EFFICIENT AND SCALABLE TRANSFER LEARNING FOR NATURAL LANGUAGE PROCESSING

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Kevin Clark
March 2021

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

Christopher Manning, Primary Adviser

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

Dan Jurafsky

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

Quoc Le

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

Abstract

Neural networks work best when trained on large amounts of data, but most labeled datasets in natural language processing (NLP) are small. As a result, neural NLP models often overfit to idiosyncrasies and artifacts in their training data rather than learning generalizable patterns. Transfer learning offers a solution: instead of learning a single task from scratch and in isolation, the model can benefit from the wealth of text on the web or other tasks with rich annotations. This additional data enables the training of bigger, more expressive networks. However, it also dramatically increases the computational cost of training, with recent models taking up to hundreds of GPU years to train.

To alleviate this cost, I develop transfer learning methods that learn much more efficiently than previous approaches while remaining highly scalable. First, I present a multi-task learning algorithm based on knowledge distillation that consistently improves over single-task training even when learning many diverse tasks. I next develop Cross-View Training, which revitalizes semi-supervised learning methods from the statistical era of NLP (self-training and co-training) while taking advantage of neural methods. The resulting models outperform pre-trained LSTM language models such as ELMo while training 10x faster. Lastly, I present ELECTRA, a self-supervised pre-training method for transformer networks based on energy-based models. ELECTRA learns 4x–10x faster than previous approaches such as BERT, resulting in excellent performance on natural language understanding tasks both when trained at large scale or even when it is trained on a single GPU.

Acknowledgements

This dissertation would not have been possible without the support of many wonderful people. First and foremost, I would like to thank my advisor Chris Manning. I deeply admire his ability to be insightful about both the low-level details and big-picture ideas of research projects; it was truly a privilege to learn from him. Thank you, Chris, for giving me the freedom to work on projects I was excited about, for direct yet kind advice, and for wisdom on many non-research topics: writing, teaching, career development, and much more.

A great feature of Stanford's computer science Ph.D. program is the rotation system, which gave me the privilege of working closely with other faculty. Thank you to my rotation advisor Dan Jurafsky, who continued to guide me throughout my Ph.D. and served on my thesis committee, for your infectious curiosity and enthusiasm. Your wealth of knowledge has always inspired me to step back and take a broader view of the field beyond the latest trendy papers. And thank you to my other rotation advisor Jure Leskovec for showing me how to find direction in a broad range of possibilities when working on analysis projects.

I feel very fortunate to have spent an incredible three and a half years working at Google Brain as an intern or part-time student researcher. Thank you to Quoc Le and Thang Luong, my mentors at Brain, for pushing me to be ambitious, forward-thinking about the field, and large scale in my research ideas. Quoc and Thang collaborated with me on pretty much every project in this dissertation and were instrumental in developing many key ideas. Quoc also generously served on my committee – I guess despite joining an industry research lab, you didn't entirely escape academia! Thanks also to Noam Shazeer and Colin Raffel for helpful discussions about ELECTRA. I'm also very grateful to Google AI for funding three years of my Ph.D. through their Ph.D. Fellowship program.

I want to thank my many other collaborators as well. Tim Althoff, Will Hamilton, Urvashi Khandelwal, Omer Levy, and John Hewitt: all of you have been a joy to work with. I've learned so much from you, and it has undoubtedly made me a better researcher. I'm looking forward to seeing each of you continue to do remarkable things. I consider myself very lucky to have been part of Stanford's NLP group. It is not just an outstanding academic environment, but also a very supportive one with many kind people, which I think is even more important. Thanks especially to Gabor Angeli, Jon Gauthier, Will Monroe, Kelvin Guu, Peng Qi, Reid Pryzant, Yuaho Zhang, Abi See, Robin Jia, John Hewitt, Allen Nie, and Ashwin Paranjape for reading my paper drafts and making them much better.

