IMPLICIT LINGUISTIC KNOWLEDGE OF NEURAL NATURAL LANGUAGE PROCESSING MODELS

NELSON F. LIU

A senior thesis
submitted in partial fulfillment of the
requirements for the degree of

Bachelor of Science
with Departmental Honors

Paul G. Allen School of Computer Science & Engineering
University of Washington

June 2019

Supervisor: Noah A. Smith

Approved by: _____

(Noah A. Smith)

Presentation of work given on May 30th, 2019.

To everyone who has taken a chance on me,
with gratitude.

ABSTRACT

Language technologies have entered the mainstream, driven by the success of neural network-based machine learning models for language processing. Although these neural natural language processing (NLP) systems enjoy increased performance, their internal workings remain poorly understood. Consequently, when neural models fail, it's difficult to discern what knowledge they are missing, let alone find principled ways of improving upon them.

This thesis seeks to better understand the abilities and limitations of neural NLP systems by characterizing their *implicit linguistic knowledge*.

We begin by studying recurrent neural networks (RNNs), which have found success in a variety of natural language processing applications. However, they are inherently general models of sequential data. We investigate how the properties of natural language data affect an RNN's ability to learn a non-linguistic task: recalling elements from its input.

We find that RNN models trained on natural language data are able to recall tokens from much longer sequences than models trained on non-language sequential data—this indicates that RNNs exploit linguistic attributes of data, even when trained to perform inherently non-linguistic tasks, and that such features improve model generalization. Furthermore, inspecting model internals reveals that the RNN solves the memorization task by using a subset of its neurons as explicit counters. We hypothesize that the patterns and structure in natural language data enable RNNs to learn by providing approximate ways of reducing loss.

To further understand how RNNs (and other neural network components of modern NLP systems) encode information, we investigate the linguistic knowledge implicitly captured by currently-unstructured NLP systems.

In particular, we study contextual word representations derived from large-scale neural language models. We (1) analyze what linguistic knowledge they capture to better understand their strengths and weaknesses, (2) quantify differences in transferability of individual contextualizer layers, and (3) compare language model pretraining with eleven supervised pretraining tasks to better understand what makes contextual word representations transferable.

These analyses enable us to better understand the inner workings of our currently-opaque neural NLP systems, a crucial step towards their principled enhancement.

BIBLIOGRAPHIC NOTE

The ideas and figures within this thesis have appeared previously in the following peer-reviewed publications:

Liu, Nelson F., Omer Levy, Roy Schwartz, Chenhao Tan, and Noah A. Smith (2018). "LSTMs Exploit Linguistic Attributes of Data." In: *Proc. of RepL4NLP*.

Liu, Nelson F., Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith (2019). "Linguistic Knowledge and Transferability of Contextual Representations." In: *Proc. of NAACL*.

Chapter 2 is based on Liu et al. (2018), and Chapter 3 is based on Liu et al. (2019). The code developed in this thesis is available at:
http://nelsonliu.me/papers/lstms-exploit-linguistic-attributes
http://nelsonliu.me/papers/contextual-repr-analysis

ACKNOWLEDGMENTS

I have been incredibly lucky over the last four years. First, thanks to Noah Smith, who gave me a chance to work on NLP research far before I was remotely qualified to do so. His support is inspiring—thanks in particular for taking the time and effort to meet one-on-one, even when I was still learning the ropes in my early years. His technical advice has refined my research taste; his mentorship helped a lost teenager find a path in life and made me the person I am today. Working with Noah truly changed my life, and it is difficult to overstate how grateful I am for the opportunity and experience.

Many others have mentored me, and all have been unique influences. Thanks to Yanchuan Sim—I will always remember your patiently working through the details of logistic regression with me in the 3rd floor breakout. Matt Gardner mentored me during two internships at the Allen Institute for Artificial Intelligence (AI2). The first time around, I completely destroyed his productivity with my constant nagging; I am thankful for the time he has poured into me. I hope to one day emulate his ability to pierce into the essence of research questions and issues. During a summer at USC ISI, Kevin Knight and Jonathan May emphasized the importance of poring over the data and the output, and they instilled a desire to build and appreciation of systems that actually work. I was also incredibly lucky to work with the postdoc dream team of Omer Levy, Roy Schwartz, and Chenhao Tan, and their experience and wisdom helped me begin to learn how to identify and tackle important questions. Roy Schwartz, in particular, has been a constant mentor and friend—thanks for your invaluable insights on research and life.

I have also had the privilege of working with many other interesting and insightful researchers. Thanks to my co-authors and collaborators: Yonatan Belinkov, Pradeep Dasigi, Gina-Anne Levow, Robert Logan, Ana Marasović, Matt Peters, Sameer Singh, and Johannes Welbl.

I have benefitted immensely from interactions with Yejin Choi and Luke Zettlemoyer, as well. Their kindness, willingness to listen, and input were especially helpful when I was figuring out my next steps.

I was fortunate to join an amazingly supportive and welcoming research group in the ARK. Thanks to everyone, past and present: Judit Ács, Tal August, Dallas Card, Elizabeth Clark, Jesse Dodge, Sarah Dreier, Emily Gade, Suchin Gururangan, Nikita Haduong, Yangfeng Ji, Jungo Kasai, Rik Koncel-Kedziorski, Lingpeng Kong, Adhi Kuncoro, Lucy Lin, Yijia Liu, Kelvin Luu, Phoebe Mulcaire, Hao Peng, Ofir Press, Maarten Sap, Roy Schwartz, Sofia Serrano, Yanchuan Sim, Swabha Swayamdipta, Chenhao Tan, Sam Thomson, Qiaolin Xia, and

Yi Zhu. Conversations with Elizabeth, Jesse, Kelvin, Ofir, Sofia, and Swabha have been particularly formative. Thanks also to the excellent undergrad and master's contingent: Emma Casper, Ethan Chau, Tam Dang, Ron Fan, Sean Hung, Sam Gehman, Leo Liu, Karishma Mandyam, Aishwarya Nirmal, Amy Shah, Deric Pang, and Michael Zhang.

The broader UW NLP group has also been a great source of inspiration and fun. In particular, Antoine Bosselut, Max Forbes, Ge Gao, Julian Michael, and Victor Zhong have often lent me their hands, eyes, and ears, and I am thankful to count them as friends.

I am also thankful for the opportunity to spend extended time at AI2, and I have learned much from Waleed Ammar, Christine Betts, Pradeep Dasigi, Xinya Du, Matt Gardner, Joel Grus, Vidur Joshi, Kevin Lin, Nicholas Lourie, Ana Marasović, Mark Neumann, Amandalynne Paullada, Matt Peters, Sarah Pratt, Vivek Ramanujan, Kyle Richardson, Brendan Roof, Michael Schmitz, Gabi Stanovsky, Eric Wallace, and Mitchell Wortsman.

Throughout the years, I have shared an office and space with many wonderful people. Thanks in particular to Sam Ainsworth, Jesse Dodge, Rahul Kidambi, Pascal Sturmfels, and Michael Zhang for making CSE 609 feel like home—I've enjoyed our many conversations about research and life. I was also happy to spend a summer and fall in the AI2 zoo with Christine Betts, Zachary Horvitz, Roy Or-El, Amandalynne Paullada, Swabha Swayamdipta, Erika Wolfe, and Fan Yang.

Thanks also to Laura McGarrity of the Department of Linguistics, whose introductory course inspired me to dive deeper into the field, Robin Chang of the Office of Merit Scholarships, Fellowships & Awards, who listened to hours of my aimless rambling and spent many more poring over application drafts, and Jennifer Harris of the Undergraduate Research Program, who offered me invaluable opportunities to share my experiences with others at UW.

Many have pushed me to become a better person and friend. Special thanks to Darin Chin, Ben Evans, Ge Gao, Preston Jiang, Mathew Luo, Kenny Le, Ryan Pham, Thai Pham, Shivam Singhal, and Joseph Zhong. Thanks also to Nick Bowman, Joshua Chen, Victor Chen, and Zaki Molvi, who remind me that a world exists outside of Seattle. Christine Betts and Kevin Lin challenge me in all the best ways, and I am deeply indebted to them.

Finally, thanks to my family: Mom, Dad, Emily, and Lily, for their unconditional love and support. They have put up with a lot through the years, and I hope I can repay their love and kindness in the years to come.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Human language technologies pervade everyday life, powering popular applications including voice assistants, translation tools, and speech transcription services. Their rapid proliferation is driven by recent advances in machine learning-based artificial intelligence. This recent progress in machine learning has relied massive training datasets, powerful computing resources, and models with the ability to take advantage of the aforementioned resources.

Neural network-based (so-called "deep learning") approaches have proved particularly successful, since they be efficiently trained with specialized hardware (e.g., graphics processing units; Raina, Madhavan, and Ng, 2009 and application-specific integrated circuits; Jouppi et al., 2017, *inter alia*) to learn predictive features from massive datasets (e.g., ImageNet; Deng et al., 2009 and the Common Crawl[1]). These neural models are the backbone of most modern language processing systems, and they are the heart of applications that answer questions, translate sentences, and recognize human speech.

Neural networks are especially useful because they can be trained "end-to-end", i.e., the entire system is based on a single model that is optimized to solve a task of interest from only a training dataset of input-output pairs (e.g., the neural translation system of Sutskever, Vinyals, and Le (2014)). This is in stark contrast to traditional NLP systems, which often contain multiple disparate components that are built separately and later combined (e.g., the syntax-based statistical machine translation system of Chiang (2005)).

Since every parameter in a neural model is expressly optimized for the sole purpose of performing the end task, they enjoy increased accuracy. These models have indeed produced large gains on a variety of standard NLP tasks, but this improved performance comes at the steep cost of model interpretability—it is difficult to discern what information is learned during the training process (global interpretability), or how the model makes individual decisions (local interpretability). As a result, neural systems are often thought of as opaque black boxes—they can learn to accurately map inputs to outputs, but their internal machinery is difficult to interpret or understand. Consequently, when neural NLP systems invariably fail, it is difficult to understand what requisite knowledge is missing and necessary for improving them.

Furthermore, the vast majority of neural models for language processing are *unstructured*, i.e., they do not explicitly consider the reg-

1 http://commoncrawl.org

ular patterns and structure within language. Instead, unconstrained end-to-end training, often combined with more data and faster computational resources, has historically led to the strongest performance gains (Sutton, 2019). It is possible that we simply have not discovered the appropriate linguistic biases for neural models—understanding what they are able to *implicitly* capture about language (i.e., through only end-to-end learning on linguistic inputs) may provide clues to the appropriate structures for augmentation.

For instance, explicitly integrating knowledge that is already implicitly learned through end-to-end training may reduce the burden of the learner and enable it to focus on learning the task at hand (rather than simultaneously struggling to understand the linguistic input itself), yielding models and algorithms that achieve high performance on smaller datasets. In a similar vein, explicitly integrating structures and knowledge that is not implicitly learned through end-to-end training can endow our models with the necessary tools for robust generalization to new domains and data distributions.

Motivated by the aforementioned shortcomings and opportunities, we investigate what neural natural language processing systems learn about language. Specifically,

- How does linguistic data itself affect the generalizability of recurrent neural networks? (Chapter 2)
- What do our currently-unstructured neural language models implicitly learn about language, as captured by derived contextual word representations? (Chapter 3)

Understanding the strengths and limitations of current NLP models is a crucial first step towards their principled enhancement. This serves as the foundation for an efficient research process—instead of tuning model architectures and training techniques in hopes of stumbling upon a stronger model,[2] expectations for the linguistic capabilities of NLP systems can be defined (perhaps by lingusts; Linzen, 2019), experiments are conducted to determine whether these criteria have been met, and targeted enhancements are developed to overcome the identified limitations.

For the sake of brevity, we defer to the broadly-accessible text of Goldberg (2017) for the necessary background on neural network-based approaches in natural language processing. Although each chapter contains a short section describing the related work necessary for situating it in the broader literature on the analysis of neural networks in language processing, we recommend the detailed survey of Belinkov and Glass (2019) to readers interested in learning more about current and related work in analyzing and understanding neural NLP models.

2 A strategy derisively called "graduate student descent".

1.1 SUMMARY OF CONTRIBUTIONS

This thesis conducts a multi-view analysis of the implicit linguistic knowledge of neural network-based models for natural language processing.

We begin by examining the role of linguistic data itself on the recent success of recurrent neural networks (RNNs). Although RNNs have seen great success on language-based tasks, they are inherently models of arbitrary sequential data—perhaps properties of natural language in particular augment them in some way? Through a carefully-controlled synthetic data experiment, we find that RNNs exploit linguistic attributes of data, even when trained on inherently non-linguistic tasks, and that such features improve model generalization. We also analyze how the RNN solves the task, and find that it learns to repurpose several of its neurons to count timesteps in its input.

Armed with evidence that neural NLP systems are indeed sensitive to linguistic features in their input, we set out to identify what linguistic features large-scale neural language models learn from their massive training datasets. Our layerwise approach to analyzing the internal representations of neural language models yields interesting insights regarding the behavior and internal mechanisms of end-to-end neural network-based models. The main insights are:

- Neural language models, trained in an end-to-end fashion, implicitly learn a non-trivial amount of linguistic information without direct linguistic supervision during the initial training process. Empirically, linear models trained atop of the internal representations of neural language models rival the performance of state-of-the-art task-specific models on a variety of tasks. However, performance on tasks that require fine-grained linguistic knowledge falls short of models trained with direct linguistic supervision.

- Long short-term memory (LSTM; Hochreiter and Schmidhuber, 1997) recurrent neural networks and transformers (Vaswani et al., 2017), two popular building blocks of modern neural NLP systems, exhibit marked differences in how they store features in their layers. In particular, the first layer output of LSTM recurrent neural networks is the most general, while higher layers are more task-specific. In contrast, it is the middle layers of transformers that encode the most general representations, and moving up their layers does not reveal the aforementioned monotonic increase in task-specificity.

- The representations within neural language models are more general, on average, than representations learned by neural models trained on eleven tasks with explicit linguistic supervision. However, pretraining with explicit linguistic supervision yields

representations that are the most informative for individual related tasks.

2

LSTMS EXPLOIT LINGUISTIC ATTRIBUTES OF DATA

2.1 INTRODUCTION

Recurrent neural networks (RNNs; Elman, 1990), especially variants with gating mechanisms such as long short-term memory units (LSTM; Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRU; Cho et al., 2014), have significantly advanced the state of the art in many NLP tasks (Mikolov et al., 2010; Vinyals et al., 2015; Bahdanau, Cho, and Bengio, 2015, among others). However, RNNs are general models of sequential data; they are not explicitly designed to capture the unique properties of language that distinguish it from generic time series data.

In this chapter, we probe how linguistic properties such as the hierarchical structure of language (Everaert et al., 2015), the dependencies between tokens, and the Zipfian distribution of token frequencies (Zipf, 1935) affect the ability of LSTMs to learn.[1] To do this, we define a simple memorization task where the objective is to recall the identity of the token that occurred a fixed number of timesteps in the past, within a fixed-length input. Although the task itself is not linguistic, we use it because (1) it is a generic operation that might form part of a more complex function on arbitrary sequential data, and (2) its simplicity allows us to unfold the mechanism in the trained RNNs.

To study how linguistic properties of the training data affect an LSTM's ability to solve the memorization task, we consider several training regimens. In the first, we train on data sampled from a uniform distribution over a fixed vocabulary. In the second, the token frequencies have a Zipfian distribution, but are otherwise independent of each other. In another, the token frequencies have a Zipfian distribution but we add Markovian dependencies to the data. Lastly, we train the model on natural language sequences. To ensure that the models truly memorize, we evaluate on uniform samples containing *only rare words*.[2]

We observe that LSTMs trained to perform the memorization task on natural language data or data with a Zipfian distribution are able to memorize from sequences of greater length than LSTMs trained directly on uniformly-sampled data. Interestingly, increasing the length of Markovian dependencies in the data does not significantly help

1 The work described in this chapter was done with Omer Levy, Roy Schwartz, Chenhao Tan, and Noah A. Smith, and was published as Liu et al. (2018).

2 This distribution is adversarial with respect to the Zipfian and natural language training sets.

LSTMs to learn the task. We conclude that linguistic properties can help or even enable LSTMs to learn the memorization task. Why this is the case remains an open question, but we propose that the additional structure and patterns within natural language data provide additional noisy, approximate paths for the model to minimize its loss, thus offering more training signal than the uniform case, in which the only way to reduce the loss is to learn the memorization function.

We further inspect *how* the LSTM solves the memorization task, and find that some hidden units count the number of inputs. Shi, Knight, and Yuret (2016) analyzed LSTM encoder-decoder translation models and found that similar counting neurons regulate the length of generated translations. Since LSTMs better memorize (and thus better count) on language data than on non-language data, and counting plays a role in encoder-decoder models, our work could also lead to improved training for sequence-to-sequence models in non-language applications (e.g., Schwaller et al., 2017).

