across all 17 pairs, without any additional training.

3The winning system (scoring 40.8 BLEU) extends this model with ensembling and $n$-best reranking.
CHAPTER 6. NEAREST NEIGHBOR MACHINE TRANSLATION

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2,000</td>
<td>2,000</td>
<td>2,000</td>
<td>993</td>
<td>1,996</td>
<td>1,000</td>
<td>1,701</td>
<td>1,997</td>
<td>2,000</td>
</tr>
<tr>
<td>Base MT</td>
<td>34.45</td>
<td>36.42</td>
<td>24.23</td>
<td>12.79</td>
<td>25.92</td>
<td>29.59</td>
<td>32.75</td>
<td>21.15</td>
<td>22.78</td>
</tr>
<tr>
<td>+kNN-MT</td>
<td><strong>35.74</strong></td>
<td><strong>37.83</strong></td>
<td><strong>27.51</strong></td>
<td>13.14</td>
<td>26.55</td>
<td>29.98</td>
<td><strong>33.68</strong></td>
<td>21.62</td>
<td><strong>23.76</strong></td>
</tr>
<tr>
<td>Datastore Size</td>
<td>5.56B</td>
<td>3.80B</td>
<td>1.19B</td>
<td>360M</td>
<td>318M</td>
<td>168M</td>
<td>696M</td>
<td>533M</td>
<td></td>
</tr>
<tr>
<td>Test set sizes</td>
<td>en-de</td>
<td>en-ru</td>
<td>en-zh</td>
<td>en-ja</td>
<td>en-fi</td>
<td>en-lt</td>
<td>fr-de</td>
<td>cs-de</td>
<td>Avg.</td>
</tr>
<tr>
<td></td>
<td>1,997</td>
<td>1,997</td>
<td>1,997</td>
<td>1,000</td>
<td>1,997</td>
<td>998</td>
<td>1,701</td>
<td>1,997</td>
<td>-</td>
</tr>
<tr>
<td>+kNN-MT</td>
<td><strong>39.49</strong></td>
<td><strong>27.91</strong></td>
<td><strong>33.63</strong></td>
<td><strong>23.23</strong></td>
<td>22.20</td>
<td>18.25</td>
<td><strong>27.81</strong></td>
<td>23.55</td>
<td><strong>27.40</strong></td>
</tr>
<tr>
<td>Datastore Size</td>
<td>6.50B</td>
<td>4.23B</td>
<td>1.13B</td>
<td>433M</td>
<td>375M</td>
<td>204M</td>
<td>3.98B</td>
<td>689M</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.1: Multilingual machine translation with \( k \)NN-MT. All test sets are from newstest2019, except \( ja-en/-en-ja \) which are from newstest2020. Adding \( k \)NN-MT increases BLEU scores in all cases, and by over 3 points for \( en-de \), \( zh-en \) and \( en-zh \). Bold scores indicate significant results based on statistically powered experiments (Card et al., 2020).

Table 6.1 also shows the sizes of each of the datastores. Datastore size and the increase in BLEU are only weakly correlated across languages, though within a language, a larger datastore is decidedly better, as shown in Section 6.4. This suggests that underlying factors, such as the quality of the parallel data used to populate the datastore, may also factor into the size of the improvements from \( k \)NN-MT.

Finally, we also observe that improvements for languages translated into English, on average 1.23 BLEU, are lower than improvements for languages translated from English, on average 1.94 BLEU. As English is the most frequent language in the base model’s training data, this suggests that \( k \)NN-MT is particularly useful for improving decoders in languages that may be underfit during training.

**Retrieving neighbors using English as the source language**  Here, we construct datastores from training examples where English is the source language, and the target language is the language being tested. This setting is useful for rarer language pairs with less bi-text, and is related to pivoting (Utiyama and Isahara, 2007; Cohn and Lapata, 2007). Table 6.2 shows that on five pairs from the Ted Talks data and three from newstest2019, we find that \( k \)NN-MT improves performance by 1 BLEU on average. This result shows that the model’s representations of the source generalize well enough across languages to make cross-lingual retrieval effective. Further investigation is needed to study the extent to which multilingual representations from related and unrelated languages can improve translation performance via \( k \)NN-MT.

For Table 6.1 and other experiments, we follow advice from Card et al. (2020) regarding the statistical power of machine translation experiments given the improvements in BLEU scores and the size of the dataset. The authors present analyses on investigating the statistical power for a single language pair using a collection of models, and provide trends for how power of a translation experiment varies as the difference in BLEU and the size of the dataset change, given assumed values of two parameters. We verify that their assumptions
for the two parameters hold for a couple of other language pairs. Specifically, we find that for Chinese-English $P_0 = 0.13$ and $b_0 = 12$, and for English-Chinese $P_0 = 0.07$ and $b_0 = 16$. This indicates that these experiments, with the test sets containing about 2,000 examples, have close to 100% power which was verified using the notebooks provided by Card et al. (2020). We refer the reader to the original paper for more details. More generally, experiments on datasets which contain about 2,000 examples, with improvements of about 1 BLEU or higher, are statistically powered.