My time at Stanford has been fantastic not only for academics, but also for the amazing friends I made. So thank you to my friends, whose encouragement and support have been invaluable, especially Jason Koenig, Raphael Townshend, Maithra Raghu, Ranjay Krishna, Daryl Hanson, and Peter Van Hoomissen. Completing a Ph.D. isn't an easy journey, and there were times when looking forward to skiing, road trips, or board games helped keep me going.

I couldn't imagine getting here without the love and support of my amazing family. My parents Peter Clark and Marianne Schneider instilled in me a love of learning and exploring, which eventually led me to do a Ph.D. I am fortunate that my dad and brother Chris are experts in the field (I still have no idea if my interest in AI is from nature or nurture), so they were able to offer me research advice as well as emotional support! And thanks to my sister Rachel for your humor, positive attitude, and willingness to suffer through our research conversations without *too* much complaining. To Arathi, who I met on one of my first days at Stanford and who has been my best friend and "partner in crime" ever since: thank you for everything.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Language is a fundamental feature of human intelligence, so unsurprisingly a large subset of research on artificial intelligence (AI) has focused on improving the ability of Natural Language Processing (NLP) systems. With much of humanity's knowledge captured in the form of text, NLP offers a way to imbue AI systems with information about the world. Building systems that can intelligently process the incredible volume of text available on the web also has important practical value: NLP systems could help people navigate, filter, and understand this ever-growing body of data. Over the past ten years, supervised deep learning approaches have led to impressive improvements on classic NLP tasks such as parsing (Chen and Manning, 2014), named entity recognition (Chiu and Nichols, 2016), and coreference resolution (Wiseman et al., 2015). These methods train multi-layer neural networks that automatically learn a hierarchy of features and thus reduce the need for feature engineering. Neural networks have far more parameters than statistical systems, making neural networks highly expressive such that they can precisely fit their training data.

However, a big gap remains between these systems and human learning. While supervised neural networks work well on in-distribution data, the success of these systems is often narrow and brittle. Their expressivity can be a downside because they learn idiosyncrasies and artifacts of their training dataset rather than generalizable patterns. For example, Jia and Liang (2017) show that small content-preserving changes to the test data can cause these models to fail, and Gururangan et al. (2018) show that even models achieving high accuracy on natural language understanding benchmarks may in fact be primarily learning simple superficial patterns. Furthermore, the manner in which these models learn is far from human development. These systems are trained on large numbers of examples that have been annotated by a human expert, learning a function that maps inputs to the annotations. For every task, such as analyzing the syntactic structure of sentences, answering questions, or translating documents, a new model is trained from scratch. In contrast, humans learn in a general and multi-task way; when learning something new, we can draw on our prior experiences and apply them to learn quickly (Luriia, 1976; Perkins et al., 1992). Additionally, humans often learn without direct feedback, whereas supervised models require many examples with the labels provided.

These differences between machine and human learning point to considerable limitations in current NLP systems. Since they learn from scratch, models have to learn everything about language from the often small and un-representative dataset they are provided. Furthermore, without any prior knowledge to draw on, NLP systems are sample-inefficient, requiring large amounts of data to learn effectively. Even with large labeled datasets, these systems can still have big gaps in their capabilities. For example, in question answering, large datasets containing thousands of questions have been collected (Rajpurkar et al., 2016). However, the amount of knowledge contained in these datasets is still a minuscule fraction of the total amount held in human writing. We can't expect an NLP model to cover the broad range of questions someone may want answered without having access to much more information than is available through only labeled datasets.

1.2 Transfer Learning

In this thesis, I argue that a solution to these limitations of supervised learning is *transfer learning* and develop methods to make transfer learning more effective. Broadly, transfer learning methods train the NLP model on additional data sources beyond examples annotated for the task of interest. This data can be annotated for a different but usually related task (multi-task learning), or be a large source of unlabeled data such as text from the web (self-supervised and semi-supervised learning). Instead of learning a single task from scratch and in isolation, the information the model acquires from the other data source can be transferred to the task, resulting in more efficient and effective learning. These models are less limited by the availability of data because they can learn from the vast wealth of knowledge on the web or from other tasks with rich annotations. Transfer learning is especially appealing in conjunction with neural networks because neural networks work thrive when provided large datasets. The central goals of transfer learning are to produce *general* systems that can perform many tasks and *generalizable* systems that work on diverse data beyond a small labeled domain. This section provides an overview of commonly used transfer learning methods; I cover these techniques in more depth in Sections 2.3, 2.4, and 2.5.