2.2 THE MEMORIZATION TASK

To assess the ability of LSTMs to retain and use information, we propose a simple memorization task. The model is presented with a sequence of tokens and is trained to recall the identity of the middle token.[3] We predict the middle token since predicting items near the beginning or the end might enable the model to avoid processing long sequences (e.g., to perfectly memorize the last token, simply set the forget gate to 0 and the input gate to 1).[4] All input sequences at train and test time are of equal length. To explore the effect of sequence length on LSTM task performance, we experiment with different input sequence lengths (10, 20, 40, 60, ..., 300).

2.3 EXPERIMENTAL SETUP

We modify the linguistic properties of the training data and observe the effects on model performance.

PENN TREEBANK PROCESSING Our experiments use a preprocessed version of the Penn Treebank commonly used in the language modeling community and first introduced by Mikolov et al. (2011). This dataset has 10K types, hence why we use this vocabulary size for all experiments. We generate examples by concatenating the sentences together and taking subsequences of the desired input sequence length.

3 Or the $(\frac{n}{2}+1)$th token if the sequence length n is even.

4 We experimented with predicting tokens at a range of positions, and our results are not sensitive to the choice of predicting *exactly* the middle token.

TRAINING The model is trained end-to-end to directly predict the tokens at a particular timestep in the past; it is optimized with Adam (Kingma and Ba, 2015) with an initial learning rate of 0.001, which is halved whenever the validation dataset (a held-out portion of the training dataset) loss fails to improve for three consecutive epochs. The model is trained for a maximum of 240 epochs or until it converges to perfect validation performance. We do not use dropout; we included it in initial experiments, but it severely hampered model performance and does not make much sense for a task where the goal is to explicitly memorize. We ran each experiment three times with different random seeds and evaluate the model with the highest validation accuracy on the test set. We take the best since we are interested in whether the LSTMs *can* be trained for the task. [5]

MODEL We train an LSTM-based sequence prediction model to perform the memorization task. The model embeds input tokens with a randomly initialized embedding matrix. The embedded inputs are encoded by a single-layer LSTM and the final hidden state is passed through a linear projection to produce a probability distribution over the vocabulary.

Our goal is to evaluate the memorization ability of the LSTM, so we freeze the weights of the embedding matrix and the linear output projection during training. This forces the model to rely on the LSTM parameters (the only trainable weights), since it cannot gain an advantage in the task by shifting words favorably in either the (random) input or output embedding vector spaces. We also tie the weights of the embeddings and output projection so the LSTM can focus on memorizing the timestep of interest rather than also transforming input vectors to the output embedding space.[6] Finally, to examine the effect of model capacity on memorization ability, we experiment with different hidden state size values.

DATASETS We experiment with several distributions of training data for the memorization task. In all cases, a 10K vocabulary is used.

- In the *uniform* setup, each token in the training dataset is randomly sampled from a uniform distribution over the vocabulary.
- In the *unigram* setup, we modify the *uniform* data by integrating the Zipfian token frequencies found in natural language data. The input sequences are taken from a modified version of the Penn Treebank (Marcus, Marcinkiewicz, and Santorini, 1993) with randomly permuted tokens.

5 Code for reproducing these results can be found at http://nelsonliu.me/papers/lstms-exploit-linguistic-attributes

6 Tying these weights constrains the embedding size to always equal the LSTM hidden state size.

- In the *5gram*, *10gram*, and *50gram* settings, we seek to augment the *unigram* setting with Markovian dependencies. We generate the dataset by grouping the tokens of the Penn Treebank into 5, 10, or 50-length chunks and randomly permuting these chunks.
- In the *language* setup, we assess the effect of using real language. The input sequences here are taken from the Penn Treebank, and thus this setup further extends the *5gram*, *10gram*, and *50gram* datasets by adding the remaining structural properties of natural language.

We evaluate each model on a test set of uniformly sampled tokens from the 100 rarest words in the vocabulary. This evaluation setup ensures that, regardless of the data distribution the models were trained on, they are forced to generalize in order to perform well on the test set. For instance, in a test on data with a Zipfian token distribution, the model may do well by simply exploiting the training distribution (e.g., by ignoring the long tail of rare words).

2.4 RESULTS

We first observe that, in every case, the LSTM is able to perform the task perfectly (or nearly so), up to some input sequence length threshold. Once the input sequence length exceeds this threshold, performance drops rapidly.

HOW DOES THE TRAINING DATA DISTRIBUTION AFFECT PERFORMANCE ON THE MEMORIZATION TASK? Figure 1 compares memorization performance of an LSTM with 50 hidden units on various input sequence lengths when training on each of the datasets. Recall that the test set of only rare words is fixed for each length, regardless of the training data. When trained on the *uniform* dataset, the model is perfect up to length 10, but does no better than the random baseline with lengths above 10. Training on the *unigram* setting enables the model to memorize from longer sequences (up to 20), but it begins to fail with input sequences of length 40; evaluation accuracy quickly falls to 0.[7] Adding Markovian dependencies to the *unigram* dataset leads to small improvements, enabling the LSTM to successfully learn on inputs of up to length 40 (in the case of *5gram* and *10gram*) and inputs of up to length 60 (in the case of *50gram*). Lastly, training on *language* significantly improves model performance, and it is able to perfectly memorize with input sequences of up to 160 tokens before any significant degradation. These results clearly indicate that training on data with linguistic properties helps the LSTM learn

7 Manual inspection of the trained models reveals that they predict the most frequent words in the corpus. Since the evaluation set has only the 100 rarest types, performance (0% accuracy) is actually worse than in the *uniform* setting.

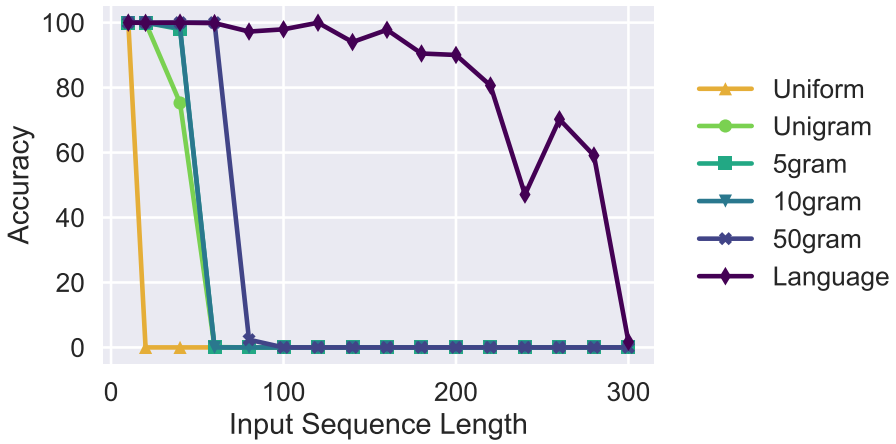

Figure 1: Test set accuracy of LSTMs with 50 hidden units trained on the *uniform*, *∗gram*, and *language* datasets with various input sequence lengths. *5gram* and *10gram* perform nearly identically, so the differences may not be apparent in the figure. *unigram* accuracy plateaus to 0, and *uniform* accuracy plateaus to ≈0.01% (random baseline). Best viewed in color.

the non-linguistic task of memorization, even though the test set has an adversarial non-linguistic distribution.

HOW DOES ADDING HIDDEN UNITS AFFECT MEMORIZATION PERFORMANCE? Figure 2 compares memorization performance on each dataset for LSTMs with 50, 100, and 200 hidden units. When training on the *uniform* dataset, increasing the number of LSTM hidden units (and thus also the embedding size) to 100 or 200 does not help it memorize longer sequences. Indeed, even at 400 and 800 we saw no improvement (not shown in Figure 2). When training on any of the other datasets, adding more hidden units eventually leads to perfect memorization for all tested input sequence lengths. We take these results as a suggestion that successful learning for this task requires sufficiently high capacity (dimensionality in the LSTM). The capacity need is diminished when the training data is linguistic, but LSTMs trained on the *uniform* set cannot learn the memorization task even given high capacity.

2.5 ANALYSIS

Throughout this section, we analyze an LSTM with 100 hidden units trained with the *language* setting with an input sequence length of 300. This setting is a somewhat closer simulation of current NLP models, since it is trained on real language and recalls perfectly with input sequence lengths of 300 (the most difficult setting tested).

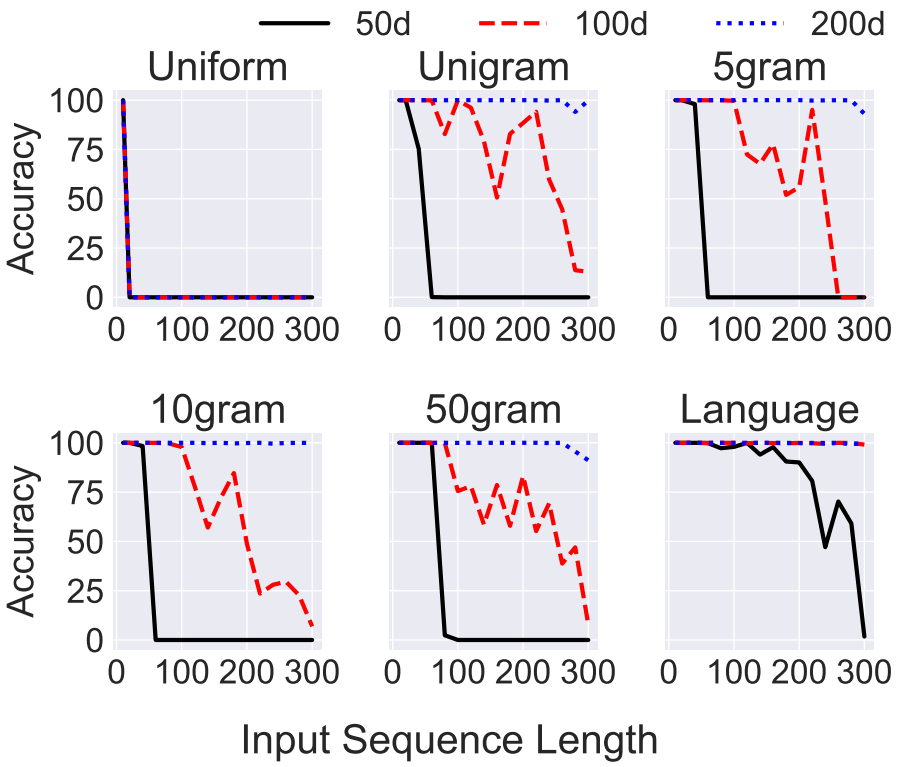

Figure 2: Test set accuracy of LSTMs with 50, 100 or 200 hidden units trained on each dataset with various input sequence lengths.

HOW DO LSTMS SOLVE THE MEMORIZATION TASK? A simple way to solve the memorization task is by counting. Since all of the input sequences are of equal length and the timestep to predict is constant throughout training and testing, a successful learner could maintain a counter from the start of the input to the position of the token to be predicted (the middle item). Then, it discards its previous cell state, consumes the label's vector, and maintains this new cell state until the end of the sequence (i.e., by setting its forget gate near 1 and its input gate near 0).

While LSTMs clearly have the expressive power needed to count and memorize, whether they can learn to do so from data is another matter. Past work has demonstrated that the LSTMs in an encoder-decoder machine translation model learn to increment and decrement a counter to generate translations of proper length (Shi, Knight, and Yuret, 2016) and that representations produced by auto-encoding LSTMs contain information about the input sequence length (Adi et al., 2017). Our experiments isolate the counting aspect from other linguistic properties of translation and autoencoding (which may indeed be correlated with counting), and also test this ability with an adversarial test distribution and much longer input sequences.

We adopt the method of Shi, Knight, and Yuret (2016) to investigate whether LSTMs solve the memorization task by learning to count.

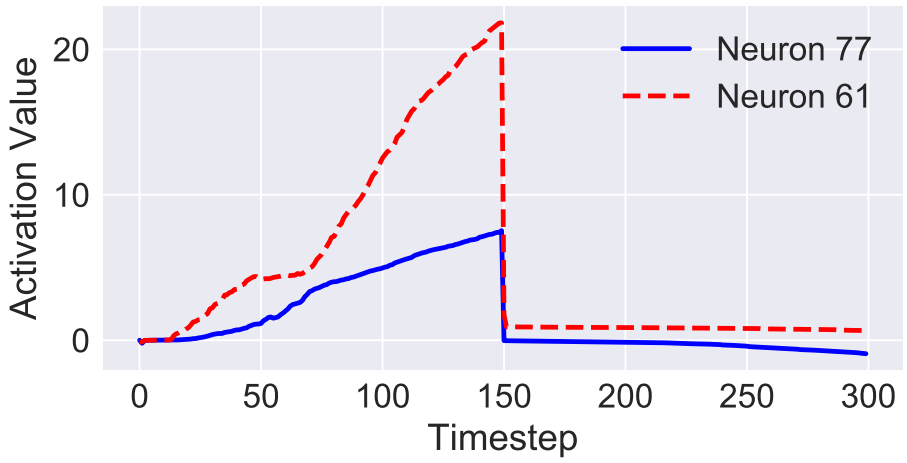

Figure 3: Activations of the neurons at indices 77 and 61 over time, showing counter-like behavior up to the target timestep to be remembered.

We identify the neurons that best predict timestep information by fitting a linear regression model to predict the number of inputs seen from the hidden unit activation. When evaluating on the test set, we observe that the LSTM cell state as a whole is very predictive of the timestep, with $R^2 = 0.998$.

While no single neuron perfectly records the timestep, several of them are strongly correlated. In our model instance, neuron 77 has the highest correlation ($R^2 = 0.919$), and neuron 61 is next ($R^2 = 0.901$). The activations of these neurons over time for a random correctly classified test input linearly increase up to the target token, after which the activations falls to nearly 0 (Figure 3).

ONE HYPOTHESIS FOR WHY LINGUISTIC DATA HELPS During training, the LSTM must: (1) determine what the objective is (here, "remember the middle token") and (2) adjust its weights to minimize loss. We observed that adding hidden units to LSTMs trained on *unigram* or *language* sets improves their ability to learn from long input sequences, but does not affect LSTMs trained on the *uniform* dataset. One explanation for this disparity is that LSTMs trained on *uniform* data are simply not learning what the task is—they do not realize that the label always matches the token in the middle of the input sequence, and thus they cannot properly optimize for the task, even with more hidden units. On the other hand, models trained on *unigram* or *language* can determine that the label is always the middle token, and can thus learn the task. Minimizing training loss ought to be easier with more parameters, so adding hidden units to LSTMs trained on data with linguistic attributes increases the length of input sequences that they can learn from.

But why might LSTMs trained on data with linguistic attributes be able to effectively learn the task for long input sequences, whereas LSTMs trained on the *uniform* dataset cannot? We conjecture that lin-

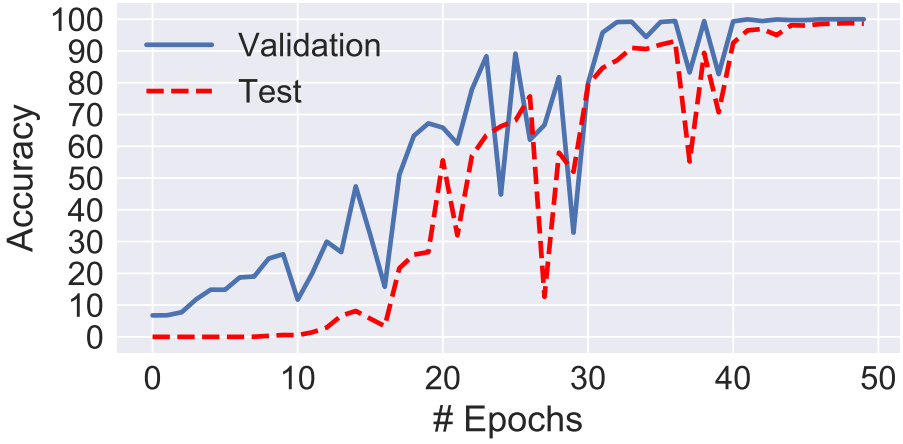

Figure 4: Model validation and test accuracy over time during training. Validation improves faster than test, indicating that the model exploits linguistic properties of the data during training.

guistic data offers more reasonable, if approximate, pathways to loss minimization, such as counting frequent words or phrases. In the *uniform* setting, the model has only one path to success: true memorization, and it cannot find an effective way to reduce the loss. In other words, linguistic structure and the patterns of language may provide additional signals that correlate with the label and facilitate learning the memorization task.

Figure 4 shows that models trained on the *unigram* and *language* datasets converge to high validation accuracy faster than high test accuracy. This suggests that models trained on data with linguistic attributes first learn to do well on the training data by exploiting the properties of language and not truly memorizing. Perhaps the model generalizes to actually recalling the target token later, as it refines itself with examples from the long tail of infrequent tokens.