### 6.4 Tuning kNN-MT

We investigate how key hyperparameters affect multilingual kNN-MT on validation data.

**Softmax temperature**  A softmax temperature is used when estimating the nearest neighbor distribution in order to prevent the model from assigning most of the probability mass to a single neighbor, thus hurting diversity. Values greater than 1 will flatten the distribution, which can improve kNN-MT performance. Figure 6.2 shows that a temperature of 1 results in significantly lower BLEU scores. For all of our experiments, values of either 10 or 100 prove to be optimal.

**Number of neighbors per query**  In our experiments, we fix the value of $k$, the number of neighbors retrieved per query, to 64. For a fixed temperature and interpolation parameter, we find that performance does not improve when retrieving a larger number of neighbors, and in some cases, performance deteriorates. This suggests that retrieving more neighbors can add noise to the sequence generation process. Figure 6.2 shows that in some cases, performance improves when retrieving fewer neighbors, and further gains may be possible by tuning this parameter.

**Datastore size**  Figure 6.3 shows increasing the size of the datastore improves translation performance. However, larger datastores result in slower retrieval, indicating a speed-performance trade-off. Much of the benefit can be realized with much smaller, and correspondingly faster, datastores.

<table>
<thead>
<tr>
<th>Test set sizes</th>
<th>Ted Talks</th>
<th>Newstest2019</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>de-ja</td>
<td>4,442</td>
<td>1,701</td>
<td>16.98</td>
</tr>
<tr>
<td>ru-ja</td>
<td>5,090</td>
<td>22.78</td>
<td></td>
</tr>
<tr>
<td>uk-ja</td>
<td>3,560</td>
<td>21.15</td>
<td></td>
</tr>
<tr>
<td>de-ru</td>
<td>4,288</td>
<td>26.04</td>
<td></td>
</tr>
<tr>
<td>de-zh</td>
<td>4,349</td>
<td>21.22</td>
<td></td>
</tr>
<tr>
<td>fr-de</td>
<td>1,997</td>
<td>21.74</td>
<td></td>
</tr>
<tr>
<td>cs-de</td>
<td>1,997</td>
<td>23.71</td>
<td></td>
</tr>
<tr>
<td>de-cs</td>
<td>1,997</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Adding datastores with English source-side data can improve translation from other languages by an average of 1 BLEU, suggesting that our representations generalize over different source languages. The model’s representations of the source generalize across languages and make cross-lingual retrieval effective.
6.5 Hyperparameter Choices

In this section, we present validation set results as well as the hyperparameter choices for the multilingual machine translation. Only two hyperparameters have been tuned on the validation sets, the interpolation parameter $\lambda$ and the softmax temperature $T$. The number of neighbors $k$ has been fixed to 64, the number of clusters searched has been set to 32 and the beam size has been set to 5. For the number of clusters in the FAISS index, preliminary experiments showed that for larger datastores, while using more clusters does not hurt performance, it does significantly speed up the search process since searching within the clusters is exhaustive. Hence, we use 131K clusters for the multilingual experiments.

Table 6.3 shows the validation set BLEU scores for the multilingual experiments as well as the hyperparameter choices. Values for the interpolation parameter lie between 0 and 1. We also note that for a fixed value of $\lambda = 0.5$, using $k$NN-MT either performs similarly to or improves the base MT model’s performance, but never hurts, on validation sets across the 17 language pairs evaluated in Section 6.3.2. For the temperature, we find that values of either 10 or 100 are optimal for all of our experiments.

6.6 Qualitative Analysis

To better understand $k$NN-MT, we examine the retrievals for several examples. We use the German-English model and generate with only the $k$NN distribution ($\lambda = 1$) with beam size 1, retrieving $k = 8$ neighbors from the News Commentary and Common Crawl subsets of WMT’19 data.

Figure 6.4 shows an example from newstest2018 where all the retrieved neighbors map to the same target, military. Many of the retrieved examples include phrases similar to tamed the military such as Autoritä
Table 6.3: Multilingual machine translation with $k$NN-MT on the validation set. We show the the tuned interpolation parameter ($\lambda$) as well as the tuned softmax temperature ($T$) for each language pair.

gegenüber dem Militär, Kontrolle des Militärs and das Militär gezwungen on the source side and authority over the military, control over the military and forced the military given the local target side context, but differ sharply in their longer context, often describing different nations and centuries. Figure 6.5 shows a case where the model has very little target-side prior context and mainly relies on the source context to retrieve the best neighbors. These examples also illustrate the fact that memorization adds interpretability to the model predictions because when generating translations by retrieval only, we can inspect the examples responsible for each token that is generated. This could help us better understand the model predictions, and perhaps highlight ways in which they could be improved.

Another interesting observation is that $k$NN, even when not interpolated with the base model, is able to reconstruct named entities that are split into multiple subword tokens, even if that particular named entity does not appear in the datastore. One such example is the name Haysom that is split into subwords Hay and som. The retrieved neighbors for the first subword token include examples that contain the names Hayes and Haydn, while those for the second include Grissom and Folsom, showing subword representations are used effectively in the nearest neighbor search.