Perhaps the most direct way of training a general-purpose NLP system is through *multi-task learning*, where the model learns from multiple labeled datasets. Ideally, multi-task learning not only results in a more broadly applicable system, but also a better one because learning one task can provide the model knowledge that improves its ability on related tasks. This moves the model, at least to a small extent, towards the kind of human learning where drawing on prior experiences let us quickly develop new skills. Multi-task learning can improve the generalization of models by exposing them to diverse inputs so they are less likely to overfit to the task data. It introduces an inductive bias provided by the auxiliary tasks, which causes the model to prefer hypotheses that explain more than one task. Early approaches to multi-task learning did *feature selection* where features useful for multiple tasks are preferred (Obozinski et al., 2006) or *joint prediction* where related linguistic structures are predicted simultaneously. Structures predicted earlier can inform later ones by providing features (e.g., part-of-speech tags), or constraints (e.g., a named entity cannot span different subtrees of a syntactic parse). For example, Durrett and Klein (2014) train a model to jointly perform coreference resolution, entity typing, and entity linking. These tasks are beneficial because they all involve modeling entities in a document.

Another direction in transfer learning research has been *semi-supervised learning*, which aims to use unlabeled data to improve task performance. The key idea is that models can generalize the patterns in the training data to the unlabeled data and thereby be able to learn new patterns. Semi-supervised learning is more scalable than multi-task learning because it is not limited by the size of human-labeled datasets and can take advantage of the abundance of text on the web. One of the earliest approaches to semi-supervised learning is self-training (Scudder, 1965), which has been successfully applied to NLP tasks such as word-sense disambiguation (Yarowsky, 1995) and parsing (McClosky et al., 2006). In each round of training, the classifier, acting as a "teacher," labels some of the unlabeled data and adds it to the training set. Then, acting as a "student," it is retrained on the new training set. This procedure can be repeated iteratively until the model converges, gradually adding more examples to the training set and expanding the range of examples the model sees; self-training hopes for a virtuous cycle where each iteration improves the model's accuracy. While these self-labeled examples will be noisy, the hope is that the model will still be able to learn from them and "smooth out" the noise by processing a large enough volume of data.

The rise of neural networks has greatly enhanced the ways in which transfer learning can be used. For multi-task learning, neural networks offer a more powerful and direct means of cross-task transfer beyond feature selection or joint training: the representations built up by the model can be shared across tasks. A learning update that improves the representations so they are more effective for one task will hopefully improve the representations in general, resulting in a stronger model. Shared features have to be useful for multiple tasks, causing the model to "focus" on more generalizable features and ignore features less useful for other tasks. Essentially sharing parameters focuses the hypothesis space of the model by encouraging the features to be useful for more tasks. The particular choice of which parameters are shared is a widely researched design decision, often using linguistic motivations (Søgaard and Goldberg, 2016; Hashimoto et al., 2017) or learned sharing schemes (Misra et al., 2016; Ruder et al., 2019; Liu et al., 2017).

Neural networks also enable a highly effective way of using unlabeled data: *self-supervised pre-training*. Self-supervised pre-training constructs what looks like a supervised task from the unlabeled data, such as deleting some input words and training the model to predict what they were originally. These methods typically operate in two phases. First, in the pre-training phase, the model performs self-supervised representation learning over unlabeled corpus. Then during a fine-tuning phase, the learned model is transferred to the task of interest. In particular, the pre-trained weights are used as initialization for a task-specific classifier, with usually a small additional randomly initialized neural network component added on top to make the final classification decisions. In pre-neural semi-supervised learning methods such as self-training, the model is limited to learning a pre-specified set of tasks; to build a model for a new task, the expensive procedure of learning over the large unlabeled corpus has to be repeated again. In contrast, self-supervised representation learning uses unlabeled data in a more efficient and general-purpose way, developing general representations that can then be transferred to a variety of tasks. Once the slow self-supervised learning phase has been performed once, the model can then be reused for many tasks.