Figure 4 may show this shift from exploiting linguistic properties to true memorization. The validation and test accuracy curves are quite synchronized from epoch 37 onward, indicating that the model's updates affect both sets identically. The model clearly learns a strategy that works well on both datasets, which strongly suggests that it has learned to memorize. In addition, when the model begins to move toward true memorization, we'd expect validation accuracy to momentarily falter as it moves away from the crutches of linguistic features—this may be the dip at around epoch 35 from perfect validation accuracy to around 95% accuracy.

2.6 RELATED WORK

To our knowledge, this work is the first to study how linguistic properties in training data affect the ability of LSTMs to learn a general, non-linguistic, sequence processing task.

Previous studies have sought to better understand the empirical capabilities of LSTMs trained on natural language data. Linzen, Dupoux, and Goldberg (2016) measured the ability of LSTMs to learn syntactic long range dependencies commonly found in language, and Gulordava et al. (2018) provide evidence that LSTMs can learn the hierarchical structure of language. Blevins, Levy, and Zettlemoyer (2018) show that the internal representations of LSTMs encode syntactic information, even when trained without explicit syntactic supervision.

Also related is the work of Weiss, Goldberg, and Yahav (2018), who demonstrate that LSTMs are able to count infinitely, since their cell states are unbounded, while GRUs cannot count infinitely since the activations are constrained to a finite range. One avenue of future work could compare the performance of LSTMs and GRUs on the memorization task.

Past studies have also investigated what information RNNs encode by directly examining hidden unit activations (Karpathy, Johnson, and Fei-Fei, 2016; Li et al., 2016; Shi, Knight, and Yuret, 2016, among others) and by training an auxiliary classifier to predict various properties of interest from hidden state vectors (Shi, Padhi, and Knight, 2016; Adi et al., 2017; Belinkov et al., 2017b, among others).

2.7 CONCLUSION

In this chapter, we examine how linguistic attributes in training data can affect an LSTM's ability to learn a simple memorization task. We find that LSTMs trained on uniformly sampled data are only able to learn the task with the sequence length of 10, whereas LSTMs trained with language data are able to learn on sequences of up to 300 tokens.

We further investigate how the LSTM learns to solve the task, and find that it uses a subset of its hidden units to track timestep information. It is still an open question why LSTMs trained on linguistic data are able to learn the task whereas LSTMs trained on uniformly sampled data cannot; based on our observations, we hypothesize that the additional patterns and structure in language-based data may provide the model with approximate paths of loss minimization, and improve LSTM trainability as a result.

3

LINGUISTIC KNOWLEDGE AND TRANSFERABILITY OF CONTEXTUAL WORD REPRESENTATIONS

3.1 INTRODUCTION

Pretrained word representations (Mikolov et al., 2013; Pennington, Socher, and Manning, 2014) are a key component of state-of-the-art neural NLP models. Traditionally, these word vectors are static—a single vector is assigned to each word. Recent work has explored *contextual* word representations (henceforth: CWRS), which assign each word a vector that is a function of the entire input sequence; this enables them to model the use of words in context. CWRS are typically the outputs of a neural network (which we call a *contextualizer*) trained on tasks with large datasets, such as machine translation (McCann et al., 2017) and language modeling (Peters et al., 2018a). CWRS are extraordinarily effective—using them in place of traditional static word vectors within the latest models leads to large gains across a variety of NLP tasks.

The broad success of CWRS indicates that they encode useful, transferable features of language. However, their *linguistic knowledge* and *transferability* are not yet well understood.

Recent work has explored the linguistic knowledge captured by language models and neural machine translation systems, but these studies often focus on a single phenomenon, e.g., knowledge of hierarchical syntax (Blevins, Levy, and Zettlemoyer, 2018) or morphology (Belinkov et al., 2017b). In this chapter, we extend prior work by studying CWRS with a diverse set of seventeen probing tasks designed to assess a wide array of phenomena, such as coreference, knowledge of semantic relations, and entity information, among others. The result is a broader view of the linguistic knowledge encoded within CWRS.[1]

With respect to transferability, pretraining contextualizers on the language modeling task has had the most empirical success, but we can also consider pretraining contextualizers with other supervised objectives and probing their linguistic knowledge. This chapter also examine how the pretraining task affects the linguistic knowledge learned, considering twelve pretraining tasks and assessing transferability to nine target tasks.

Better understanding the linguistic knowledge and transferability of CWRS is necessary for their principled enhancement through new

1 The work described in this chapter was done with Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith, and was published as Liu et al. (2019).

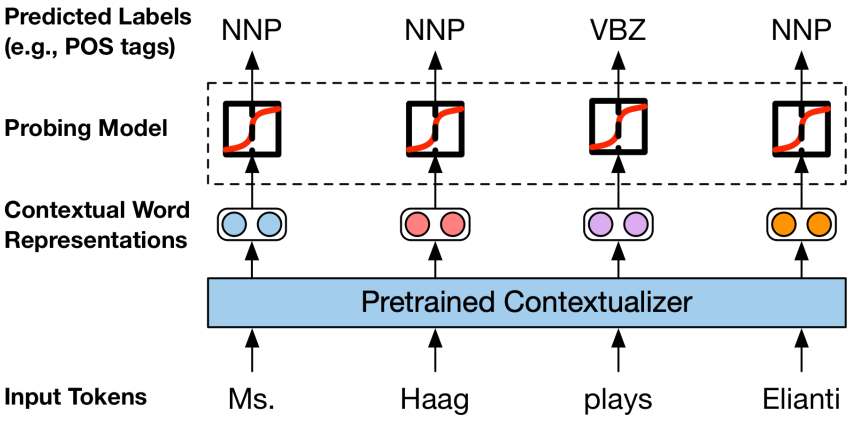

Figure 5: An illustration of the probing model setup used to study the linguistic knowledge within contextual word representations.

encoder architectures and pretraining tasks that build upon their strengths or alleviate their weaknesses (Linzen, 2019). This chapter asks and answers:

1. What features of language do these vectors capture, and what do they miss? (§3.4)
2. How and why does transferability vary across representation layers in contextualizers? (§3.5)
3. How does the choice of pretraining task affect the vectors' learned linguistic knowledge and transferability? (§3.6)

We use probing models[2] (Shi, Padhi, and Knight, 2016; Adi et al., 2017; Hupkes, Veldhoen, and Zuidema, 2018; Belinkov and Glass, 2019) to analyze the linguistic information within CWRs. Concretely, we generate features for words from pretrained contextualizers and train a model to make predictions from those features alone (Figure 5). If a simple model can be trained to predict linguistic information about a word (e.g., its part-of-speech tag) or a pair of words (e.g., their semantic relation) from the CWR(s) alone, we can reasonably conclude that the CWR(s) encode this information.

Our analysis reveals interesting insights such as:

1. Linear models trained on top of frozen CWRs are competitive with state-of-the-art task-specific models in many cases, but fail on tasks requiring fine-grained linguistic knowledge. In these cases, we show that *task-trained contextual features* greatly help with encoding the requisite knowledge.
2. The first layer output of long short-term memory (LSTM) recurrent neural networks is consistently the most transferable, whereas it is the middle layers for transformers.

2 Sometimes called auxiliary or diagnostic classifiers.

3. Higher layers in LSTMs are more task-specific (and thus less general), while the transformer layers do not exhibit this same monotonic increase in task-specificity.
4. Language model pretraining yields representations that are more transferable *in general* than eleven other candidate pretraining tasks, though pretraining on related tasks yields the strongest results for individual end tasks.

3.2 PROBING TASKS

We construct a suite of seventeen diverse English probing tasks and use it to better understand the linguistic knowledge contained within CWRs. In contrast to previous studies that analyze the properties and task performance of sentence embeddings Adi et al., 2017; Conneau et al., 2018, *inter alia*, we specifically focus on understanding the CWRs of individual or pairs of words. We release this analysis toolkit to support future work in probing the contents of representations.[3] See Appendix A.1 for details about task setup.

3.2.1 *Token Labeling*

The majority of past work in probing the internal representations of neural models has examined various token labeling tasks, where a decision is made independently for each token (Belinkov et al., 2017b; Belinkov et al., 2017a; Blevins, Levy, and Zettlemoyer, 2018, *inter alia*). We synthesize these disparate studies and build upon them by proposing additional probing tasks.

The **part-of-speech tagging (POS)** task assesses whether CWRs capture basic syntax. We experiment with two standard datasets: the Penn Treebank (PTB; Marcus, Marcinkiewicz, and Santorini, 1993) and the Universal Dependencies English Web Treebank (UD-EWT; Silveira et al., 2014).

The **CCG supertagging (CCG)** task assesses the vectors' fine-grained information about the syntactic roles of words in context. It is considered "almost parsing" (Bangalore and Joshi, 1999), since a sequence of supertags maps a sentence to a small set of possible parses. We use CCGbank (Hockenmaier and Steedman, 2007), a conversion of the PTB into CCG derivations.

The **syntactic constituency ancestor tagging** tasks are designed to probe the vectors' knowledge of hierarchical syntax. For a given word, the probing model is trained to predict the constituent label of its parent **(Parent)**, grandparent **(GParent)**, or great-grandparent **(GGParent)** in the phrase-structure tree (from the PTB).

3 http://nelsonliu.me/papers/contextual-repr-analysis

(1) I was booked **for**/DURATION 2 nights **at**/LOCUS this hotel **in**/TIME Oct 2007 .

(2) I went **to**/GOAL ohm **after**/EXPLANATION⇝TIME reading some **of**/QUANTITY⇝WHOLE the reviews .

(3) It was very upsetting to see this kind **of**/SPECIES behavior especially **in_front_of**/LOCUS **my**/SOCIALREL⇝GESTALT four year_old .

Figure 6: Annotated sentences from the STREUSLE 4.0 corpus, used in the preposition supersense disambiguation task. Prepositions are marked by boldface, immediately followed by their labeled function. If applicable, ⇝ precedes the preposition's labeled role. Figure reproduced from Schneider et al. (2018).

In the **semantic tagging** task (**ST**), tokens are assigned labels that reflect their semantic role in context. These semantic tags assess lexical semantics, and they abstract over redundant POS distinctions and disambiguate useful cases within POS tags. We use the dataset of Bjerva, Plank, and Bos (2016); the tagset has since been developed as part of the Parallel Meaning Bank (Abzianidze et al., 2017).

Preposition supersense disambiguation is the task of classifying a preposition's lexical semantic contribution (the function; **PS-fxn**) and the semantic role or relation it mediates (the role; **PS-role**). This task is a specialized kind of word sense disambiguation, and examines one facet of lexical semantic knowledge. In contrast to the tagging tasks above, the model is trained and evaluated on single-token prepositions (rather than making a decision for every token in a sequence). We use the STREUSLE 4.0 corpus (Schneider et al., 2018); example sentences appear in Figure 6.

The **event factuality (EF)** task involves labeling phrases with the factuality of the events they describe (Saurí and Pustejovsky, 2009; Saurí and Pustejovsky, 2012; Marneffe, Manning, and Potts, 2012). For instance, in the following example reproduced from Rudinger, White, and Durme (2018), "(1a) conveys that the leaving didn't happen, while the superficially similar (1b) does not".

(1) a. Jo didn't remember to **leave**.
b. Jo didn't remember **leaving**.

We use the Universal Decompositional Semantics It Happened v2 dataset (Rudinger, White, and Durme, 2018), and the model is trained to predict a (non)factuality value in the range $[-3, 3]$. Unlike the tagging tasks above, this task is treated as a regression problem, where a prediction is made only for tokens corresponding to events (rather than every token in a sequence). Performance is measured using Pear-

son correlation (r); we report ($r \times 100$) so metrics for all tasks fall between 0 and 100.

3.2.2 *Segmentation*

Several of our probing tasks involve segmentation using BIO or IO tags. Here the model is trained to predict labels from *only* a single word's CWR.

Syntactic chunking (Chunk) tests whether CWRs contain notions of spans and boundaries; the task is to segment text into shallow constituent chunks. We use the CoNLL 2000 shared task dataset (Tjong Kim Sang and Buchholz, 2000).

Named entity recognition (NER) examines whether CWRs encode information about entity types. We use the CoNLL 2003 shared task dataset (Tjong Kim Sang and De Meulder, 2003).

Grammatical error detection (GED) is the task of identifying tokens which need to be edited in order to produce a grammatically correct sentence. Given that CWRs are extracted from models trained on large amounts of grammatical text, this task assesses whether embeddings encode features that indicate anomalies in their input (in this case, ungrammaticality). We use the First Certificate in English (Yannakoudakis, Briscoe, and Medlock, 2011) dataset, converted into sequence-labeling format by Rei and Yannakoudakis (2016).

The **conjunct identification (Conj)** task challenges the model to identify the tokens that comprise the conjuncts in a coordination construction. Doing so requires highly specific syntactic knowledge. The data comes from the coordination-annotated PTB of Ficler and Goldberg (2016).

3.2.3 *Pairwise Relations*

We also design probing tasks that examine whether relationships *between* words are encoded in CWRs (Figure 7). In these tasks, given a word pair w_1, w_2, we input $[w_1, w_2, w_1 \odot w_2]$ into the probing model; it is trained to predict information about the relation between the tokens (Belinkov, 2018).

We distinguish between **arc prediction** and **arc classification** tasks. Arc prediction is a binary classification task, where the model is trained to identify *whether* a relation exists between two tokens. Arc classification is a multiclass classification task, where the model is provided with two tokens that are linked via some relationship and trained to identify *how* they are related.

For example, in the **syntactic dependency arc prediction** task, the model is given the representations of two tokens (w_a, w_b) and trained to predict whether the sentence's syntactic dependency parse contains a dependency arc with w_a as the head and w_b as the modifier.

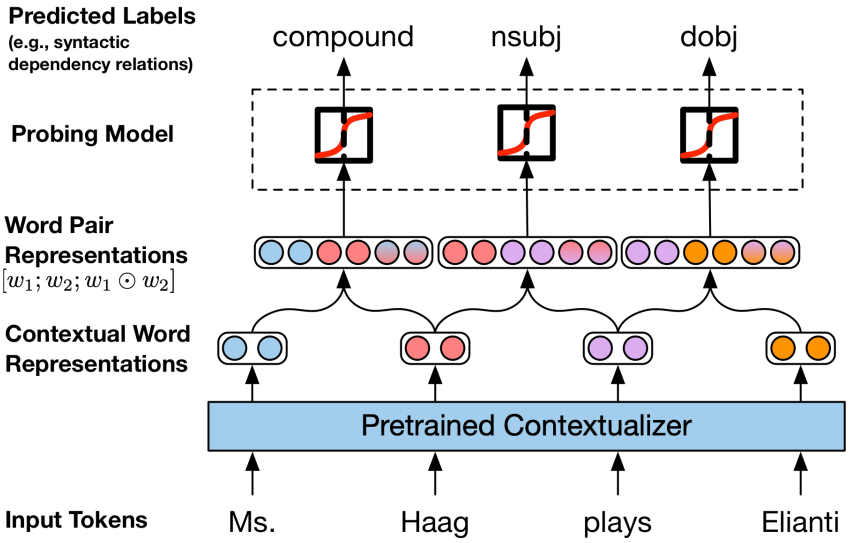

Figure 7: An illustration of the pairwise probing model setup used to study the linguistic knowledge within pairs of contextual word representations.

The **syntactic dependency arc classification** task presents the model with the representations of two tokens (w_{head}, w_{mod}), where w_{mod} is the modifier of w_{head}, and the model is trained to predict the type of syntactic relation that link them (the label on that dependency arc). We use the PTB (converted to UD) and the UD-EWT.

Similarly, **semantic dependency arc prediction** trains the model to predict whether two tokens are connected by a semantic dependency arc, while the **semantic dependency arc classification** task trains models to classify the semantic relations between tokens. We use the dataset from the SemEval 2015 shared task (Oepen et al., 2015) with the DELPH-IN MRS-Derived Semantic Dependencies (DM) target representation.

The syntactic and semantic dependency arc prediction and classification tasks are closely related to state-of-the-art models for semantic and syntactic dependency parsing, which score pairs of CWRs to make head attachment and arc labeling decisions (Dozat and Manning, 2016; Dozat and Manning, 2018).

To generate negative examples for the dependency arc prediction tasks, we take each positive example (w_{head}, w_{mod}) and generate a new negative example (w_{rand}, w_{mod}). w_{rand} is a random token in the sentence that is not the head of w_{mod}. Thus, the datasets used in these tasks are balanced.