### 6.7 Related Work

**Retrieval in Translation** Recent work has integrated retrieval of words and phrases into neural translation, to gain some of the advantages of the previous generation of word- and phrase-based methods (Brown et al., 1993; Koehn et al., 2003). For example, Zhang et al. (2018) proposed guiding models by retrieving n-grams and up-weighting the probabilities of retrieved tokens. Tu et al. (2018) use cache-based models (Grave et al.,
Test Input: Dabei schien es, als habe Erdogan das Militär gezähmt.
Generated tokens: In doing so, it seems as if Erdogan has tamed the

<table>
<thead>
<tr>
<th>Training Set Translation Context (source and target)</th>
<th>Training Set Target</th>
<th>Context Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dem charismatischen Ministerpräsidenten Recep Tayyip Erdogan, der drei aufeinanderfolgende Wahlen für sich entscheiden konnte, ist es gelungen seine Autorität gegenüber dem Militär geltend zu machen.</td>
<td>The charismatic prime minister, Recep Tayyip Erdoğan, having won three consecutive elections, has been able to exert his authority over the military</td>
<td>military</td>
</tr>
<tr>
<td>Ein bemerkenswerter Fall war die Ermordung des gemäßigten Premiers Inukai Tsuyoshi im Jahre 1932, die das Ende jeder wirklichen zivilen Kontrolle über das Militär markiert.</td>
<td>One notable case was the assassination of moderate Prime Minister Inukai Tsuyoshi in 1932, which marked the end of any real civilian control of the military</td>
<td>military</td>
</tr>
<tr>
<td>Sie sind Teil eines Normalisierungsprozesses und der Herstellung der absoluten zivilen Kontrolle über das Militär und bestätigen das Prinzip, dass niemand über dem Gesetz steht.</td>
<td>They are part of a process of normalization, of the establishment of absolute civilian control of the military.</td>
<td>military</td>
</tr>
<tr>
<td>Diese hart formulierte Erklärung wurde als verschleierte, jedoch unmissverständliche Warnung angesehen, dass das Militär bereit wäre einzuschreiten...</td>
<td>That toughly worded statement was seen as a veiled but unmistakable warning that the military...</td>
<td>military</td>
</tr>
</tbody>
</table>

Final kNN distribution: military = 1.0
Final Translation: In doing so, Erdogan seemed to have tamed the military.
Reference: In doing so, it seems as if Erdogan has tamed the military.

Figure 6.4: Example retrievals using kNN-MT. Not only do the retrievals all correctly predict the target word military, but the local contexts tend to be semantically related. Both the source and the three nearest retrievals express the concept of control over the military.

Other work has retrieved complete example translation sentences at test time. Nagao (1984) proposed example-based MT for translating sequences by analogy. As discussed in Section 4.2, before deep learning was widely adopted, this approach was extended to identifying portions of the training data that could
**Test Input:** Wir werden das Beste tun, mit dem, was wir haben.

**Generated tokens:** We

<table>
<thead>
<tr>
<th>Training Set Translation Context (source and target)</th>
<th>Training Set Target</th>
<th>Context Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wir werden versuchen zu beweisen, dass die Vermutung falsch ist, dies zu tun, nur um ein Gegenbeispiel Leinwände, dass die Aussage falsch ist, zu finden.</td>
<td>We</td>
<td>will</td>
</tr>
<tr>
<td>Allerdings, wenn man sich diese große Nation, wird diese Nation aussehen zu weit, und wir werden das tun, was wichtig und wertvoll, Identity Wiederherstellen der Nation.</td>
<td>However, if you look at this great nation, this nation will look too wide and we</td>
<td>will</td>
</tr>
<tr>
<td>Wir werden alles tun, um die Dinge für die Anfänger sehr einfach zu machen, während wir es den Experten erlauben, Dinge zu verändern, falls sie wollen.</td>
<td>We</td>
<td></td>
</tr>
<tr>
<td>“Wir werden ihre Fälle und die Fälle anderer politischer Gefangener vor die Gerichte bringen und das falsche Bild zerstören...”</td>
<td>“We”</td>
<td>are</td>
</tr>
</tbody>
</table>
| ... | ... | ... | ...