Early self-supervised pre-training approaches in NLP such as word2vec (Mikolov et al., 2013b) and GloVe (Pennington et al., 2014) learned word embeddings: low-dimensional vector representations of words. Pre-trained word embeddings quickly became standard use by the NLP community. Although learned purely from self-supervised learning, they demonstrate a remarkable ability to capture lexical features. While pre-trained embeddings have been highly successful, these methods produce a fixed representation of each word. However, in practice language is highly contextual, with the meaning of a word depending on its neighbors. For this reason, it can be more effective to pre-train neural networks such as Transformers and LSTMs that build *contextual* representations of word tokens rather than static representations of work types.

Two key pre-training tasks for training models producing contextual word representations are language modeling and cloze modeling. Language models such as ELMo (Peters et al., 2018) predict the next word in an utterance based on its left context: the preceding words. Cloze models such as BERT (Devlin et al., 2019) predict a word given both its left and right context: the surrounding words. In practice, the cloze task is usually implemented by replacing input words with a special `[MASK]` token and then training the model to reconstruct the masked-out words. These tasks are highly effective for representation learning because they require modeling syntax and semantics to solve. For example, solving subject-verb number agreement (e.g., predicting "the chef who made the pizzas" is followed by "is" instead of "are") requires modeling the hierarchical structure of the sentence. In other cases, commonsense reasoning or world knowledge is required, for example in predicting "small" follows "It didn't fit because the suitcase was too" or "France" follows "Paris is the capital of". Indeed, analysis of pre-trained models shows that their representations capture key aspects of syntax (Linzen et al., 2016; Blevins et al., 2018; Hewitt and Manning, 2019), entity recognition (Tenney et al., 2018; Liu et al., 2019c), and coreference (Tenney et al., 2019), even though they receive no explicit supervision for the tasks. While once these linguistic features were added to systems through training human-annotated datasets, self-supervised learning approaches have reduced the need to develop systems for these tasks previously considered a core part of NLP.

The rise of self-supervised pre-training has been transformative to NLP. Not unlike how neural methods led to a decrease in engineered features, self-supervised learning combined with powerful neural network architectures such as the Transformer (Vaswani et al., 2017) have to a large extent ended task-specific engineered architectures because a single pre-trained model can be effectively transferred to many tasks. A key advantage of self-supervised methods is that they scale very well. Bigger neural nets generally perform better when enough data is available. However, when learning from small supervised datasets, large models start overfitting and the performance gains stop. In contrast, pre-training methods can leverage vast amounts of unlabeled text, enabling the use of more expressive neural networks with many parameters.

This excellent scaling has led to increasingly large models being pre-trained with increasingly large amounts of compute (see Figure 1.1), resulting in incredible progress on NLP benchmarks. However, this trend also makes models expensive to train, decreasing the usability and accessibility of transfer learning. Recent models require hundreds of GPU years to train, far more compute than most academic research labs or NLP practitioners have access to. These requirements limit the development of new pre-training methods or the ability to apply them to specific domains and languages. With AI systems increasingly performing higher-stakes tasks and becoming more prevalent in the world, democratizing AI so that any person or group can contribute is becoming increasingly important. Due to these concerns, a central goal of this thesis is to develop transfer learning methods that are *efficient* and work well when using relatively little compute (i.e., that are towards the left of Figure 1.1 as well as high up).

I argue that efficiency is particularly important for highly scalable methods like recent transfer learning approaches. Figure 1.1 shows that existing methods mostly lie on the same curve trading off compute for downstream task performance. This raises a question: are we actually developing better methods, or are we just throwing more compute at the models? A way to demonstrate clear progress is through achieving better downstream task performance given the same compute use; in other words, through improving efficiency. If an efficient method scales as well as these existing approaches, it effectively raises up the compute vs. accuracy and improves results across the full spectrum of model sizes. From this perspective, my dissertation develops transfer learning methods that are scalable (i.e., also able to benefit from large amounts of compute and data), while still remaining efficient (i.e., making good use of compute resources and not requiring the massive scale of by other methods to be effective) to produce general and effective NLP systems.