We also consider a **coreference arc prediction** task, where the model is trained to predict whether two entities corefer from their CWRs. We use the dataset from the CoNLL 2012 shared task (Pradhan et al., 2012). To generate negative examples, we follow a similar procedure as the dependency arc prediction tasks: given a positive exam-

ple (w_a, w_b), where w_b occurs after w_a and the two tokens share a coreference cluster, we create a negative example $(w_{\textit{random\_entity}}, w_b)$, where $w_{\textit{random\_entity}}$ is a token that occurs before w_b and belongs to a different coreference cluster.

3.3 MODELS

PROBING MODEL We use a linear model as our probing model; limiting its capacity enables us to focus on what information can be *easily* extracted from CWRs. See Appendix A.2 for probing model training hyperparameters and other details.

CONTEXTUALIZERS We study six publicly-available models for contextualized word representation in English.

ELMo (Peters et al., 2018a) concatenates the output of two contextualizers independently trained on the bidirectional language modeling (biLM) task. **ELMo (original)** uses a 2-layer LSTM for contextualization. We also study two variations from Peters et al. (2018b): **ELMo (4-layer)** uses a 4-layer LSTM, and **ELMo (transformer)** uses a 6-layer transformer (Vaswani et al., 2017). Each of these models is trained on 800M tokens of sentence-shuffled newswire text (the 1 Billion Word Benchmark; Chelba et al., 2014).

The **OpenAI transformer** (Radford et al., 2018) is a left-to-right 12-layer transformer language model trained on 800M tokens of contiguous text from over 7,000 unique unpublished books (BookCorpus; Zhu et al., 2015).

BERT (Devlin et al., 2018) uses a bidirectional transformer jointly trained on a masked language modeling task and a next sentence prediction task. The model is trained on BookCorpus and the English Wikipedia, a total of approximately 3300M tokens. We study **BERT (base, cased)**, which uses a 12-layer transformer, and **BERT (large, cased)**, which uses a 24-layer transformer.

3.4 PRETRAINED CONTEXTUALIZER COMPARISON

To better understand the linguistic knowledge captured by pretrained contextualizers, we analyze each of their layers with our set of probing tasks. These contextualizers differ in many respects, and it is outside the scope of this work to control for all differences between them. We focus on probing the models that are available to us, leaving a more systematic comparison of training regimes and model architectures to future work.

3.4.1 *Experimental Setup*

Our probing models are trained on the representations produced by the individual layers of each contextualizer. We also compare to a linear probing model trained on noncontextual vectors (300-dimensional GloVe trained on the cased Common Crawl; Pennington, Socher, and Manning, 2014) to assess the gains from contextualization.

3.4.2 *Results and Discussion*

Table 1 compares each contextualizer's best-performing probing model with the GloVe baseline and the previous state of the art for the task (excluding methods that use pretrained CWRs).[4,5]

With just a linear model, we can readily extract much of the information needed for high performance on various NLP tasks. In all cases, CWRs perform significantly better than the noncontextual baseline. Indeed, we often see probing models rivaling or exceeding the performance of (often carefully tuned and task-specific) state-of-the-art models. In particular, the linear probing model surpasses the published state of the art for grammatical error detection and preposition supersense identification (both role and function).

Comparing the ELMo-based contextualizers, we see that ELMo (4-layer) and ELMo (original) are essentially even, though both recurrent models outperform ELMo (transformer). We also see that the OpenAI transformer significantly underperforms the ELMo models and BERT. Given that it is also the only model trained in a unidirectional (left-to-right) fashion, this reaffirms that bidirectionality is a crucial component for the highest-quality contextualizers (Devlin et al., 2018). In addition, the OpenAI transformer is the only model trained on lowercased text, which hinders its performance on tasks like NER. BERT significantly improves over the ELMo and OpenAI models.

Our probing task results indicate that current methods for CWR do not capture much transferable information about entities and coreference phenomena in their input (e.g., the NER results in Table 1 and the coreference arc prediction results in Appendix A.4). To alleviate this weakness, future work could augment pretrained contextualizers with explicit entity representations (Ji et al., 2017; Yang et al., 2017; Bosselut et al., 2017).

PROBING FAILURES While probing models are at or near state-of-the-art performance across a number of tasks, they also do not perform as well on several others, including NER, grammatical error

4 See Appendix A.3 for references to the previous state of the art (without pretraining).
5 For brevity, in this section we omit probing tasks that cannot be compared to prior work. See Appendix A.4 for pretrained contextualizer performance for all layers and all tasks.

Pretrained Representation			POS						Supersense ID		
	Avg.	CCG	PTB	EWT	Chunk	NER	ST	GED	PS-Role	PS-Fxn	EF
ELMo (original) best layer	81.58	93.31	97.26	95.61	90.04	82.85	93.82	29.37	75.44	84.87	73.20
ELMo (4-layer) best layer	81.58	93.81	**97.31**	95.60	89.78	82.06	**94.18**	29.24	74.78	85.96	73.03
ELMo (transformer) best layer	80.97	92.68	97.09	95.13	93.06	81.21	93.78	30.80	72.81	82.24	70.88
OpenAI transformer best layer	75.01	82.69	93.82	91.28	86.06	58.14	87.81	33.10	66.23	76.97	74.03
BERT (base, cased) best layer	84.09	93.67	96.95	95.21	92.64	82.71	93.72	43.30	**79.61**	87.94	75.11
BERT (large, cased) best layer	**85.07**	**94.28**	96.73	**95.80**	**93.64**	**84.44**	93.83	**46.46**	79.17	**90.13**	**76.25**
GloVe (840B.300d)	59.94	71.58	90.49	83.93	62.28	53.22	80.92	14.94	40.79	51.54	49.70
Previous state of the art (without pretraining)	83.44	94.7	97.96	95.82	95.77	91.38	95.15	39.83	66.89	78.29	77.10

Table 1: Performance of the best layerwise linear probing model for each contextualizer compared against a GloVe-based linear probing baseline and the previous state of the art. The best contextualizer for each task is bolded. Results for all layers on all tasks, and papers describing the prior state of the art, are given in Appendix A.4.

Probing Model	NER	GED	Conj	GGParent
Linear	82.85	29.37	38.72	67.50
MLP (1024d)	87.19	47.45	55.09	78.80
LSTM (200d) + Linear	**88.08**	**48.90**	**78.21**	**84.96**
BiLSTM (512d) + MLP (1024d)	90.05	48.34	87.07	90.38

Table 2: Comparison of different probing models trained on ELMo (original); best-performing probing model is bolded. Results for each probing model are from the highest-performing contextualizer layer. Enabling probing models to learn task-specific contextual features (with LSTMs) yields outsized benefits in tasks requiring highly specific information.

detection, and conjunct identification. This may occur because (1) the CWR simply does not encode the pertinent information or any predictive correlates, or (2) the probing model does not have the capacity necessary to extract the information or predictive correlates from the vector. In the former case, learning *task-specific contextual features* might be necessary for encoding the requisite task-specific information into the CWRs. Learning task-specific contextual features with a contextual probing model also helps with (2), but we would expect the results to be comparable to increasing the probing model's capacity.

To better understand the failures of our probing model, we experiment with (1) a contextual probing model that uses a task-trained LSTM (unidirectional, 200 hidden units) before the linear output layer (thus adding task-specific contextualization) or (2) replacing the linear probing model with a multilayer perceptron (MLP; adding more parameters to the probing model: a single 1024d hidden layer activated by ReLU). These alternate probing models have nearly the same number of parameters (LSTM + linear has slightly fewer).

We also compare to a full-featured model to estimate an upper bound on performance for our probing setup. In this model, the CWRs are inputs to a 2-layer BiLSTM with 512 hidden units, and the output is fed into a MLP with a single 1024-dimensional hidden layer activated by a ReLU to predict a label. A similar model, augmented with a conditional random field (CRF; Lafferty, McCallum, and Pereira, 2001), achieved state-of-the-art results on the CoNLL 2003 NER dataset (Peters et al., 2018a). We remove the CRF, since other probing models have no global context.

For this experiment, we focus on the ELMo (original) pretrained contextualizer. Table 2 presents the performance of the best layer within each alternative probing model on the two tasks with the

largest gap between the linear probing model and state-of-the-art methods: NER and grammatical error detection. We also include great-grandparent prediction and conjunct identification, two tasks that require highly specific syntactic knowledge. In all cases, we see that adding more parameters (either by replacing the linear model with a MLP, or using a contextual probing model) leads to significant gains over the linear probing model. On NER and grammatical error detection, we observe very similar performance between the MLP and LSTM + Linear models—this indicates that the probing model simply needed more capacity to extract the necessary information from the CWRs. On conjunct identification and great-grandparent prediction, two tasks that probe syntactic knowledge unlikely to be encoded in CWRs, adding parameters as a task-trained component of our probing model leads to large gains over simply adding parameters to the probing model. This indicates that the pretrained contextualizers do not capture the information necessary for the task, since such information is learnable by a task-specific contextualizer.

This analysis also reveals insights about contextualizer fine-tuning, which seeks to specialize the CWRs for an end task (Howard and Ruder, 2018; Radford et al., 2018; Devlin et al., 2018). Our results confirm that task-trained contextualization is important when the end task requires specific information that may not be captured by the pretraining task (§3.4). However, such end-task–specific contextualization can come from either fine-tuning CWRs or using fixed output features as inputs to a task-trained contextualizer; Peters, Ruder, and Smith (2019) begin to explore when each approach should be applied.

3.5 ANALYZING LAYERWISE TRANSFERABILITY

We quantify the transferability of CWRs by how well they can do on the range of probing tasks—representations that are more transferable will perform better than alternatives across tasks. When analyzing the representations produced by each layer of pretrained contextualizers, we observe marked patterns in layerwise transferability (Figure 8). The first layer of contextualization in recurrent models (original and 4-layer ELMo) is consistently the most transferable, even outperforming a scalar mix of layers on most tasks (see Appendix A.4 for scalar mix results). Schuster et al. (2019) see the same trend in English dependency parsing. By contrast, transformer-based contextualizers have no single most-transferable layer; the best performing layer for each task varies, and is usually near the middle. Accordingly, a scalar mix of transformer layers outperforms the best individual layer on most tasks (see Appendix A.4).

Pretraining encourages the model to encode pretraining-task–specific information; they learn transferable features incidentally. We hypothesize that this is an inherent trade-off—since these models used fixed-

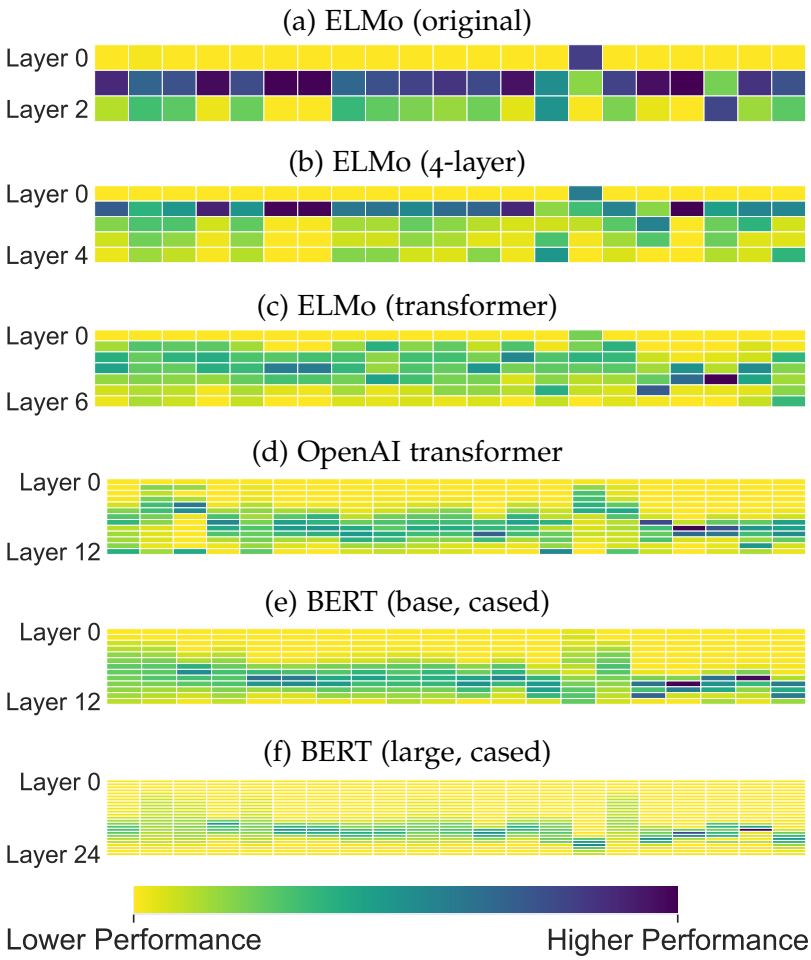

Figure 8: A visualization of layerwise patterns in task performance. Each column represents a probing task, and each row represents a contextualizer layer.

sized vector representations, task-specificity comes at the cost of generality and transferability. To investigate the task-specificity of the representations generated by each contextualizer layer, we assess how informative each layer of representation is for the pretraining task, essentially treating it as a probe.

3.5.1 *Experimental Setup*

We focus on the ELMo-based models, since the authors have released code for training their contextualizers. Furthermore, the ELMo-based models facilitate a controlled comparison—they only differ in the contextualizer architecture used.

We evaluate how well CWR features perform the pretraining task—bidirectional language modeling. Specifically, we take the pretrained representations for each layer and relearn the language model softmax classifiers used to predict the next and previous token. The

ELMo models are trained on the Billion Word Benchmark, so we retrain the softmax classifier on similar data to mitigate any possible effects from domain shift. We split the held-out portion of the Billion Word Benchmark into train (80%, 6.2M tokens) and evaluation (20%, 1.6M tokens) sets and use this data to retrain and evaluate the softmax classifiers. We expect that biLM perplexity will be lower when training the softmax classifiers on representations from layers that capture more information about the pretraining task.

3.5.2 *Results and Discussion*

Figure 9 presents the performance of softmax classifiers trained to perform the bidirectional language modeling task, given just the CWRs as input. We notice that higher layers in recurrent models consistently achieve lower perplexities. Interestingly, we see that layers 1 and 2 in the 4-layer ELMo model have very similar performance—this warrants further exploration. On the other hand, the layers of the ELMo (transformer) model do not exhibit such a monotonic increase. While the topmost layer is best (which we expected, since this is the vector originally fed into a softmax classifier during pretraining), the middle layers show varying performance. Across all models, the representations that are better-suited for language modeling are also those that exhibit worse probing task performance (Figure 8), indicating that contextualizer layers trade off between encoding general and task-specific features.

These results also reveal a difference in the layerwise behavior of LSTMs and transformers; moving up the LSTM layers yields more task-specific representations, but the same does not hold for transformers. Better understanding the differences between transformers and LSTMs is an active area of research (Chen et al., 2018; Tang et al., 2018), and we leave further exploration of these observations to future work.

These observations motivate the gradual unfreezing method of Howard and Ruder (2018), where the model layers are progressively unfrozen (starting from the final layer) during the fine-tuning process. Given our observation that higher-level LSTM layers are less general (and more pretraining task-specific), they likely have to be fine-tuned a bit more in order to make them appropriately task specific. Meanwhile, the base layer of the LSTM already learns highly transferable features, and may not benefit from fine-tuning.

3.6 TRANSFERRING BETWEEN TASKS

Successful pretrained contextualizers have used self-supervised tasks such as bidirectional language modeling (Peters et al., 2018a) and next sentence prediction (Devlin et al., 2018), which enable the use of

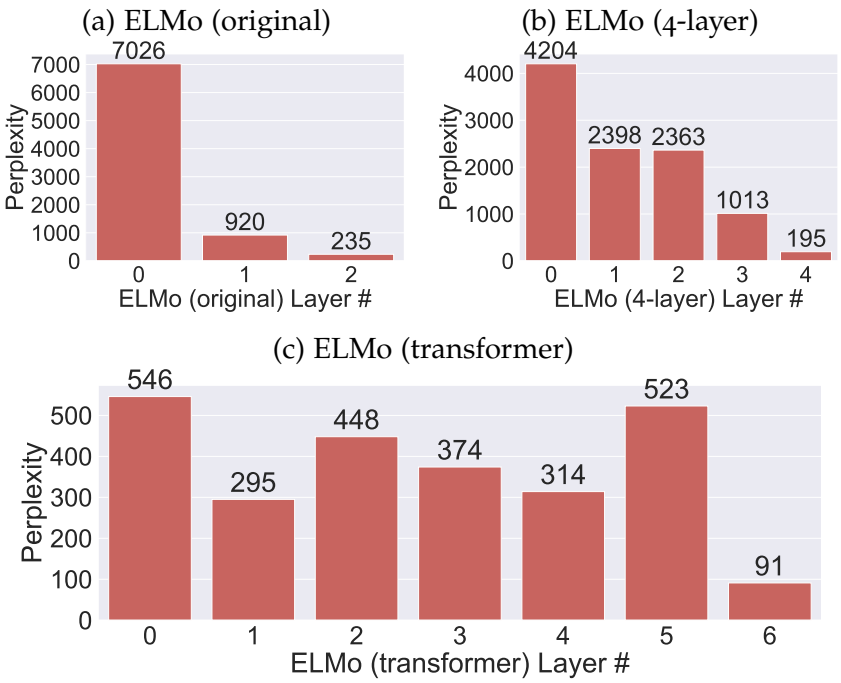

Figure 9: Bidirectional language modeling as a probe: average of forward and backward perplexity (lower is better) of each ELMo contextualizer layer. We see a monotonic decrease in BiLM perplexity when trained on the outputs of higher LSTM layers, but transformer layers do not exhibit the same pattern.

large, unannotated text corpora. However, contextualizers can also be pretrained on explicitly supervised objectives, as done in pretrained *sentence* embedding methods (Conneau et al., 2017). To better understand how the choice of pretraining task affects the linguistic knowledge within and transferability of CWRs, we compare pretraining on a range of different explicitly-supervised tasks with bidirectional language model pretraining.