**Final kNN distribution:** will = 0.639, are = 0.238, intend= 0.123

**Final Translation:** We will do the best we can with what we have.

**Reference:** We’ll do the best we can with what we got.

Figure 6.5: An example where the model has access to a very short amount of target-side context that is ambiguous. kNN-MT is able to rely on source context to resolve this and generate the correct target token, *will*.

be a translation based on edit distance (Doi et al., 2005), matching training examples based on local trigram contexts (van den Bosch et al., 2007b), using phrase-based memories (van Gompel et al., 2010) and incorporating syntactic features when retrieving similar examples (Stroppa et al., 2007; Haque et al., 2009). Recently, Gu et al. (2018) proposed a model that retrieves examples similar to the test source sequence and then attends over this subset of retrieved source-target pairs at the token level, while generating translations. Bulte and Tezcan (2019) and Xu et al. (2020) use fuzzy-matching with translation memories and augment source sequences with retrieved source-target pairs. These techniques face challenges in identifying relevant retrieval candidates, as they focus on sentence-level retrieval. In contrast, kNN-MT focuses on token level retrieval from billions of key-value pairs, meaning that each word can retrieve the most relevant examples for its translation.
Retrieval in Text Generation  Retrieval mechanisms have also been applied to generation tasks more broadly. Weston et al. (2018) and Fan et al. (2020) improve dialogue response generation systems by retrieving examples and concatenating them to model inputs. Lewis et al. (2020) improve open-domain question answering systems by retrieving relevant contexts from Wikipedia and concatenating them to the inputs. Hashimoto et al. (2018) use a retrieve-and-edit framework to generate structured outputs such as code, by jointly training the editor and retriever. For $k$NN-MT, retrieval results in a distribution over the vocabulary that is used for generation directly and does not require further training or providing the retrieval candidates as input.

6.8 Conclusion

In this chapter, we have introduced $k$NN-MT, a simple and effective memorization-based approach that can be applied to any neural MT model without the need for further training. This model is similar to the previously described $k$NN-LM model, but demonstrates that MT decoder representations encode a combination of information from the input sequence in the source language as well as the prior context in the target language, making these representations particularly effective for retrieval in more complex embedding spaces. Experimental results show that memorization is, in fact, quite a general approach that is useful beyond the density estimation problem for language modeling. This approach also makes translation models more interpretable.

Given the massive scale of the datastores used in this study, similar to $k$NN-LM, reducing the size of the memories by removing the less helpful examples is important to make the method more effective and to reduce latency. However, since these models involve generating entire sequences word-by-word, the importance of the quality of data that is memorized becomes key. We observed that noisy examples can lead to noisy retrievals and degenerate translations when adding extra back-translated parallel data into the datastore for the German-English model from Section 6.3.1. This shows that larger datastores are only helpful when the examples stored meet some quality threshold. Further investigation is necessary in order to understand the true effects of noise on model behavior and whether certain types of errors are more detrimental than others.
Chapter 7

Domain Adaptation

So far, we have seen that memorization improves model generalization in a range of settings including by scaling models to larger corpora and by facilitating task specialization. However, inputs to models do not always belong to the same distribution as the data that they are trained on. Adversarial examples, distributional shifts caused over time and data from different domains are all examples of settings where the model may need to adapt to inputs for the same task but generated by different underlying distributions. For this reason, we want to understand if memorization can help the model to adapt to inputs from different distributions, that is, whether memorization improves out-of-distribution generalization.

In this dissertation, we study the problem of domain adaptation, which is a specific instance of the out-of-distribution evaluation setting. Suppose you have access to a pre-trained machine translation (MT) model that translates German news articles into English. However, you now need to use this model to translate German medical texts into English. Since news articles and medical texts differ in terms of content and structure, we say they belong to different domains and are generated by different underlying distributions. And since the MT model has not been trained on medical data, it demonstrates poor and unreliable performance when translating medical texts (see Table 7.2).

There are typically two formulations of the domain adaptation problem, as summarized by Daumé III (2009). First, when apart from having access to labeled data from the original domain, called the source domain, we also have access to labeled data from the target domain.\(^1\) Second, when we have access to labeled data only from the source domain, but have access to unlabeled data from the target domain. We use the term *zero-short transfer* to describe the direct application of a model to a new domain without any data from the target domain.

A popular approach to domain adaptation, in presence of unlabeled data in the target domain, is domain adversarial training which creates a distinction between domain-specific and domain-agnostic features at the representation learning stage, using the latter for transfer (Ganin et al., 2016). Recently, domain adaptive

\(^1\)Here we assume that under the self-supervised learning approach language modeling data is labeled. Therefore, domain adaptation for language models always belongs to this category.
training has also been shown to be highly effective for this setting (Farajian et al., 2017; Gururangan et al., 2020). When provided access to labeled data, as in the first setting, the model can be directly fine-tuned on this target domain data. However, fine-tuning can cause catastrophic forgetting of the original domain, which then needs to be mitigated (Thompson et al., 2019). A solution where a single model can be made effective in several new domains simultaneously, is more desirable in order to avoid training \( n \) models for \( n \) domains. While jointly fine-tuning the model on a combination of many domains is viable (Khayrallah et al., 2018), it can be challenging to design the right optimization objective to achieve this, and to anticipate a comprehensive notion of distributional shifts.