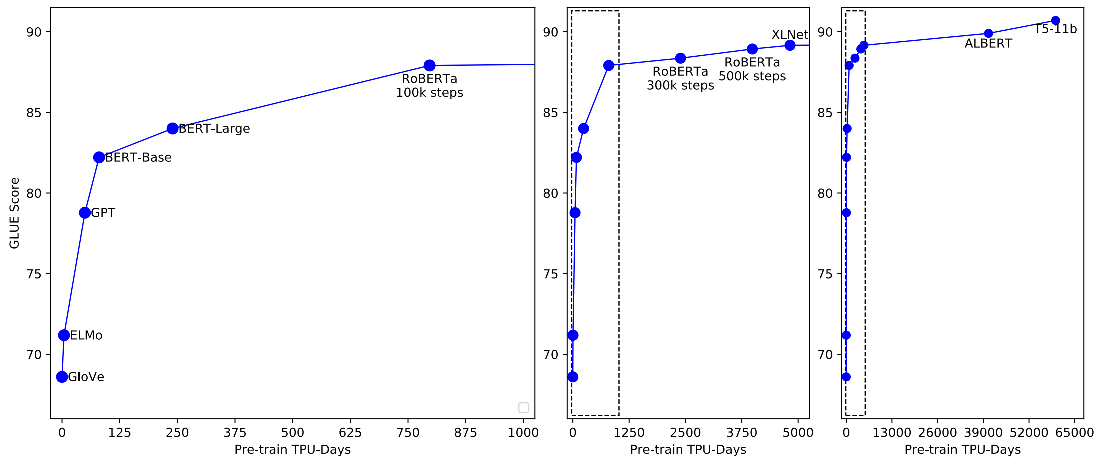


Figure 1.1: Scores on the GLUE natural language understanding benchmark vs training times of large-scale NLP models. Each figure is a zoomed-in version of the dashed box to its right. TPU-Day estimates come from computing the total floating point operations required to train each of these models and then scaling according to TPUv3 compute. A conservative estimate of human performance is around 90 points (Nangia and Bowman, 2019). Self-supervised pre-training methods, starting in 2018 with ELMo, have completely closed the gap from far-below-human performance to “superhuman” performance. However, this progress has largely been driven by massively scaling up the compute used to train these systems.

1.3 Research Goals

Broadly, this thesis studies using transfer learning to train effective representations of language that can be applied to a diverse range of NLP tasks. Here I lay out three goals that will be addressed by the approaches proposed in this thesis.

1. **Generality:** I aim to develop methods that are broadly applicable across a range of NLP settings. Designing task-specific training algorithms and architectures can lead to improved results. However, I believe they make little progress towards the sort of general language understanding humans are capable of. They rely on specific details of the task and baseline method, which often cannot be easily extended to other settings. In contrast, human learning is remarkably general, with the connection between different subjects being learned rather than baked into the underlying hardware of the brain. On the more practical side, improving a general method can lead to gains across many tasks, or even future tasks that aren't currently being worked on, instead of just the single one the method is designed on. Due to these concerns, I propose transfer learning approaches where the transfer can aid many tasks rather than just one.

2. **Scalability:** Transfer learning methods should scale well to large amounts of data and compute. That is, they should continue to produce better results as model sizes and datasets get larger. A scalable method will continue to get better "for free" as more data and compute becomes available. This is important because computing hardware has consistently been improving over the past decades, and in all likelihood this trend will continue. Being able to take advantage of future advances in hardware helps ensure the method will be long-lived and continue to remain relevant. Transfer learning has enabled highly scalable training by vastly increasing the amount of data available to models so they are no longer being limited by the amount of data available for a specific task. My thesis research therefore develops methods that work well as models get larger, the number of tasks increases, and the training dataset gets larger.