3.6.1 *Experimental Setup*

To ensure a controlled comparison of different pretraining tasks, we fix the contextualizer's architecture and pretraining dataset. All of our contextualizers use the ELMo (original) architecture, and the training data from each of the pretraining tasks is taken from the PTB. Each of the (identical) models thus see the same tokens, but the supervision signal differs.[6] We compare to (1) a noncontextual baseline (GloVe) to assess the effect of contextualization, (2) a randomly-initialized, untrained ELMo (original) baseline to measure the effect of pretraining, and (3) the ELMo (original) model pretrained on the Billion Word

6 We omit the OpenAI transformer and BERT from this comparison, since code for pretraining these contextualizers is not publicly available.

Benchmark to examine the effect of training the bidirectional language model on more data.

3.6.2 *Results and Discussion*

Table 3 presents the average target task performance of each layer in contextualizers pretrained on twelve different tasks (biLM and the eleven tasks from §3.2 with PTB annotations). Bidirectional language modeling pretraining is the most effective on average. However, the settings that achieve the highest performance for individual target tasks often involve transferring between related tasks (not shown in Table 3; see Appendix A.5). For example, when probing CWRs on the syntactic dependency arc classification (EWT) task, we see the largest gains from pretraining on the task itself, but with a different dataset (PTB). However, pretraining on syntactic dependency arc prediction (PTB), CCG supertagging, chunking, the ancestor prediction tasks, and semantic dependency arc classification all give better performance than bidirectional language model pretraining.

Although related task transfer is beneficial, we naturally see stronger results from training on more data (the ELMo original BiLM trained on the Billion Word Benchmark). This indicates that the transferability of pretrained CWRs relies on pretraining on large corpora, emphasizing the utility and importance of self-supervised pretraining.

Furthermore, layer 0 of the BiLM is the highest-performing single layer among PTB-pretrained contextualizers. This observation suggests that lexical information is the source of the language model's initial generalizability, since layer 0 is the output of a character-level convolutional neural network with no token-level contextual information.

3.7 RELATED WORK

Methodologically, our work is most similar to Shi, Padhi, and Knight (2016), Adi et al. (2017), and Hupkes, Veldhoen, and Zuidema (2018), who use the internal representations of neural models to predict properties of interest. Conneau et al. (2018) construct probing tasks to study the linguistic properties of sentence embedding methods. We focus on contextual word representations, which have achieved state-of-the-art results on a variety of tasks, and examine a broader range of linguistic knowledge.

In contemporaneous work, Tenney et al. (2019) evaluate CoVe (McCann et al., 2017), ELMo (Peters et al., 2018a), the OpenAI Transformer (Radford et al., 2018), and BERT (Devlin et al., 2018) on a variety of sub-sentence linguistic analysis tasks. Their results also suggest that the aforementioned pretrained models for contextualized word representation encode stronger notions of syntax than higher-level se-

Pretraining Task	Layer Average Target Task Performance			
	0	1	2	Mix
CCG	56.70	64.45	63.71	66.06
Chunk	54.27	62.69	63.25	63.96
POS	56.21	63.86	64.15	65.13
Parent	54.57	62.46	61.67	64.31
GParent	55.50	62.94	62.91	64.96
GGParent	54.83	61.10	59.84	63.81
Syn. Arc Prediction	53.63	59.94	58.62	62.43
Syn. Arc Classification	56.15	64.41	63.60	66.07
Sem. Arc Prediction	53.19	54.69	53.04	59.84
Sem. Arc Classification	56.28	62.41	61.47	64.67
Conj	50.24	49.93	48.42	56.92
BiLM	66.53	65.91	65.82	66.49
GloVe (840B.300d)	60.55			
Untrained ELMo (original)	52.14	39.26	39.39	54.42
ELMo (original) (BiLM on 1B Benchmark)	64.40	79.05	77.72	78.90

Table 3: Performance (averaged across target tasks) of contextualizers pretrained on a variety of tasks.

mantics. They also find that using a scalar mix of output layers is particularly effective in deep transformer-based models, aligned with our own probing results and our observation that transformers tend to encode transferable features in their intermediate layers. Furthermore, they find that ELMo's performance cannot be explained by a model with access to only local context, indicating that ELMo encodes linguistic features from distant tokens.

Several other papers have examined how architecture design and choice of pretraining task affect the quality of learned CWRs. Peters et al. (2018b) study how the choice of neural architecture influences the end-task performance and qualitative properties of CWRs derived from bidirectional language models (ELMo). Bowman et al. (2018) compare a variety of pretraining tasks and explore the the impact of multitask learning.

Prior work has employed a variety of other methods to study the learned representations in neural models, such as directly examining the activations of individual neurons (Karpathy, Johnson, and Fei-Fei, 2016; Li et al., 2016; Shi, Knight, and Yuret, 2016, *inter alia*), ablating

components of the model and dataset (Kuncoro et al., 2017; Gaddy, Stern, and Klein, 2018; Khandelwal et al., 2018), or interpreting attention mechanisms (Bahdanau, Cho, and Bengio, 2015); see Belinkov and Glass (2019) for a recent survey. One particularly relevant line of work involves the construction of synthetic tasks that a model can only solve if it captures a particular phenomenon (Linzen, Dupoux, and Goldberg, 2016; Jumelet and Hupkes, 2018; Wilcox et al., 2018; Futrell and Levy, 2019, *inter alia*). Zhang and Bowman (2018) compare the syntactic knowledge of language models and neural machine translation systems. We widen the range of pretraining tasks and target probing model tasks to gain a more complete picture. We also focus on a stronger contextualizer architecture, ELMo (original), that has produced state-of-the-art results.

Several studies have sought to intrinsically evaluate noncontextual word representations with word similarity tasks, such as analogies (Mikolov et al., 2013). These methods differ from our approach in that they require no extra parameters and directly assess the vectors, while our probing models must be trained. In this regard, our method is similar to QVEC (Tsvetkov et al., 2015).

3.8 CONCLUSION

We study the linguistic knowledge and transferability of contextualized word representations with a suite of seventeen diverse probing tasks. The features generated by pretrained contextualizers are sufficient for high performance on a broad set of tasks. For tasks that require specific information not captured by the contextual word representation, we show that learning task-specific contextual features helps encode the requisite knowledge. In addition, our analysis of patterns in the transferability of contextualizer layers shows that the lowest layer of LSTMs encodes the most transferable features, while transformers' middle layers are most transferable. We find that higher layers in LSTMs are more task-specific (and thus less general), while transformer layers do not exhibit this same monotonic increase in task-specificity. Prior work has suggested that higher-level contextualizer layers may be expressly encoding higher-level semantic information. Instead, it seems likely that certain high-level semantic phenomena are incidentally useful for the contextualizer's pretraining task, leading to their presence in higher layers. Lastly, we find that bidirectional language model pretraining yields representations that are more transferable *in general* than eleven other candidate pretraining tasks.

4

AFTERWORD

This thesis studied the implicit linguistic knowledge of end-to-end neural network-based natural language processsing models. Chapter 2 demonstrates that recurrent neural networks, specifically long short-term memory (LSTM) units, are sensitive to linguistic features in their input; these linguistic features aid model generalization. Chapter 3 describes an analysis focused on characterizing what specific linguistic features and phenomena are implicitly learned by large-scale neural language models. We show that language models trained in an unsupervised fashion on massive datasets learn representations that are sufficient for high performance on a variety of syntactic and semantic natural language processing tasks. However, we also demonstrated that the representations are (relatively) poor when used for tasks that require knowledge of discourse structures and entities in text. We also studied the differences in representations learned by the layers of LSTMs and transformers, uncovering marked differences in how they encode and store information. Lastly, we examined unsupervised language model pretraining as a task for learning general representations of language—can we do better with explicit linguistic supervision? Our experiments indicate that models trained solely on tasks with explicit linguistic supervision learn representations that are less general than neural language models, though their specificity leads to particular gains on related tasks.

I speculatively posit the following directions for future research:

IMPROVED ANALYSIS METHODS While probing models are straightforward to apply, drawing meaningful conclusions can be difficult. Probing models, at best, only provide *existence proofs*—the inability to extract information required for a task from a particular representation does not necessarily imply that the representation fails to encode the feature of interest, since the probing model itself could be at fault (e.g., if it does not have enough capacity). Furthermore, probing classifiers trained atop of representations produced from randomly-initialized networks often perform quite well, despite the fact that randomly-initialized networks are unlikely to have knowledge of any linguistic properties (Zhang and Bowman, 2018; Wieting and Kiela, 2019). In general, it can be hard to disentangle the ability and capacity of the probing model from the features inherently present in the analyzed representations. While Chapter 3 makes some progress toward better understanding this tradeoff, further study is needed—

when can we trust the results from our probing models, and what are the best practices for conducting these analyses?

UNDERSTANDING HOW LINGUISTIC PHENOMENA ARE ENCODED Probing models are a coarse tool—they provide evidence of the presence of particular predictive features of interest, but little information about *how* they are encoded in the studied representations. Recent work has studied how syntactic information, in the form of parse trees, are encoded in language model-derived contextual word representations (Hewitt and Manning, 2019), but further work is needed to characterize how other linguistic phenomena are represented within the outputs of end-to-end neural NLP systems.

EXPLICIT LINGUISTIC BIASES Although neural NLP systems seem to perform well on standard NLP benchmarks, they suffer from a host of maladies: they require massive training datasets, fail to generalize to even different domains, and make opaque decisions. Linguistic structure may provide a path toward useful inductive biases in neural models that can improve their generalizability and interpretability. As different as any new domains or data may be, the data is still human language. Language, in all its varying shapes and forms, has regular patterns and structure. This linguistic structure underlies all of these differing data distributions, but the majority of neural NLP models do not explicitly consider this unifying feature. On the contrary, modern neural models are unconstrained by design, since end-to-end learning gives them the freedom to tune their millions of parameters toward optimizing an end objective. I suspect that this freedom is also a curse, since these drastically overparameterized models overfit to their particular training dataset, annotation artifacts and all (Levy et al., 2015; Gururangan et al., 2018, *inter alia*). Reining in these currently-unconstrained models by integrating the appropriate linguistic biases may improve their generalization while also increasing their sample efficiency. Such biases may also provide avenues for increased interpretability of model decisions, perhaps through the generation of intermediary structures.

REFERENCES

Abzianidze, Lasha, Johannes Bjerva, Kilian Evang, Hessel Haagsma, Rik van Noord, Pierre Ludmann, Duc-Duy Nguyen, and Johan Bos (2017). "The Parallel Meaning Bank: Towards a Multilingual Corpus of Translations Annotated with Compositional Meaning Representations." In: *Proc. of EACL*.

Adi, Yossi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg (2017). "Fine-grained Analysis of Sentence Embeddings Using Auxiliary Prediction Tasks." In: *Proc. of ICLR*.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate." In: *Proc. of ICLR*.

Bangalore, Srinivas and Aravind K. Joshi (1999). "Supertagging: An Approach To Almost Parsing." In: *Computational Linguistics* 25.2, pp. 237–265.

Belinkov, Yonatan (2018). "On Internal Language Representations in Deep Learning: An Analysis of Machine Translation and Speech Recognition." PhD thesis. Massachusetts Institute of Technology.

Belinkov, Yonatan and James Glass (2019). "Analysis Methods in Neural Language Processing: A Survey." In: *Transactions of the Association for Computational Linguistics*.

Belinkov, Yonatan, Lluís Màrquez, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James R. Glass (2017a). "Evaluating Layers of Representation in Neural Machine Translation on Part-of-Speech and Semantic Tagging Tasks." In: *Proc. of IJCNLP*.

Belinkov, Yonatan, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James R. Glass (2017b). "What do Neural Machine Translation Models Learn about Morphology?" In: *Proc. of ACL*.

Bjerva, Johannes, Barbara Plank, and Johan Bos (2016). "Semantic Tagging with Deep Residual Networks." In: *Proc. of COLING*.

Blevins, Terra, Omer Levy, and Luke Zettlemoyer (2018). "Deep RNNs Encode Soft Hierarchical Syntax." In: *Proc. of ACL*.

Bohnet, Bernd, Ryan T. McDonald, Gonçalo Simões, Daniel Andor, Emily Pitler, and Joshua Maynez (2018). "Morphosyntactic Tagging with a Meta-BiLSTM Model over Context Sensitive Token Encodings." In: *Proc. of ACL*.

Bosselut, Antoine, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi (2017). "Simulating Action Dynamics with Neural Process Networks." In: *Proc. of ICLR*.

Bowman, Samuel R. et al. (2018). *Looking for ELMo's friends: Sentence-Level Pretraining Beyond Language Modeling*. ArXiv:1812.10860.

Chelba, Ciprian, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn (2014). "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling." In: *Proc. of INTERSPEECH*.

Chen, Mia Xu et al. (2018). "The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation." In: *Proc. of ACL*.

Chiang, David (2005). "A Hierarchical Phrase-Based Model for Statistical Machine Translation." In: *Proc. of ACL*.

Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." In: *Proc. of EMNLP*.

Conneau, Alexis, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes (2017). "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data." In: *Proc. of EMNLP*.

Conneau, Alexis, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni (2018). "What you can cram into a single vector: Probing sentence embeddings for linguistic properties." In: *Proc. of ACL*.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "ImageNet: A large-scale hierarchical image database." In: *Proc. of CVPR*.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Proc. of NAACL*.

Dozat, Timothy and Christopher D. Manning (2016). "Deep Biaffine Attention for Neural Dependency Parsing." In: *Proc. of ICLR*.

Dozat, Timothy and Christopher D. Manning (2018). "Simpler but More Accurate Semantic Dependency Parsing." In: *Proc. of ACL*.

Elman, Jeffrey L. (1990). "Finding Structure in Time." In: *Cognitive Science* 14, pp. 179–211.

Everaert, Martin B.H., Marinus A.C. Huybregts, Noam Chomsky, Robert C. Berwick, and Johan J. Bolhuis (2015). "Structures, Not Strings: Linguistics as Part of the Cognitive Sciences." In: *Trends in Cognitive Sciences* 19.12, pp. 729–743.

Ficler, Jessica and Yoav Goldberg (2016). "Coordination Annotation Extension in the Penn Tree Bank." In: *Proc. of ACL*.

Futrell, Richard and Roger P. Levy (2019). "Do RNNs learn human-like abstract word order preferences?" In: *Proc. of SCiL*.

Gaddy, David, Mitchell Stern, and Dan Klein (2018). "What's Going On in Neural Constituency Parsers? An Analysis." In: *Proc. of NAACL*.

Gardner, Matt, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, and

Luke Zettlemoyer (2018). "AllenNLP: A Deep Semantic Natural Language Processing Platform." In: *Proc. of NLP-OSS*.

Goldberg, Yoav (2017). "Neural Network Methods for Natural Language Processing." In: *Synthesis Lectures on Human Language Technologies* 10.1, pp. 1–309.

Gulordava, Kristina, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni (2018). "Colorless green recurrent networks dream hierarchically." In: *Proc. of NAACL*.

Gururangan, Suchin, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith (2018). "Annotation Artifacts in Natural Language Inference Data." In: *Proc. of NAACL*.

Hashimoto, Kazuma, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher (2017). "A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks." In: *Proc. of EMNLP*.

Hewitt, John and Christopher D Manning (2019). "A Structural Probe for Finding Syntax in Word Representations." In: *Proc. of NAACL*.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory." In: *Neural Computation* 9 8, pp. 1735–80.

Hockenmaier, Julia and Mark Steedman (2007). "CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank." In: *Computational Linguistics* 33.3, pp. 355–396.

Howard, Jeremy and Sebastian Ruder (2018). "Universal Language Model Fine-tuning for Text Classification." In: *Proc. of ACL*.