In this work, we focus on the first setting and assume access to labeled data in the target domain. In addition, our goal is to understand the extent to which memorization can adapt a single model to many domains and improve performance beyond zero-shot transfer. Specifically, in this chapter, we verify the hypothesis that by memorizing domain-specific data, an existing model should be able to adapt to different domains without having to train on data from those domains. We do this by designing domain adaptation experiments for language modeling and machine translation. We begin by formally defining the problem of domain adaptation. Then, we share experiments and results for using \( k \)NN-LM and \( k \)NN-MT to adapt pre-trained models to new domains, establishing a new state-of-the-art for the translation experiments without any additional model training. We also provide additional insights on how varying the datastore impacts performance. Specifically, for the translation experiments we evaluate the usefulness of a large but out-of-domain datastore, as well as a datastore that contains data from a large collection of domains. These investigations were originally presented in Khandelwal et al. (2020) for language modeling and Khandelwal et al. (2021) for machine translation.

### 7.1 Task Definition

In this section, we formally define the domain adaptation setting for the subsequent experiments. Recall that the supervised learning problem typically involves training a model \( f \), parametrized by \( \theta \), on a dataset of examples \((x_{i,S}, y_{i,S}) \in \mathcal{D}_S\) for \( i = 1, \ldots, m_S \), where \( S \) denotes that the data is generated by some underlying distribution which we call the source distribution. Through maximum likelihood estimation, the model parameters are estimated by minimizing the negative log-likelihood of the data \( \mathcal{D}_S \), where log-likelihood \( \ell \) is defined as:

\[
\ell(\theta) = \frac{1}{m_S} \sum_{i=1}^{m_S} \ln p(y_{i,S}|x_{i,S}; \theta)
\]  

(7.1)

The objective is to learn a model that can generalize to test examples \( \mathcal{D}_{S,\text{test}} \) drawn from the same distribution \( S \).

The problem of domain adaptation arises when we wish to transfer the model to test samples \( \mathcal{D}_{T,\text{test}} \) that come from a different distribution which we call the target distribution \( T \). In other words, we wish to adapt the model from the source domain to the target domain. Note that the underlying task is the same, only the
Training Data | Datastore | Perplexity (\(\cdot\))
---|---|---
WIKI-3B | - | 37.13 | 34.84
BOOKS | - | 14.75 | 11.89
WIKI-3B | BOOKS | 24.85 | 20.47

Table 7.1: Domain adaptation experiments, with results on BOOKS. Adding an in-domain datastore to a Wikipedia-trained model improves results by 23 points, approaching in-domain training.

Table 7.1: Domain adaptation experiments, with results on BOOKS. Adding an in-domain datastore to a Wikipedia-trained model improves results by 23 points, approaching in-domain training.

Table 7.1: Domain adaptation experiments, with results on BOOKS. Adding an in-domain datastore to a Wikipedia-trained model improves results by 23 points, approaching in-domain training.

<table>
<thead>
<tr>
<th>Training Data</th>
<th>Datastore</th>
<th>Perplexity ((\cdot))</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIKI-3B</td>
<td>-</td>
<td>37.13</td>
</tr>
<tr>
<td>BOOKS</td>
<td>-</td>
<td>14.75</td>
</tr>
<tr>
<td>WIKI-3B</td>
<td>BOOKS</td>
<td>24.85</td>
</tr>
</tbody>
</table>

7.2 Experiments

7.2.1 Domain Adaptation for Language Modeling

We use the \(k\)NN-LM model proposed in Chapter 5 to evaluate the effectiveness of memorization for domain adaptation of LMs.

Setup For this experiment, we use two datasets from the \(k\)NN-LM study. WIKI-3B, which includes English Wikipedia articles containing about 2.87B tokens, is the source domain. BOOKS, which is the Toronto Books Corpus (Zhu et al., 2015) containing 0.7B tokens, is the target domain. Complete books are held out for validation/test and are used for tuning hyperparameters/evaluation. We also use the same Transformer LM trained on WIKI-3B from Chapter 5, along with the same settings for creating the datastore.

Results Table 7.1 shows that an in-domain LM on BOOKS has a relatively low perplexity (11.89), while a model trained on WIKI-3B performs poorly on the BOOKS domain (34.84 perplexity). Adding \(k\)NN search over BOOKS to the WIKI-3B model reduces perplexity by 14 points (to 20.47), demonstrating that \(k\)NN-LM allows a single model to be useful in multiple domains, by simply adding a datastore per domain.

Tuning the Interpolation Parameter We use a parameter \(\lambda\) to interpolate between the base model distribution and the distribution from \(k\)NN search over the dataset. Figure 7.1 shows that \(\lambda = 0.25\) is optimal.
CHAPTER 7. DOMAIN ADAPTATION

Figure 7.1: Effect of interpolation parameter $\lambda$ on in-domain (left y-axis) and out-of-domain (right y-axis) validation set performances. More weight on $p_{kNN}$ improves domain adaptation.

for the in-domain BOOKS model, which is the in-distribution setting explored in Chapter 5 (see Table 5.2 for the final perplexity scores). However, $\lambda = 0.65$ works best for domain adaptation results (Figure 7.1). This contrast shows that the model relies more on retrieval for data that it was not trained on, as in the case of domain adaptation.

7.2.2 Domain Adaptation for Machine Translation

We also measure the effectiveness of $k$NN-MT for domain adaptation.