3. **Efficiency:** Scalable methods benefit from as much compute and data as possible,

but these resources are expensive. Consequently, it is important to develop efficient methods that get as much value from their compute resources as possible. Historically, efficiency and accuracy have been considered orthogonal research goals, with efficiency research aiming to achieve accuracy close to state-of-the-art methods while requiring less compute. However, the scalability of transfer learning makes efficiency and accuracy two sides of the same coin; improving efficiency means models are trained as though they are using more compute resources, which leads to consistent gains in accuracy. Another important aspect of efficiency is accessibility. Models costing thousands or even millions of dollars to train are beyond the reach of most people working on NLP. Having models that can be trained by a academic researcher or machine learning practitioner working on a narrow domain or low-resource language is crucial for ensuring equitable usage of AI. Lastly, in addition to the financial cost, the environmental impact of training large NLP models is considerable (Strubell et al., 2019), and improving efficiency alleviates this cost.

1.4 This Dissertation

This thesis aims to realize these goals across the three core types of transfer learning I discussed above:

- *Multi-task learning*, where a single model learns from human-labeled data to perform multiple distinct NLP tasks. As a single neural network is using the same representations for all tasks, knowledge across related tasks can be shared to improve performance.

- *Semi-supervised learning*, where the model learns from large amounts of unlabeled data as well as human-annotated labeled data. Semi-supervised methods can generalize patterns learned from the labeled data to the unlabeled data to gain better coverage of the task.

- *Self-supervised learning*, where the model learns purely from unlabeled data to develop effective representations for text. These representations can then be fine-tuned on specific tasks of interest.

While self-supervised learning methods have recently become dominant compared to other transfer learning types, I believe each of them still has high-value use cases. For example, recent work shows that multi-task learning (Aghajanyan et al., 2021) and semi-supervised learning (Du et al., 2020) improve pre-trained model when used at a large enough scale. In this dissertation, I present a concrete system in each area of transfer learning that achieves excellent scalability and computational efficiency while being applicable to a general set of tasks.

Chapter 3: Born-Again Multi-Task Networks (BAM). Building a single model that jointly learns to perform many tasks effectively has been a long-standing challenge in NLP. Having one model with a diverse set of capabilities moves towards a more human-like way of doing NLP. Furthermore, multi-task models also have practical advantages. In the pursuit of goal 3 (efficiency), developing, deploying, and running a single multi-task model can be simpler and faster than doing so with many independent models. However, existing work in multi-task NLP has had mixed results, with multi-task models often performing worse than their single-task counterparts (Plank and Alonso, 2017; Bingel and Søgaard, 2017). While similar pairs of tasks can benefit each other, models performing many tasks exhibit negative transfer where tasks interfere rather than bolstering each other during learning. Moving towards goal 1 (generality), I develop a new multi-task learning algorithm called Born-Again Multi-tasking (BAM) with *robust* multi-task learning capabilities. Here "robust" means the multi-task model is never worse than its single-task counterparts, even across many diverse tasks, but still achieves gains across tasks where multi-tasking is helpful. I achieve this goal by applying knowledge distillation (Ba and Caruana, 2014; Hinton et al., 2015) so that single-task models effectively teach a multi-task model.

Knowledge distillation transfers knowledge from a "teacher" model to a "student" model by training the student to imitate the teacher's outputs. In "born-again networks" (Furlanello et al., 2018), the teacher and student have the same neural architecture and model size, but surprisingly the student is able to surpass the teacher's accuracy. My work extends born-again networks to the multi-task setting. In particular, I first train a single-task model for each task and then distill them into a multi-task one. Intuitively, the multi-task model will never perform worse than the single-task one when it can learn directly how the single-task one performs the task. I enhance this distillation with a simple teacher-annealing method that helps the student model outperform its teachers. Teacher annealing gradually transitions the student from learning from the teacher to learning from the gold labels. This method ensures the student gets a rich training signal early in training, but is not limited to only imitating the teacher.