Hupkes, Dieuwke, Sara Veldhoen, and Willem Zuidema (2018). "Visualisation and 'Diagnostic Classifiers' Reveal How Recurrent and Recursive Neural Networks Process Hierarchical Structure." In: *Proc. of IJCAI*.

Ji, Yangfeng, Chenhao Tan, Sebastian Martschat, Yejin Choi, and Noah A. Smith (2017). "Dynamic Entity Representations in Neural Language Models." In: *Proc. of EMNLP*.

Jouppi, Norman P. et al. (2017). "In-Datacenter Performance Analysis of a Tensor Processing Unit." In: *Proc. of ISCA*.

Jumelet, Jaap and Dieuwke Hupkes (2018). "Do Language Models Understand Anything? On the Ability of LSTMs to Understand Negative Polarity Items." In: *Proc. of BlackboxNLP*.

Karpathy, Andrej, Justin Johnson, and Li Fei-Fei (2016). "Visualizing and Understanding Recurrent Networks." In: *Proc. of ICLR*.

Khandelwal, Urvashi, He He, Peng Qi, and Dan Jurafsky (2018). "Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context." In: *Proc. of ACL*.

Kingma, Diederik P. and Jimmy Lei Ba (2015). "Adam: A Method for Stochastic Optimization." In: *Proc. of ICLR*.

Kuncoro, Adhiguna, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A Smith (2017). "What Do Recurrent

Neural Network Grammars Learn About Syntax?" In: *Proc. of EACL*.

Lafferty, John D., Andrew McCallum, and Fernando Pereira (2001). "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data." In: *Proc. of ICML*.

Levy, Omer, Steffen Remus, Chris Biemann, and Ido Dagan (2015). "Do Supervised Distributional Methods Really Learn Lexical Inference Relations?" In: *Proc. of NAACL*.

Lewis, Mike, Kenton Lee, and Luke Zettlemoyer (2016). "LSTM CCG Parsing." In: *Proc. of NAACL*.

Li, Jiwei, Xinlei Chen, Eduard Hovy, and Dan Jurafsky (2016). "Visualizing and Understanding Neural Models in NLP." In: *Proc. of NAACL*.

Linzen, Tal (2019). "What can linguistics and deep learning contribute to each other?" In: *Language*.

Linzen, Tal, Emmanuel Dupoux, and Yoav Goldberg (2016). "Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies." In: *Transactions of the Association of Computational Linguistics* 4.1, pp. 521–535.

Liu, Nelson F., Omer Levy, Roy Schwartz, Chenhao Tan, and Noah A. Smith (2018). "LSTMs Exploit Linguistic Attributes of Data." In: *Proc. of RepL4NLP*.

Liu, Nelson F., Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith (2019). "Linguistic Knowledge and Transferability of Contextual Representations." In: *Proc. of NAACL*.

Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini (1993). "Building a Large Annotated Corpus of English: The Penn Treebank." In: *Computational Linguistics* 19.2, pp. 313–330.

Marneffe, Marie-Catherine de, Christopher D. Manning, and Christopher Potts (2012). "Did It Happen? The Pragmatic Complexity of Veridicality Assessment." In: *Computational Linguistics* 38, pp. 301–333.

McCann, Bryan, James Bradbury, Caiming Xiong, and Richard Socher (2017). "Learned in Translation: Contextualized Word Vectors." In: *Proc. of NeurIPS*.

Mikolov, Tomáš, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur (2010). "Recurrent neural network based language model." In: *Proc. of INTERSPEECH*.

Mikolov, Tomáš, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur (2011). "Extensions of recurrent neural network language model." In: *Proc of. ICASSP*.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "Distributed Representations of Words and Phrases and their Compositionality." In: *Proc. of NeurIPS*.

Oepen, Stephan, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, and Zdenka Uresova

(2015). "SemEval 2015 Task 18: Broad-Coverage Semantic Dependency Parsing." In: *Proc. of SemEval 2015*.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "GloVe: Global Vectors for Word Representation." In: *Proc. of EMNLP*.

Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018a). "Deep contextualized word representations." In: *Proc. of NAACL*.

Peters, Matthew E., Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih (2018b). "Dissecting Contextual Word Embeddings: Architecture and Representation." In: *Proc. of EMNLP*.

Peters, Matthew, Sebastian Ruder, and Noah A Smith (2019). *To Tune or Not to Tune? Adapting Pretrained Representations to Diverse Tasks*. ArXiv:1903.05987.

Pradhan, Sameer, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang (2012). "CoNLL-2012 Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes." In: *Proc. of CoNLL*.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever (2018). *Improving Language Understanding by Generative Pre-Training*. Technical report, OpenAI.

Raina, Rajat, Anand Madhavan, and Andrew Y Ng (2009). "Large-scale deep unsupervised learning using graphics processors." In: *Proc. of ICML*.

Rei, Marek and Anders Sogaard (2019). "Jointly Learning to Label Sentences and Tokens." In: *Proc. of AAAI*.

Rei, Marek and Helen Yannakoudakis (2016). "Compositional Sequence Labeling Models for Error Detection in Learner Writing." In: *Proc. of ACL*.

Rudinger, Rachel, Aaron Steven White, and Benjamin Van Durme (2018). "Neural models of factuality." In: *Proc. of NAACL*.

Saurí, Roser and James Pustejovsky (2009). "FactBank: a corpus annotated with event factuality." In: *Language Resources and Evaluation* 43, pp. 227–268.

Saurí, Roser and James Pustejovsky (2012). "Are You Sure That This Happened? Assessing the Factuality Degree of Events in Text." In: *Computational Linguistics* 38, pp. 261–299.

Schneider, Nathan, Jena D. Hwang, Vivek Srikumar, Jakob Prange, Austin Blodgett, Sarah R. Moeller, Aviram Stern, Adi Bitan, and Omri Abend (2018). "Comprehensive Supersense Disambiguation of English Prepositions and Possessives." In: *Proc. of ACL*.

Schuster, Tal, Ori Ram, Regina Barzilay, and Amir Globerson (2019). "Cross-Lingual Alignment of Contextual Word Embeddings, with Applications to Zero-shot Dependency Parsing." In: *Proc. of NAACL*.

Schwaller, Philippe, Theophile Gaudin, David Lanyi, Costas Bekas, and Teodoro Laino (2017). ""Found in Translation": Predicting

Outcomes of Complex Organic Chemistry Reactions using Neural Sequence-to-Sequence Models." In: *Proc. of NIPS Machine Learning for Molecules and Materials Workshop*.

Shi, Xing, Kevin Knight, and Deniz Yuret (2016). "Why Neural Translations are the Right Length." In: *Proc. of EMNLP*.

Shi, Xing, Inkit Padhi, and Kevin Knight (2016). "Does String-Based Neural MT Learn Source Syntax?" In: *Proc. of EMNLP*.

Silveira, Natalia, Timothy Dozat, Marie-Catherine de Marneffe, Samuel R. Bowman, Miriam Connor, John Bauer, and Christopher D. Manning (2014). "A Gold Standard Dependency Corpus for English." In: *Proc. of LREC*.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). "Sequence to sequence learning with neural networks." In: *Proc. of NeurIPS*.

Sutton, Richard S. (2019). *The Bitter Lesson*. URL: http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Tang, Gongbo, Mathias Müller, Annette Rios, and Rico Sennrich (2018). "Why Self-Attention? A Targeted Evaluation of Neural Machine Translation Architectures." In: *Proc. of EMNLP*.

Tenney, Ian et al. (2019). "What do you learn from context? Probing for sentence structure in contextualized word representations." In: *Proc. of ICLR*.

Tjong Kim Sang, Erik F. and Sabine Buchholz (2000). "Introduction to the CoNLL-2000 Shared Task: Chunking." In: *Proc. of LLL and CoNLL*.

Tjong Kim Sang, Erik F. and Fien De Meulder (2003). "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition." In: *Proc. of CoNLL*.

Tsvetkov, Yulia, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer (2015). "Evaluation of Word Vector Representations by Subspace Alignment." In: *Proc. of EMNLP*.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). "Attention Is All You Need." In: *Proc. of NeurIPS*.

Vinyals, Oriol, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton (2015). "Grammar as a Foreign Language." In: *Proc. of NIPS*.

Weiss, Gail, Yoav Goldberg, and Eran Yahav (2018). "On the Practical Computational Power of Finite Precision RNNs for Language Recognition." In: *Proc. of ACL*.

Wieting, John and Douwe Kiela (2019). "No Training Required: Exploring Random Encoders for Sentence Classification." In: *Proc. of ICLR*.

Wilcox, Ethan, Roger Levy, Takashi Morita, and Richard Futrell (2018). "What do RNN Language Models Learn about Filler-Gap Dependencies?" In: *Proc. of BlackboxNLP*.

Yang, Zichao, Phil Blunsom, Chris Dyer, and Wang Ling (2017). "Reference-Aware Language Models." In: *Proc. of EMNLP*.

Yannakoudakis, Helen, Ted Briscoe, and Ben Medlock (2011). "A New Dataset and Method for Automatically Grading ESOL Texts." In: *Proc. of ACL*.

Yasunaga, Michihiro, Jungo Kasai, and Dragomir R. Radev (2018). "Robust Multilingual Part-of-Speech Tagging via Adversarial Training." In: *Proc. of NAACL*.

Zhang, Kelly W. and Samuel R. Bowman (2018). "Language Modeling Teaches You More Syntax than Translation Does: Lessons Learned Through Auxiliary Task Analysis." In: *Proc. of BlackboxNLP*.

Zhu, Yukun, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler (2015). "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books." In: *Proc. of ICCV*.

Zipf, George Kingsley (1935). *The Psycho-Biology of Language*. Houghton, Mifflin.

APPENDIX

A

SUPPLEMENTAL MATERIAL: LINGUISTIC KNOWLEDGE AND TRANSFERABILITY OF CONTEXTUAL WORD REPRESENTATIONS

A.1 PROBING TASK SETUP DETAILS

SYNTACTIC CONSTITUENCY ANCESTOR TAGGING We remove the top-level ROOT node in each sentence. For words that do not have a parent, grandparent, or great-grandparent, we set the label to "None". The example is then treated as any other, and the probing model is required to predict this "None" label during training and evaluation.

PREPOSITION SUPERSENSE DISAMBIGUATION Since we focus on the linguistic knowledge within individual or pairs of CWRs, we train and evaluate our probing models on only single-word adpositions.

CONJUNCT IDENTIFICATION Our probing models are only trained and evaluated on sentences with a coordination construction in them.

A.2 PROBING MODEL TRAINING DETAILS

Our probing models are trained with Adam (Kingma and Ba, 2015), using a learning rate of 0.001. We train for 50 epochs, using early stopping with a patience of 3. Our models are implemented in the AllenNLP framework (Gardner et al., 2018).

For contextualizers that use subword representations (e.g., the OpenAI transformer and BERT), we aggregate subword representations into token representations by taking a token's representation to be the representation of its final subword.

A.3 REFERENCES TO STATE-OF-THE-ART TASK-SPECIFIC MODELS (WITHOUT PRETRAINING)

Task	Previous state of the art (without pretraining)
CCG	94.7 (Lewis, Lee, and Zettlemoyer, 2016)
POS (PTB)	97.96 (Bohnet et al., 2018)
POS (EWT)	95.82 (Yasunaga, Kasai, and Radev, 2018)
Chunk	95.77 (Hashimoto et al., 2017)
NER	91.38 (Hashimoto et al., 2017)
ST	95.15 (Bjerva, Plank, and Bos, 2016)
GED	39.83 (Rei and Sogaard, 2019)
PS-Role	66.89 (Schneider et al., 2018)
PS-Fxn	78.29 (Schneider et al., 2018)
EG	77.10 (Rudinger, White, and Durme, 2018)

Table 4: Performance of prior state of the art models (without pretraining) for each task.

Note that the performance reported in this paper for the preposition supersense identification models of Schneider et al. (2018) differs from their published result. Their published result is the accuracy on all adpositions; since we only train and evaluate our model on single-word adpositions, the number we report in this paper is the performance of the Schneider et al. (2018) model on only single-word adpositions.

A.4 PERFORMANCE OF PRETRAINED CONTEXTUALIZERS ON ALL TASKS

A.4.1 *Token Labeling (ELMo and OpenAI Transformer)*

Pretrained Representation	POS							Supersense ID		
	CCG	PTB	EWT	Parent	GParent	GGParent	ST	PS-Role	PS-Fxn	EF
ELMo Original, Layer 0	73.43	93.31	89.71	85.23	54.58	41.57	83.99	41.45	52.41	52.49
ELMo Original, Layer 1	93.31	97.26	95.61	95.56	81.61	67.50	93.82	74.12	84.87	73.20
ELMo Original, Layer 2	91.23	96.45	94.52	94.35	76.22	62.32	92.41	75.44	83.11	72.11
ELMo Original, Scalar Mix	92.96	97.19	95.09	95.56	81.56	67.42	93.86	74.56	84.65	72.96
ELMo (4-layer), Layer 0	73.41	93.42	89.30	85.45	55.40	42.22	83.95	40.13	55.26	53.58
ELMo (4-layer), Layer 1	93.81	97.31	95.60	95.70	81.57	67.66	94.18	74.78	85.96	73.03
ELMo (4-layer), Layer 2	92.47	97.09	95.08	95.01	77.08	63.04	93.43	74.12	85.53	70.97
ELMo (4-layer), Layer 3	91.56	96.82	94.56	94.65	75.58	61.04	92.82	74.12	83.55	70.66
ELMo (4-layer), Layer 4	90.67	96.44	93.99	94.24	75.70	61.45	91.90	73.46	83.77	72.59
ELMo (4-layer), Scalar Mix	93.23	97.34	95.14	95.55	81.36	67.47	94.05	76.10	84.65	72.70
ELMo (transformer), Layer 0	73.06	93.27	89.42	85.59	55.03	41.38	83.81	41.45	54.39	53.13
ELMo (transformer), Layer 1	91.66	97.09	94.78	94.43	77.28	62.69	93.78	65.13	80.04	67.19
ELMo (transformer), Layer 2	92.68	96.93	95.13	95.15	81.37	67.39	93.71	69.74	80.26	70.88
ELMo (transformer), Layer 3	92.82	96.97	94.74	95.28	82.16	68.06	93.45	70.61	82.24	70.24
ELMo (transformer), Layer 4	91.86	96.71	94.41	94.97	81.48	67.33	92.82	72.81	82.02	69.97
ELMo (transformer), Layer 5	91.06	96.24	93.85	94.30	79.65	64.92	91.92	69.52	79.82	70.21
ELMo (transformer), Layer 6	90.19	96.33	93.62	93.98	77.40	63.49	91.78	65.57	80.48	70.82
ELMo (transformer), Scalar Mix	93.66	97.35	94.59	95.16	83.38	69.29	94.26	72.59	82.46	71.94
OpenAI transformer, Layer 0	71.58	89.54	87.44	84.50	56.24	46.31	81.18	37.72	48.90	55.03
OpenAI transformer, Layer 1	78.08	93.32	89.93	88.75	63.59	53.28	85.73	43.64	61.40	63.13
OpenAI transformer, Layer 2	78.19	92.71	85.27	88.22	65.85	56.34	85.54	52.41	66.45	65.69
OpenAI transformer, Layer 3	79.53	93.43	89.67	88.73	67.34	58.10	86.17	53.51	70.18	68.39
OpenAI transformer, Layer 4	80.95	93.82	91.28	90.07	69.34	60.74	87.34	58.55	71.27	69.82
OpenAI transformer, Layer 5	82.03	93.82	91.11	90.51	71.41	62.69	87.81	60.75	73.46	70.92
OpenAI transformer, Layer 6	82.38	93.45	88.09	90.32	72.10	63.68	87.46	64.04	74.12	72.08
OpenAI transformer, Layer 7	82.61	93.25	86.50	90.71	72.60	63.69	86.49	65.13	76.32	73.87
OpenAI transformer, Layer 8	81.43	92.10	86.66	91.00	72.66	64.01	86.65	66.23	76.97	73.86
OpenAI transformer, Layer 9	81.73	91.99	86.60	90.84	72.34	63.72	86.19	66.01	76.54	74.03
OpenAI transformer, Layer 10	81.73	92.05	86.37	90.74	71.41	62.45	86.22	63.38	75.88	73.30
OpenAI transformer, Layer 11	81.97	91.64	86.62	90.43	70.48	60.84	85.91	63.16	76.97	71.99
OpenAI transformer, Layer 12	82.69	92.18	90.87	90.89	69.14	58.74	87.43	63.60	75.66	71.34
OpenAI transformer, Scalar Mix	83.94	94.63	92.60	92.08	73.11	64.64	88.73	64.69	79.17	74.25
GloVe (840B.300d)	71.58	90.49	83.93	81.77	54.01	41.21	80.92	40.79	51.54	49.70
Previous state of the art	94.7	97.96	96.73	-	-	-	95.15	66.89	78.29	77.10

Table 5: Token labeling task performance of a linear probing model trained on top of the ELMo and OpenAI contextualizers, compared against a GloVe-based probing baseline and the previous state of the art.