Setup For these experiments, we use the MULTI-DOMAINS dataset (Koehn and Knowles, 2017), re-split by Aharoni and Goldberg (2020). It includes German-English parallel data for train/validation/test sets in five domains: Medical, Law, IT, Koran and Subtitles. We use the German-English translation model from Section 6.3.1 as our base MT system and we also use the NEWSTEST German-English validation and test sets as our in-domain evaluation data. Domain-specific data for the five target domains is provided in the datastores and we use 1M keys to learn 4K cluster centroids for each. We also explore the effects of retrieving neighbors from a large amount of out-of-domain data as well as from a single multi-domain datastore.

Domain-specific datastores Table 7.2 shows the base MT system’s in-domain performance on newstest2019, as well as zero-shot transfer to five other domains. $k$NN-MT significantly outperforms the base MT system in all settings. For the multi-domains dataset, $k$NN-MT improves the base MT model performance by an average of 9.2 BLEU, with improvements as large as 16 BLEU on Law and 14.5 BLEU on Medical, all without any further training. We also provide scores from Aharoni and Goldberg (2020) for models trained on in-domain data, those trained on all domains jointly, and those trained using the best-performing data
CHAPTER 7. DOMAIN ADAPTATION

<table>
<thead>
<tr>
<th>Test set sizes</th>
<th>Newstest 2019</th>
<th>Medical</th>
<th>Law</th>
<th>IT</th>
<th>Koran</th>
<th>Subtitles</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Aharoni and Goldberg (2020):
- one model per domain: 37.59 53.3 54.8
- one model for all domains: 39.08 54.35
- best data selection method: 38.88 54.54

### Base MT
- 37.59 39.91 45.71 37.98 16.30 29.21 33.82

### +kNN-MT:
- in-domain datastore: 39.08 54.35 61.78 45.82 19.45 31.70 42.63
- WMT’19 datastore: 39.08 40.22 46.74 40.27 17.99 29.23 34.89
- all-domains datastore: 38.88 54.54 61.11 48.63 19.22 31.70 43.04

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>770M</td>
<td>48.07</td>
<td>39.94</td>
<td>45.78</td>
<td>35.78</td>
<td>16.30</td>
<td>29.74</td>
<td>33.51</td>
</tr>
</tbody>
</table>

### Table 7.2: Domain adaptation using kNN-MT. The base MT system is trained on WMT’19 data which is also treated as the in-domain data for newstest2019. kNN-MT improves the base model by an average of 9.2 BLEU, without training, to achieve the best reported results on this task.

### Table 7.3: Domain adaptation using kNN-MT on the multi-domains validation data and newstest2018. The base MT system is trained on WMT’19 data which is also treated as the in-domain data for newstest2018. We present the interpolation ($\lambda$) and softmax temperature ($T$) hyperparameter choices for each domain.

### Out-of-domain and multi-domain datastores
- Table 7.2 also shows performance for retrieving neighbors from 770M tokens of WMT’19 data that the model has been trained on. While the average BLEU for the multi-domain data is 1 point higher, the improvements are much smaller compared to using in-domain data. This illustrates the value of adding domain-specific data to the datastore over adding a large amount of arbitrary data. We also measure the effectiveness of building a single multi-domain datastore containing parallel data from all six settings. Performance on IT improves by 3 BLEU but scores for the other domains are mostly the same. This shows that kNN-MT is robust to the presence of out-of-domain examples since retrieving neighbors from a datastore where large amounts of data is out-of-domain does not hurt performance relative to using only in-domain data.

- Selection method proposed by the authors. We find that kNN-MT also outperforms the best reported average on the multi-domains dataset by 1.4 BLEU.
CHAPTER 7. DOMAIN ADAPTATION

Related Work  Various studies have explored retrieving additional information to improve domain adaptation, often using lexicons (Hu et al., 2019), domain-adaptive training (Farajian et al., 2017) or attending over neighbors similar to n-grams in the source (Bapna and Firat, 2019). These modifications require additional training, whereas kNN-MT provides the flexibility to use different datastores when decoding in different domains, keeping the model fixed.

7.3 Conclusion

In this chapter, we have shown that memorization-based models are extremely effective for domain adaptation. Specifically, we have shown that, in the case of language modeling and machine translation, existing models can be adapted to new domains by simply storing domain-specific examples in an external memory and querying this memory during inference, without any in-domain training. Experiments demonstrate that when adapting to new domains the model relies on the datastore more than its parameters, with values for the interpolation parameter being higher compared to the in-distribution setting. In addition, we find that small in-domain datastores are far more valuable than large out-of-domain datastores, and that cross-domain retrieval by using a single multi-domain datastore may not always be useful.