In my experiments, I apply BAM to train large models built on top of BERT. I start from BERT in order to show the scalability of my method along the vein of goal 2, showing that the method works even on top of a large and already very strong model. BAM outperforms both standard single-task and multi-task training across 10 diverse natural language understanding tasks. Importantly, the gains are consistent, with BAM almost never performing worse than single-task models. Further analysis shows the multi-task models benefit from both better regularization and transfer between related tasks. This work was first presented at ACL 2019 in *BAM! Born-Again Multi-Task Networks for Natural Language Understanding* (Clark et al., 2019b).

Chapter 4: Cross-View Training (CVT). While multi-task learning methods can improve models, they are limited in requiring labeled data from related tasks, hindering them from realizing goals 1 and 2 (generality and scalability). Cross-view training is a semi-supervised learning method that instead can make use of lots of unlabeled data. CVT extends the idea of self-distillation to operating across different *views* of the input sequence. It revitalizes old semi-supervised learning methods from the statistical era of NLP (self-training and co-training) while taking advantage of neural methods.

In self-training, the model learns as normal on labeled examples. On unlabeled examples, the model acts as both a "teacher" that makes predictions about the examples and a "student" that is trained on those predictions. Although this process has shown value for some tasks, it is somewhat tautological: the model already produces the predictions it is being trained on. Recent research on computer vision addresses this by adding noise to the student's input, training the model so it is robust to input perturbations (Sajjadi et al., 2016; Wei et al., 2018). However, applying noise is difficult for discrete inputs like text. As a solution, I take inspiration from multi-view learning (Blum and Mitchell, 1998; Xu et al., 2013) and train the model to produce consistent predictions across different *views* of the input. The “student” parts of the model see only part of the input (e.g., only the left context of a word), so they can learn from the teacher, which sees the whole input. Crucially, the student and teacher have shared parameters, so as the student learns the teacher will improve because it is built on the same representations.

More specifically, CVT trains the model on a mix of labeled and unlabeled examples. On labeled examples, standard supervised learning is used. On unlabeled examples, CVT teaches auxiliary prediction modules that see restricted views of the input (e.g., only part of a sentence) to match the predictions of the full model seeing the whole input. These auxiliary prediction modules are essentially softmax layers placed on top of the main text encoder. I take advantage of the network structure (in my case a Bi-LSTM) to create the different views. For example, one auxiliary prediction module for sequence tagging is attached to only the forward LSTM in the model’s first Bi-LSTM layer, so it makes predictions without seeing any tokens to the right of the current one. The auxiliary prediction modules can learn from the full model’s predictions because the full model has a better, unrestricted view of the input. Since the auxiliary modules and the full model share intermediate representations, this in turn improves the full model.

Importantly, the unlabeled data can be from a different source than the original examples, making CVT highly scalable (goal 2). CVT also offers efficiency advantages over self-supervised methods such as ELMo and BERT (goal 3). With CVT, the model learns representations targeted towards a particular task rather than general-purpose ones, meaning the model can be smaller and trained on less data because its target is narrower. CVT can be effectively combined with multi-task learning to perform well at many tasks (goal 1), with the unlabeled data being used to learn all the tasks in parallel in a highly efficient way. In fact, a model with a similar neural architecture to ELMo can be trained for 10% the time while still performing better across a diverse set of tasks, including dependency parsing, named entity recognition, and machine translation. This work was first presented at EMNLP 2018 in *Semi-Supervised Sequence Modeling with Cross-View Training* (Clark et al., 2018).

Chapter 5: Efficiently Learning an Encoder that Classifies Token Replacements Accurately (ELECTRA). While CVT yields models that perform well on many tasks, it still requires the set of tasks to be specified in advance, limiting the models' generality (goal 1). Furthermore, CVT requires roughly in-domain data, making it difficult to apply to some tasks such as question answering, where unlabeled examples are not easy to come by. Self-supervised learning methods offer a solution, where the model learns as general representations as possible from unlabeled data alone. An efficiency advantage over standard semi-supervised learning is that the long self-supervised pre-training phase only has to be done once and then the model can be adapted to individual tasks relatively quickly (goal 3).