A.4.2 *Token Labeling (BERT)*

Pretrained Representation	POS							Supersense ID		
	CCG	PTB	EWT	Parent	GParent	GGParent	ST	PS-Role	PS-Fxn	EF
BERT (base, cased), Layer 0	71.45	89.99	86.77	84.41	55.92	46.07	82.25	42.11	54.82	52.70
BERT (base, cased), Layer 1	81.67	93.80	90.58	89.47	62.92	50.93	88.89	50.88	67.76	59.83
BERT (base, cased), Layer 2	88.43	95.76	93.72	92.98	71.73	57.84	92.23	63.60	75.00	64.91
BERT (base, cased), Layer 3	89.77	96.08	94.30	93.92	73.24	58.57	92.85	64.69	78.95	65.58
BERT (base, cased), Layer 4	91.41	96.57	94.58	94.67	76.09	61.17	93.38	66.23	79.17	67.55
BERT (base, cased), Layer 5	92.22	96.68	94.93	95.10	77.79	63.56	93.47	68.20	82.89	69.08
BERT (base, cased), Layer 6	93.14	96.95	95.15	95.46	79.75	65.36	93.72	76.10	84.65	71.26
BERT (base, cased), Layer 7	93.51	96.92	95.12	95.70	80.38	65.96	93.62	77.85	86.40	71.54
BERT (base, cased), Layer 8	93.67	96.80	95.21	95.60	81.04	66.66	93.37	79.61	87.94	73.49
BERT (base, cased), Layer 9	93.51	96.68	94.94	95.64	80.70	66.53	93.18	79.39	86.84	75.11
BERT (base, cased), Layer 10	93.25	96.54	94.51	95.26	79.60	65.49	92.90	79.17	86.18	74.70
BERT (base, cased), Layer 11	92.75	96.40	94.31	95.00	78.50	64.34	92.64	77.41	85.53	75.11
BERT (base, cased), Layer 12	92.21	96.09	93.86	94.55	76.95	62.87	92.34	78.07	84.65	73.77
BERT (base, cased), Scalar Mix	93.78	97.02	95.63	95.83	81.67	67.48	93.85	78.51	85.96	74.88
BERT (large, cased), Layer 0	71.06	89.84	86.81	84.28	55.84	46.17	82.31	38.38	54.61	52.81
BERT (large, cased), Layer 1	79.49	92.58	89.45	88.50	60.96	49.88	87.16	53.51	65.13	59.49
BERT (large, cased), Layer 2	83.30	94.03	91.70	90.48	64.91	51.94	89.47	58.55	71.93	62.49
BERT (large, cased), Layer 3	83.32	94.09	91.92	90.76	64.99	52.26	89.67	58.33	72.81	62.52
BERT (large, cased), Layer 4	88.51	95.61	93.36	93.26	70.99	56.22	92.58	65.35	78.29	65.06
BERT (large, cased), Layer 5	89.69	95.95	94.15	93.94	72.62	57.58	93.05	62.06	76.97	65.79
BERT (large, cased), Layer 6	90.91	96.14	94.35	94.47	75.59	60.80	93.35	62.72	78.51	67.00
BERT (large, cased), Layer 7	91.72	96.30	94.64	94.55	76.35	60.98	93.55	67.98	81.36	66.42
BERT (large, cased), Layer 8	91.56	96.36	94.80	94.61	76.40	61.93	93.50	66.89	80.26	68.56
BERT (large, cased), Layer 9	91.76	96.31	94.86	94.70	75.95	61.60	93.44	66.89	82.02	69.12
BERT (large, cased), Layer 10	91.71	96.27	94.89	94.88	75.84	61.44	93.42	68.64	79.39	69.37
BERT (large, cased), Layer 11	92.01	96.26	94.96	95.10	77.01	62.79	93.39	70.83	81.80	71.12
BERT (large, cased), Layer 12	92.82	96.48	95.27	95.31	78.66	64.51	93.61	74.34	84.21	72.44
BERT (large, cased), Layer 13	93.48	96.73	95.56	95.72	80.51	65.85	93.83	76.54	85.75	72.91
BERT (large, cased), Layer 14	93.85	96.73	95.54	95.98	81.89	67.02	93.81	78.95	87.94	72.72
BERT (large, cased), Layer 15	94.21	96.72	95.80	96.10	82.46	67.53	93.76	79.17	89.25	72.79
BERT (large, cased), Layer 16	94.28	96.67	95.62	96.05	82.78	67.90	93.61	78.73	90.13	74.27
BERT (large, cased), Layer 17	94.13	96.53	95.55	95.92	82.56	67.74	93.45	79.17	87.06	75.52
BERT (large, cased), Layer 18	93.76	96.38	95.45	95.57	81.47	67.11	93.21	79.17	87.06	75.95
BERT (large, cased), Layer 19	93.36	96.25	95.30	95.38	80.47	66.08	93.01	76.10	85.96	76.25
BERT (large, cased), Layer 20	93.06	96.10	94.96	95.20	79.32	64.86	92.78	78.29	87.72	75.92
BERT (large, cased), Layer 21	91.83	95.38	94.05	94.16	76.84	62.43	91.65	74.12	82.89	75.16
BERT (large, cased), Layer 22	89.66	93.88	92.30	92.62	74.73	60.76	89.42	73.90	82.02	74.28
BERT (large, cased), Layer 23	88.70	93.02	91.90	92.36	73.33	59.27	88.92	69.08	80.70	73.54
BERT (large, cased), Layer 24	87.65	92.60	90.84	91.81	71.98	57.95	88.26	69.74	78.73	72.65
BERT (large, cased), Scalar Mix	94.48	97.17	96.05	96.27	83.51	68.90	93.96	78.95	87.06	76.13

Table 6: Token labeling task performance of a linear probing model trained on top of the BERT contextualizers.

A.4.3 *Segmentation (ELMo and OpenAI Transformer)*

Pretrained Representation	Chunk	NER	GED	Conj
ELMo Original, Layer 0	70.68	64.39	18.49	15.59
ELMo Original, Layer 1	90.04	82.85	29.37	38.72
ELMo Original, Layer 2	86.47	82.80	26.08	29.08
ELMo Original, Scalar Mix	89.29	82.90	27.54	39.57
ELMo (4-layer), Layer 0	70.57	63.96	8.46	15.15
ELMo (4-layer), Layer 1	89.78	81.04	28.07	36.37
ELMo (4-layer), Layer 2	87.18	80.19	29.24	31.44
ELMo (4-layer), Layer 3	86.20	81.56	28.51	28.57
ELMo (4-layer), Layer 4	85.07	82.06	23.85	26.31
ELMo (4-layer), Scalar Mix	86.67	82.37	30.46	28.42
ELMo (transformer), Layer 0	71.01	64.23	13.25	15.69
ELMo (transformer), Layer 1	91.75	78.51	25.29	26.56
ELMo (transformer), Layer 2	92.18	80.92	28.63	34.99
ELMo (transformer), Layer 3	92.14	80.80	29.16	38.23
ELMo (transformer), Layer 4	91.32	80.47	29.71	38.52
ELMo (transformer), Layer 5	89.18	81.21	30.80	35.49
ELMo (transformer), Layer 6	87.96	79.77	27.20	29.17
ELMo (transformer), Scalar Mix	92.08	81.68	26.56	38.45

Pretrained Representation	Chunk	NER	GED	Conj
OpenAI transformer, Layer 0	66.59	46.29	14.78	16.84
OpenAI transformer, Layer 1	77.87	48.88	19.72	17.59
OpenAI transformer, Layer 2	79.67	52.13	21.59	20.72
OpenAI transformer, Layer 3	80.78	52.40	22.58	22.36
OpenAI transformer, Layer 4	82.95	54.62	25.61	23.04
OpenAI transformer, Layer 5	84.67	56.25	29.69	25.53
OpenAI transformer, Layer 6	85.46	56.46	30.69	27.25
OpenAI transformer, Layer 7	86.06	57.73	33.10	30.68
OpenAI transformer, Layer 8	85.75	56.50	32.17	33.06
OpenAI transformer, Layer 9	85.40	57.31	31.90	32.65
OpenAI transformer, Layer 10	84.52	57.32	32.08	30.27
OpenAI transformer, Layer 11	83.00	56.94	30.22	26.60
OpenAI transformer, Layer 12	82.44	58.14	30.81	25.19
OpenAI transformer, Scalar Mix	87.44	59.39	34.54	31.65
GloVe (840B.300d)	62.28	53.22	14.94	10.53
Previous state of the art	95.77	91.38	34.76	-

Table 7: Segmentation task performance of a linear probing model trained on top of the ELMo and OpenAI contextualizers, compared against a GloVe-based probing baseline and the previous state of the art.

A.4.4 *Segmentation (BERT)*

Pretrained Representation	Chunk	NER	GED	Conj
BERT (base, cased), Layer 0	69.86	53.50	12.63	16.24
BERT (base, cased), Layer 1	75.56	66.94	16.85	21.83
BERT (base, cased), Layer 2	86.64	71.08	22.66	22.87
BERT (base, cased), Layer 3	87.70	73.83	25.80	25.50
BERT (base, cased), Layer 4	90.64	77.28	31.35	29.39
BERT (base, cased), Layer 5	91.21	78.81	32.34	30.58
BERT (base, cased), Layer 6	92.29	80.81	37.85	35.26
BERT (base, cased), Layer 7	92.64	81.50	40.14	35.86
BERT (base, cased), Layer 8	92.11	82.45	42.08	42.26
BERT (base, cased), Layer 9	91.95	82.71	43.20	43.93
BERT (base, cased), Layer 10	91.30	82.66	42.46	43.38
BERT (base, cased), Layer 11	90.71	82.42	43.30	41.35
BERT (base, cased), Layer 12	89.38	80.64	39.87	39.34
BERT (base, cased), Scalar Mix	92.96	82.43	43.22	43.15

Pretrained Representation	Chunk	NER	GED	Conj
BERT (large, cased), Layer 0	70.42	53.95	13.44	16.65
BERT (large, cased), Layer 1	73.98	65.92	16.20	19.58
BERT (large, cased), Layer 2	79.82	67.96	17.26	20.01
BERT (large, cased), Layer 3	79.50	68.82	17.42	21.83
BERT (large, cased), Layer 4	87.49	71.13	24.06	23.21
BERT (large, cased), Layer 5	89.81	72.06	30.27	24.13
BERT (large, cased), Layer 6	89.92	74.30	31.44	26.75
BERT (large, cased), Layer 7	90.39	75.93	33.27	27.74
BERT (large, cased), Layer 8	90.28	76.99	33.34	29.94
BERT (large, cased), Layer 9	90.09	78.87	33.16	30.07
BERT (large, cased), Layer 10	89.92	80.08	33.31	30.17
BERT (large, cased), Layer 11	90.20	81.23	34.49	31.78
BERT (large, cased), Layer 12	91.22	83.00	37.27	34.10
BERT (large, cased), Layer 13	93.04	83.66	40.10	35.04
BERT (large, cased), Layer 14	93.64	84.11	43.11	39.67
BERT (large, cased), Layer 15	93.18	84.21	44.92	43.12
BERT (large, cased), Layer 16	93.14	84.34	45.37	46.54
BERT (large, cased), Layer 17	92.80	84.44	45.60	47.76
BERT (large, cased), Layer 18	91.72	84.03	45.82	47.34
BERT (large, cased), Layer 19	91.48	84.29	46.46	46.00
BERT (large, cased), Layer 20	90.78	84.25	46.07	44.81
BERT (large, cased), Layer 21	87.97	82.36	44.53	41.91
BERT (large, cased), Layer 22	85.19	77.58	43.03	37.49
BERT (large, cased), Layer 23	84.23	77.02	42.00	35.21
BERT (large, cased), Layer 24	83.30	74.83	41.29	34.38
BERT (large, cased), Scalar Mix	93.59	84.98	47.32	45.94

Table 8: Segmentation task performance of a linear probing model trained on top of the BERT contextualizers.

A.4.5 *Pairwise Relations (ELMo and OpenAI Transformer)*

Pretrained Representation	Syntactic Dep. Arc Prediction		Syntactic Dep. Arc Classification		Semantic Dep. Arc Prediction	Semantic Dep. Arc Classification	Coreference Arc Prediction
	PTB	EWT	PTB	EWT			
ELMo (original), Layer 0	78.27	77.73	82.05	78.52	70.65	77.48	72.89
ELMo (original), Layer 1	89.04	86.46	96.13	93.01	87.71	93.31	71.33
ELMo (original), Layer 2	88.33	85.34	94.72	91.32	86.44	90.22	68.46
ELMo (original), Scalar Mix	89.30	86.56	95.81	91.69	87.79	93.13	73.24
ELMo (4-layer), Layer 0	78.09	77.57	82.13	77.99	69.96	77.22	73.57
ELMo (4-layer), Layer 1	88.79	86.31	96.20	93.20	87.15	93.27	72.93
ELMo (4-layer), Layer 2	87.33	84.75	95.38	91.87	85.29	90.57	71.78
ELMo (4-layer), Layer 3	86.74	84.17	95.06	91.55	84.44	90.04	70.11
ELMo (4-layer), Layer 4	87.61	85.09	94.14	90.68	85.81	89.45	68.36
ELMo (4-layer), Scalar Mix	88.98	85.94	95.82	91.77	87.39	93.25	73.88
ELMo (transformer), Layer 0	78.10	78.04	81.09	77.67	70.11	77.11	72.50
ELMo (transformer), Layer 1	88.24	85.48	93.62	89.18	85.16	90.66	72.47
ELMo (transformer), Layer 2	88.87	84.72	94.14	89.40	85.97	91.29	73.03
ELMo (transformer), Layer 3	89.01	84.62	94.07	89.17	86.83	90.35	72.62
ELMo (transformer), Layer 4	88.55	85.62	94.14	89.00	86.00	89.04	71.80
ELMo (transformer), Layer 5	88.09	83.23	92.70	88.84	85.79	89.66	71.62
ELMo (transformer), Layer 6	87.22	83.28	92.55	87.13	84.71	87.21	66.35
ELMo (transformer), Scalar Mix	90.74	86.39	96.40	91.06	89.18	94.35	75.52
OpenAI transformer, Layer 0	80.80	79.10	83.35	80.32	76.39	80.50	72.58
OpenAI transformer, Layer 1	81.91	79.99	88.22	84.51	77.70	83.88	75.23
OpenAI transformer, Layer 2	82.56	80.22	89.34	85.99	78.47	85.85	75.77
OpenAI transformer, Layer 3	82.87	81.21	90.89	87.67	78.91	87.76	75.81
OpenAI transformer, Layer 4	83.69	82.07	92.21	89.24	80.51	89.59	75.99
OpenAI transformer, Layer 5	84.53	82.77	93.12	90.34	81.95	90.25	76.05
OpenAI transformer, Layer 6	85.47	83.89	93.71	90.63	83.88	90.99	74.43
OpenAI transformer, Layer 7	86.32	84.15	93.95	90.82	85.15	91.18	74.05
OpenAI transformer, Layer 8	86.84	84.06	94.16	91.02	85.23	90.86	74.20
OpenAI transformer, Layer 9	87.00	84.47	93.95	90.77	85.95	90.85	74.57
OpenAI transformer, Layer 10	86.76	84.28	93.40	90.26	85.17	89.94	73.86
OpenAI transformer, Layer 11	85.84	83.42	92.82	89.07	83.39	88.46	72.03
OpenAI transformer, Layer 12	85.06	83.02	92.37	89.08	81.88	87.47	70.44
OpenAI transformer, Scalar Mix	87.18	85.30	94.51	91.55	86.13	91.55	76.47
GloVe (840B.300d)	74.14	73.94	77.54	72.74	68.94	71.84	72.96

Table 9: Pairwise relation task performance of a linear probing model trained on top of the ELMo and OpenAI contextualizers, compared against a GloVe-based probing baseline.