The effectiveness of memorization for domain adaptation suggests that the representations learned by these models are, in fact, domain-agnostic. Understanding how fine-tuning on in-domain data differs from directly using model representations for retrieval could help to shed some light on the kinds of information that is being encoded in model parameters after fine-tuning, and the extent to which novel generalization behaviors emerge beyond memorizing information in parameters. This kind of analysis would not only help us understand how existing models learn to generalize, but will also provide insights on when memorization versus fine-tuning serves as the better approach.
Chapter 8

Conclusion

The monolithic challenge of improving neural language models, and consequently most other NLP models, is faced with two key problems: (1) the lack of model interpretability, and (2) the inability of these models to generalize consistently in a range of settings. In this dissertation, we have discussed these problems and proposed methods that allow us to make progress towards solving them. As we conclude, we first provide a summary of the work presented, and then share some thoughts on future directions.

8.1 Thesis Summary

In Chapter 3, we describe our analytic study of neural LMs which was motivated by the fact that we knew very little about how neural language models used prior linguistic context. We present our study where we investigate the role of context in an LSTM LM via black-box analysis, which allowed us to examine how model predictions changed in response to changes in inputs. Specifically, we analyze the increase in perplexity when prior context words are shuffled, replaced, or dropped. On two standard datasets we find that the model is capable of using about 200 tokens of context on average, but sharply distinguishes nearby context (recent 50 tokens) from the distant history. The model is highly sensitive to the order of words within the most recent sentence, but ignores word order in the long-range context (beyond 50 tokens), suggesting the distant past is modeled only as a rough semantic field or topic. We further find that the neural caching model Grave et al. (2017c) especially helps the LSTM to copy words from within this distant context. Overall, our analysis serves two purposes in relation to interpretability, namely adding transparency to decisions by providing a better understanding of how neural LMs use their context, and suggesting ways in which these models could be improved.

Then, we discuss our work on Generalization through Memorization. In Chapter 4, we provide an overview for the proposed approach including discussions for motivations and connections with exemplar models. We highlight the limitations of existing neural language models which seemingly memorize information in their parameters in a brittle fashion, are unable to adapt to new and changing environments, require
tremendous amounts of compute and lack interpretability. Our proposed version of memorization, on the other hand, extends existing models with examples stored in an external, non-parametric memory, does not add any additional training costs, adds interpretability to model predictions and, most of all, improves model generalization both in- and out-of-distribution. We also describe the general framework for memorization in a task-agnostic fashion which is specialized for language modeling and machine translation in subsequent chapters.

In Chapter 5, we introduce $k$NN-LMs, which extend a pre-trained neural language model by linearly interpolating it with a $k$-nearest neighbors ($k$NN) model. The nearest neighbors are computed according to distance in the pre-trained LM embedding space, and can be drawn from any text collection, including the original LM training data. Applying this augmentation to a strong WIKITEXT-103 LM, with neighbors drawn from the original training set, our $k$NN-LM achieves a new state-of-the-art perplexity of 15.79 – a 2.9 point improvement with no additional training. We also show that this approach has implications for efficiently scaling up to larger training sets by simply varying the nearest neighbor datastore, again without further training. Qualitatively, the model is particularly helpful in predicting rare patterns, such as factual knowledge. Together, these results strongly support our hypothesis that learning similarity between sequences of text is easier than predicting the next word, and that nearest neighbor search is an effective approach for language modeling in the long tail.

In Chapter 6, we introduce $k$NN-MT, which predicts tokens with a nearest neighbor classifier over a large datastore of cached examples, using representations from a neural translation model for similarity search. Following the memorization framework, this approach requires no additional training, and its non-parametric nature allows scaling to give the decoder direct access to billions of examples at test time, resulting in a highly expressive model that consistently improves performance across many settings. Simply adding nearest neighbor search improves a state-of-the-art German-English translation model by 1.5 BLEU. $k$NN-MT also allows task specialization for a massively multilingual translation model by adapting it to particular language pairs at test time, with improvements of over 3 BLEU observed for translating from English into German and Chinese. Qualitatively, $k$NN-MT is easily interpretable since predictions can be traced back to specific retrieved examples; it combines source and target context to draw on highly relevant memorized examples.

And finally in Chapter 7, we discuss the problem of domain adaptation and demonstrate the effectiveness of memorization in improving out-of-distribution generalization. Specifically, we design experiments for language modeling and machine translation using base models that have been trained on domains separate from the evaluation set. By storing domain-specific data in the nearest neighbor datastore, we find that memorization allows models to adapt to new domains without having to train on any in-domain data. For language modeling, we show that $k$NN-LM is extremely effective in adapting to new domains with a 14 point perplexity improvement when adapting from the domain of Wikipedia to Books. For machine translation, we show that $k$NN-MT allows a single model to be adapted to a diverse set of five domains by simply using a domain-specific datastore for each, improving results by an average of 9.2 BLEU over zero-shot transfer, and achieving new state-of-the-art results, without training on any of these domains.
8.2 Future Directions