A.4.6 *Pairwise Relations (BERT)*

Pretrained Representation	Syntactic Dep. Arc Prediction		Syntactic Dep. Arc Classification		Semantic Dep. Arc Prediction	Semantic Dep. Arc Classification	Coreference Arc Prediction
	PTB	EWT	PTB	EWT			
BERT (base, cased), Layer 0	83.00	80.36	83.47	79.15	80.26	80.35	74.93
BERT (base, cased), Layer 1	83.66	81.69	86.92	82.62	80.81	82.69	75.35
BERT (base, cased), Layer 2	84.00	82.66	91.90	88.51	79.34	87.45	75.19
BERT (base, cased), Layer 3	84.12	82.86	92.80	89.49	79.05	88.41	75.83
BERT (base, cased), Layer 4	85.50	84.07	93.91	91.02	81.37	90.20	76.14
BERT (base, cased), Layer 5	86.67	84.69	94.87	92.01	83.41	91.34	76.35
BERT (base, cased), Layer 6	87.98	85.91	95.57	93.01	85.73	92.47	75.95
BERT (base, cased), Layer 7	88.24	86.30	95.65	93.31	85.96	92.75	75.37
BERT (base, cased), Layer 8	88.64	86.49	95.90	93.39	86.59	93.18	76.39
BERT (base, cased), Layer 9	88.76	86.17	95.84	93.32	86.74	92.68	76.62
BERT (base, cased), Layer 10	88.16	85.86	95.42	92.82	86.29	91.79	76.84
BERT (base, cased), Layer 11	87.74	85.40	95.09	92.37	85.83	91.07	76.88
BERT (base, cased), Layer 12	85.93	83.99	94.79	91.70	82.71	90.10	76.78
BERT (base, cased), Scalar Mix	89.06	86.58	95.91	93.10	87.10	93.38	77.88
BERT (large, cased), Layer 0	82.22	79.92	83.57	79.32	79.04	81.25	73.75
BERT (large, cased), Layer 1	81.65	80.04	85.23	80.95	77.97	81.36	73.99
BERT (large, cased), Layer 2	81.84	80.09	87.39	83.80	77.17	82.44	73.89
BERT (large, cased), Layer 3	81.66	80.35	87.36	83.74	76.92	82.91	73.62
BERT (large, cased), Layer 4	83.56	82.17	91.44	88.45	78.43	87.32	72.99
BERT (large, cased), Layer 5	84.24	82.94	92.33	89.62	79.28	88.85	73.34
BERT (large, cased), Layer 6	85.05	83.50	93.75	91.02	80.18	90.14	74.02
BERT (large, cased), Layer 7	85.43	84.03	94.06	91.65	80.64	90.69	74.55
BERT (large, cased), Layer 8	85.41	83.92	94.18	91.66	80.64	90.82	75.92
BERT (large, cased), Layer 9	85.35	83.76	94.11	91.10	80.64	90.62	76.00
BERT (large, cased), Layer 10	85.51	83.92	94.09	91.17	81.51	90.43	76.19
BERT (large, cased), Layer 11	85.91	83.88	94.48	91.73	82.05	91.13	75.86
BERT (large, cased), Layer 12	86.80	85.13	95.03	92.37	83.99	92.08	75.13
BERT (large, cased), Layer 13	87.64	86.00	95.54	93.02	84.91	92.74	74.63
BERT (large, cased), Layer 14	88.62	86.50	95.94	93.62	85.91	93.51	75.16
BERT (large, cased), Layer 15	88.87	86.95	96.02	93.66	86.49	93.86	75.58
BERT (large, cased), Layer 16	89.36	87.25	96.18	93.86	87.79	93.83	75.15
BERT (large, cased), Layer 17	89.62	87.47	96.01	93.88	88.14	93.41	75.93
BERT (large, cased), Layer 18	89.41	87.00	95.82	93.47	87.77	93.00	77.85
BERT (large, cased), Layer 19	88.78	86.60	95.59	92.98	87.16	92.27	80.47
BERT (large, cased), Layer 20	88.24	85.87	95.12	92.47	86.45	91.33	80.94
BERT (large, cased), Layer 21	86.48	84.21	94.21	91.12	83.94	89.42	81.14
BERT (large, cased), Layer 22	85.42	83.24	92.94	90.02	82.01	88.17	80.36
BERT (large, cased), Layer 23	84.69	82.81	92.28	89.47	81.07	87.32	79.64
BERT (large, cased), Layer 24	83.24	81.48	91.07	87.88	78.24	85.98	79.35
BERT (large, cased), Scalar Mix	90.09	87.51	96.15	93.61	88.49	94.25	81.16

Table 10: Pairwise relation task performance of a linear probing model trained on top of the BERT contextualizers.

A.5 FULL RESULTS FOR TRANSFERRING BETWEEN PRETRAINING TASKS

A.5.1 *Token Labeling*

Pretrained Representation			Supersense ID		
	POS (EWT)	ST	PS-Role	PS-Fxn	EF
Untrained ELMo (original), Layer 0	77.05	76.09	36.99	48.17	43.08
Untrained ELMo (original), Layer 1	56.03	68.63	16.01	24.71	45.57
Untrained ELMo (original), Layer 2	55.89	68.51	16.01	25.44	46.06
Untrained ELMo (original), Scalar Mix	78.58	82.45	38.23	48.90	47.37
CCG, Layer 0	84.33	79.53	38.38	53.29	47.71
CCG, Layer 1	88.02	87.97	46.27	58.48	57.96
CCG, Layer 2	87.81	87.38	43.79	58.55	57.98
CCG, Scalar Mix	90.44	91.21	50.07	65.57	60.24
Chunk, Layer 0	82.51	78.45	37.06	49.12	38.93
Chunk, Layer 1	87.33	87.42	44.81	59.36	55.66
Chunk, Layer 2	86.61	87.04	39.91	58.11	56.95
Chunk, Scalar Mix	88.62	89.77	44.23	60.01	56.24
PTB (POS), Layer 0	84.58	79.95	37.43	49.49	46.19
PTB (POS), Layer 1	90.53	90.10	42.47	59.80	61.28
PTB (POS), Layer 2	90.45	89.83	44.37	58.92	62.14
PTB (POS), Scalar Mix	90.75	91.13	45.39	60.67	62.77
Parent, Layer 0	81.84	78.47	36.33	49.71	38.35
Parent, Layer 1	87.21	87.36	45.98	58.85	54.45
Parent, Layer 2	86.57	86.18	42.69	58.48	54.58
Parent, Scalar Mix	89.10	90.01	44.88	61.92	55.64
GParent, Layer 0	81.85	78.77	37.06	51.75	40.46
GParent, Layer 1	86.05	86.78	46.86	60.82	55.58
GParent, Layer 2	85.64	86.17	45.25	62.13	55.65
GParent, Scalar Mix	88.08	89.48	48.03	63.38	55.96
GGParent, Layer 0	81.44	77.88	38.74	49.12	42.17
GGParent, Layer 1	83.51	85.23	44.08	57.68	55.77
GGParent, Layer 2	83.17	84.10	39.40	56.29	55.82
GGParent, Scalar Mix	86.18	88.84	44.52	61.62	55.50
Syn. Arc Prediction (PTB), Layer 0	79.97	77.34	36.26	47.15	38.81
Syn. Arc Prediction (PTB), Layer 1	80.67	82.60	40.06	54.61	47.86
Syn. Arc Prediction (PTB), Layer 2	78.83	80.91	34.65	52.12	45.64
Syn. Arc Prediction (PTB), Scalar Mix	85.76	88.13	40.79	54.17	50.91

Pretrained Representation			Supersense ID		
	POS (EWT)	ST	PS-Role	PS-Fxn	EF
Syn. Arc Classification (PTB), Layer 0	83.61	79.61	37.21	51.97	42.07
Syn. Arc Classification (PTB), Layer 1	89.28	88.70	47.22	61.11	55.55
Syn. Arc Classification (PTB), Layer 2	88.77	88.12	44.66	58.92	56.16
Syn. Arc Classification (PTB), Scalar Mix	90.18	90.99	48.17	62.21	56.90
Sem. Arc Prediction, Layer 0	78.64	76.95	34.43	49.78	39.64
Sem. Arc Prediction, Layer 1	74.66	74.83	33.92	47.88	36.46
Sem. Arc Prediction, Layer 2	74.06	73.42	30.85	45.39	35.63
Sem. Arc Prediction, Scalar Mix	83.77	85.06	38.45	57.16	48.27
Sem. Arc Classification, Layer 0	83.17	79.17	38.60	51.54	44.79
Sem. Arc Classification, Layer 1	86.45	87.04	44.81	58.19	55.18
Sem. Arc Classification, Layer 2	85.42	85.87	41.45	58.55	52.87
Sem. Arc Classification, Scalar Mix	88.44	90.00	45.03	61.33	56.07
Conj, Layer 0	72.21	73.87	37.43	47.95	36.33
Conj, Layer 1	64.95	68.96	27.70	41.89	42.10
Conj, Layer 2	64.03	67.17	27.56	37.21	40.59
Conj, Scalar Mix	76.96	80.22	36.33	50.66	42.79
BiLM, Layer 0	87.54	90.22	50.88	67.32	59.65
BiLM, Layer 1	86.55	87.19	50.22	67.11	59.32
BiLM, Layer 2	86.49	89.67	49.34	66.01	59.45
BiLM, Scalar Mix	86.76	90.11	50.44	67.32	67.32
ELMo (original), Layer 0	89.71	83.99	41.45	52.41	52.49
ELMo (original), Layer 1	95.61	93.82	74.12	84.87	73.20
ELMo (original), Layer 2	94.52	92.41	75.44	83.11	72.11
ELMo (original), Scalar Mix	95.09	93.86	74.56	84.65	84.65
GloVe (840B.300d)	83.93	80.92	40.79	51.54	49.70

Table 11: Target token labeling task performance of contextualizers pretrained on a variety of different tasks. The probing model used is linear, and the contextualizer architecture is ELMo (original).

A.5.2 *Segmentation*

Pretrained Representation	NER	GED
Untrained ELMo (original), Layer 0	24.71	0.00
Untrained ELMo (original), Layer 1	0.00	0.00
Untrained ELMo (original), Layer 2	0.00	0.00
Untrained ELMo (original), Scalar Mix	34.28	1.81
CCG, Layer 0	32.30	8.89
CCG, Layer 1	44.01	22.68
CCG, Layer 2	42.45	25.15
CCG, Scalar Mix	49.07	4.52
Chunk, Layer 0	23.47	5.80
Chunk, Layer 1	45.44	5.46
Chunk, Layer 2	43.59	24.11
Chunk, Scalar Mix	46.83	4.30
PTB (POS), Layer 0	32.64	7.87
PTB (POS), Layer 1	52.03	5.80
PTB (POS), Layer 2	52.04	9.76
PTB (POS), Scalar Mix	53.51	3.19
Parent, Layer 0	25.11	6.66
Parent, Layer 1	42.76	6.22
Parent, Layer 2	42.49	8.33
Parent, Scalar Mix	47.06	3.01
GParent, Layer 0	30.39	4.58
GParent, Layer 1	47.67	6.20
GParent, Layer 2	47.87	10.34
GParent, Scalar Mix	50.06	1.71
GGParent, Layer 0	28.57	2.25
GGParent, Layer 1	46.21	4.32
GGParent, Layer 2	45.34	3.74
GGParent, Scalar Mix	48.19	1.54
Syn. Arc Prediction (PTB), Layer 0	26.77	1.82
Syn. Arc Prediction (PTB), Layer 1	43.93	5.94
Syn. Arc Prediction (PTB), Layer 2	41.83	14.50
Syn. Arc Prediction (PTB), Scalar Mix	46.58	1.47

Pretrained Representation	NER	GED
Syn. Arc Classification (PTB), Layer 0	33.10	3.51
Syn. Arc Classification (PTB), Layer 1	50.76	3.92
Syn. Arc Classification (PTB), Layer 2	49.64	5.77
Syn. Arc Classification (PTB), Scalar Mix	53.00	1.27
Sem. Arc Prediction, Layer 0	24.47	1.05
Sem. Arc Prediction, Layer 1	34.47	10.78
Sem. Arc Prediction, Layer 2	31.30	10.77
Sem. Arc Prediction, Scalar Mix	36.97	0.32
Sem. Arc Classification, Layer 0	34.00	5.08
Sem. Arc Classification, Layer 1	48.07	5.39
Sem. Arc Classification, Layer 2	46.67	6.24
Sem. Arc Classification, Scalar Mix	50.80	1.75
Conj, Layer 0	17.15	3.99
Conj, Layer 1	37.61	0.87
Conj, Layer 2	34.78	2.38
Conj, Scalar Mix	40.97	0.33
BiLM, Layer 0	56.05	3.99
BiLM, Layer 1	57.19	1.22
BiLM, Layer 2	57.05	1.03
BiLM, Scalar Mix	58.50	1.29
ELMo (original), Layer 0	64.39	18.49
ELMo (original), Layer 1	82.85	29.37
ELMo (original), Layer 2	82.80	26.08
ELMo (original), Scalar Mix	82.90	27.54
GloVe (840B.300d)	53.22	14.94

Table 12: Target segmentation task performance of contextualizers pretrained on a variety of different tasks. The probing model used is linear, and the contextualizer architecture is ELMo (original).

A.5.3 *Pairwise Prediction*

Pretrained Representation	Syn. Arc Prediction (EWT)	Syn. Arc Classification (EWT)	Coreference Arc Prediction
Untrained ELMo (original), Layer 0	73.75	66.27	66.25
Untrained ELMo (original), Layer 1	68.40	56.73	62.82
Untrained ELMo (original), Layer 2	68.86	56.62	63.15
Untrained ELMo (original), Scalar Mix	72.24	70.62	69.72
CCG, Layer 0	75.92	69.84	67.84
CCG, Layer 1	84.93	85.59	62.10
CCG, Layer 2	84.45	84.59	59.19
CCG, Scalar Mix	85.44	88.11	70.14
Chunk, Layer 0	76.67	69.72	65.60
Chunk, Layer 1	85.18	86.50	62.74
Chunk, Layer 2	84.80	84.84	60.23
Chunk, Scalar Mix	85.42	87.57	68.92
PTB (POS), Layer 0	76.07	70.32	67.50
PTB (POS), Layer 1	83.97	86.64	63.43
PTB (POS), Layer 2	83.88	86.44	61.61
PTB (POS), Scalar Mix	84.17	87.72	69.61
Parent, Layer 0	76.20	68.99	67.80
Parent, Layer 1	84.93	86.15	62.69
Parent, Layer 2	85.57	85.61	59.10
Parent, Scalar Mix	86.01	87.49	69.34
GParent, Layer 0	76.59	69.51	68.99
GParent, Layer 1	85.96	85.33	60.84
GParent, Layer 2	85.69	84.38	58.76
GParent, Scalar Mix	86.17	87.49	70.24
GGParent, Layer 0	76.28	69.91	69.24
GGParent, Layer 1	85.74	83.45	59.73
GGParent, Layer 2	85.49	82.12	58.89
GGParent, Scalar Mix	86.27	86.57	70.58
Syn. Arc Prediction (PTB), Layer 0	77.04	68.01	68.28
Syn. Arc Prediction (PTB), Layer 1	90.39	81.00	60.29
Syn. Arc Prediction (PTB), Layer 2	90.82	76.50	57.46
Syn. Arc Prediction (PTB), Scalar Mix	91.66	84.18	69.15

Pretrained Representation	Syn. Arc Prediction (EWT)	Syn. Arc Classification (EWT)	Coreference Arc Prediction
Syn. Arc Classification (PTB), Layer 0	76.14	71.80	68.40
Syn. Arc Classification (PTB), Layer 1	86.55	90.04	62.10
Syn. Arc Classification (PTB), Layer 2	87.46	89.35	59.74
Syn. Arc Classification (PTB), Scalar Mix	87.78	90.98	70.00
Sem. Arc Prediction, Layer 0	76.25	67.73	69.44
Sem. Arc Prediction, Layer 1	84.91	73.11	57.62
Sem. Arc Prediction, Layer 2	85.86	69.75	55.91
Sem. Arc Prediction, Scalar Mix	86.37	80.74	69.72
Sem. Arc Classification, Layer 0	75.85	70.12	68.96
Sem. Arc Classification, Layer 1	85.30	86.21	60.25
Sem. Arc Classification, Layer 2	86.10	84.50	58.39
Sem. Arc Classification, Scalar Mix	86.53	87.75	70.36
Conj, Layer 0	72.62	58.40	68.50
Conj, Layer 1	80.84	68.12	58.46
Conj, Layer 2	80.46	64.30	57.89
Conj, Scalar Mix	80.96	73.89	71.96
BiLM, Layer 0	84.27	86.74	71.75
BiLM, Layer 1	86.36	86.86	70.47
BiLM, Layer 2	86.44	86.19	70.14
BiLM, Scalar Mix	86.42	85.93	71.62
ELMo (original), Layer 0	77.73	78.52	72.89
ELMo (original), Layer 1	86.46	93.01	71.33
ELMo (original), Layer 2	85.34	91.32	68.46
ELMo (original), Scalar Mix	86.56	91.69	73.24
GloVe (840B.300d)	73.94	72.74	72.96

Table 13: Target pairwise prediction task performance of contextualizers pretrained on a variety of different tasks. The probing model used is linear, and the contextualizer architecture is ELMo (original).

COLOPHON

This thesis was typeset with the `classicthesis` style for LaTeX.

Final Version as of June 16, 2019.