The ideas discussed in this dissertation present novel avenues for the NLP community to explore on our quest towards building interpretable NLP systems that can generalize and adapt in a range of settings. Moving forward, as we build on the ideas of memorization presented in this work, using analysis to better understand why this approach works so well will further help us understand how and when to apply it. One of the most pressing questions that remains unanswered is: when is memorization really helping? Qualitative analyses have provided some intuitions for this. One hypothesis is that memorization helps with long-tail patterns like factual knowledge which need to be memorized, such as facts like *Pride and Prejudice was written by Jane Austen*. Such information needs to be memorized by the language model but its capacity limitations might prevent it from storing the fact in its parameters, which is where an external memory can be helpful. Another hypothesis could be that memorization helps with disambiguating contexts where precise information is necessary for making a prediction. For instance, differentiating between two contexts with high lexical overlap such as *Elizabeth Bennet is the protagonist of the famous Jane Austen novel ______ versus Emma Woodhouse is the protagonist of the famous Jane Austen novel ______*.\(^1\) However, in order to move past mere hypotheses and uncover the precise nature of when memorization is useful, we would need to conduct a carefully designed analytic study where we control for specific types of contexts and target words, and quantitatively identify the groups for which this approach causes the largest improvements.

The results of such a study could also be impactful in reducing the size of our nearest neighbor datastores and improving inference speed, which can be a hindrance to deploying these models in the real world. While naïvely subsampling items from the datastore is unhelpful, removing examples based on a dynamic interpolation parameter could be a promising direction. Currently, our experiments involve manually tuning the interpolation parameter which is then fixed to the same value for all samples. This brute-force setting for the hyperparameter could be obscuring the true use-cases for memorization and may even be harmful in cases where the relevant examples are not present in the datastore.

The analysis of when memorization is helpful would also shed more light on the symbiotic relationship between the non-parametric memory and the parametric model. Can a smaller memory-augmented model generalize as well as a large parametric model? Is there a specific set of examples or a new optimization objective that could make the smaller memory-augmented models more effective? How does memorization impact the few-shot learning behaviors of language models that we have seen emerging in the past year (*Brown et al., 2020; Schick and Schütze, 2020*)? These questions serve as deep learning counterparts to the long-standing abstract representations versus exemplar theory debate, discussed in Section 4.2, about whether humans represent categories using individual exemplars (like in memorization) or by abstractions of the general categories which might be what model parameters are learning to represent. We hypothesize that, similarly to the trade-offs explored in relation to categorization, selecting for the right combination of the two would lead to better generalization for existing models, as preliminarily demonstrated by our experiments.

---

1Elizabeth Bennet is the protagonist of *Pride and Prejudice*, while Emma Woodhouse is the eponymous protagonist of the novel *Emma*. 
with $k$NN-LM and $k$NN-MT. But, answering the aforementioned questions would be an important step along the way to refining and ultimately verifying this hypothesis, evidenced by the evolution of exemplar theory over the decades.

As we think about when memorization is helpful, it is also interesting to think about what kinds of tasks are amenable to generalization through memorization. In the case of our work on language modeling and machine translation, the keys in the key-value datastores were LM context and translation context representations. Such a formulation could be applicable for a number of other tasks as well, such as automatic speech recognition, code generation, chatbots and conversational agents, and more. However, memorizing such contexts and retrieving from previously seen examples may be less useful for other tasks like, for instance, text summarization where the model is required to generate a short summary containing the most salient information from some given document. Given a news article based on current affairs, previously seen examples would not contain the facts present in the input and could lead to hallucinations. However, in such cases, there might still be meta-patterns that could be worth memorizing. For example, what are the key pieces of salient information in most sports articles? Or which are the most important clauses in a particular type of legal contract? Thus, it is important to emphasize that the memorization approach presented in this thesis is more generally applicable to a range of settings, where choosing what to memorize and how to measure distances between examples are key decisions to consider.

As the NLP community works towards the goal of building models that can understand and generate language like humans, we hope the ideas in this dissertation will inspire work on making models more interpretable and trustworthy, understanding our best models which will help us understand how best to use them, as well as improving generalization by exploring creative and effective approaches beyond just model scaling.
Bibliography


Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. 2019. What does BERT look at? an analysis of BERT’s attention. In BlackBoxNLP@ACL.


Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.


Rejwanul Haque, Sudip Naskar, Yanjun Ma, and Andy Way. 2009. Using supertags as source language context in SMT. In *Conference of the European Association for Machine Translation (EAMT)*.


Zachary C Lipton. 2018. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57.


Hongyin Luo, Lan Jiang, Yonatan Belinkov, and James Glass. 2019. Improving neural language models by segmenting, attending, and predicting the future. In *Association for Computational Linguistics (ACL)*.


Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR’s WMT19 news translation task submission. In *Conference on Machine Translation (WMT)*.


Adam Pauls and Dan Klein. 2011. Faster and smaller n-gram language models. In *Association for Computational Linguistics (ACL)*.


Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Association for Computational Linguistics (ACL)*.


Jesse Vig. 2019. A multiscale visualization of attention in the transformer model. In *Association for Computational Linguistics: System Demonstrations (ACL)*.


Jason Weston, Emily Dinan, and Alexander H Miller. 2018. Retrieve and refine: Improved sequence generation models for dialogue. In *Workshop on Search-Oriented Conversational AI (SCAI@EMNLP)*.