BERT is a particularly notable self-supervised method due to its improved performance across many tasks. BERT is trained with the masked language modeling task, where it corrupts the input by replacing some tokens with `[MASK]` and then learns to reconstruct the original tokens. It scales very well (goal 2) and is transferable to many tasks (goal 1) because of the richness of this task; different word deletions can require many different linguistic abilities to solve. However, this generality has a downside: BERT requires huge amounts of compute and unlabeled text to be effective, limiting its efficiency (goal 3). The main efficiency drawback of BERT comes from masking. Masking too many tokens causes the input to become corrupted so that predictions are difficult and the input no longer looks like real text. Therefore BERT only masks out 15% of tokens per input sequence. In contrast, the earlier pre-training task of language modeling predicts every input token, providing them much more training signal per example. However, language models autoregressively generate text left-to-right, meaning they are "unidirectional" in that they see only the text to one side of the token they are predicting. I develop a new pre-training method called ELECTRA that learns from every input token while still being bidirectional, getting the best of both worlds. ELECTRA learns 4–10x faster than BERT, a substantial gain in compute efficiency.

Underlying ELECTRA is a sample-efficient pre-training task called replaced token detection that learns much faster than BERT. Instead of masking the input, my approach corrupts it by replacing some tokens with plausible alternatives sampled from a small generator network. Then, instead of training a model that predicts the original identities of the corrupted tokens, I train a discriminative model that predicts whether each token in the corrupted input was replaced by a generator sample or not. Crucially, this binary classification task is applied to every input token, instead of only a small number of masked tokens, making replaced token detection more efficient than masked language modeling — the model needs to see fewer examples to achieve the same performance because it receives more training signal per example. At the same time, replaced token detection results in powerful representation learning, because the model must learn an accurate representation of the data distribution in order to solve the task. Indeed, theoretical analysis shows that replaced token detection can be viewed as training an energy-based model to perform the same cloze modeling tasks that yields such rich representations learning in BERT.

Thorough experiments demonstrate replaced token detection is more efficient than masked language modeling because the task is defined over *all* input tokens rather than just the small subset that was masked out. As a result, the contextual representations learned by my approach substantially outperform the ones learned by BERT given the same model size, data, and compute. The gains are particularly strong for small models; for example, I train a model on one GPU for 4 days that outperforms GPT (trained using 30x more compute) on the GLUE natural language understanding benchmark. However, the method also works well at scale, where it improved the state-of-the art across several natural language understanding tasks, including question answering. This work was presented at ICLR 2020 in *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators* (Clark et al., 2020a) and in EMNLP 2020 in *Pre-Training Transformers as Energy-Based Cloze Models* (Clark et al., 2020b).

Chapter 2

Background

This chapter provides background material for the subsequent chapters. It first reviews a selection of Transformer networks and knowledge distillation, important deep learning techniques that will be used throughout this dissertation. It is not meant to be a comprehensive overview of deep learning: for that, I refer the reader to (Goodfellow et al., 2016). Next, it provides an overview of prior work on transfer learning, with an emphasis on applications to natural language processing. It covers three of the primary kinds of transfer learning: using additional labeled data (multi-task learning), using unlabeled data to improve task performance (semi-supervised learning), and using unlabeled data to learn general representations that can be applied to a variety of tasks (self-supervised learning). Lastly, it gives an overview of common natural language processing tasks and associated datasets, which will be used in experiments throughout the rest of this dissertation.

2.1 Transformer Networks

Transformer networks (Vaswani et al., 2017) are highly expressive models for processing sequences. They have achieved excellent results across computer vision (Carion et al., 2020), reinforcement learning (Parisotto et al., 2020), and speech recognition (Dong et al., 2018), as well as in NLP@İ use Transformers for the multi-task learning and self-supervised pre-training parts of this dissertation.

The key modeling challenge that Transformers address is capturing long-distance dependencies, which can be crucial for modeling text. Before Transformers, most neural models in NLP were based on recurrent neural networks (RNNs). RNNs process the input one token at a time, at each step updating the representation of the sequence seen so far. While appealing in that they can "read" text left to right the way humans do, capturing the entire context read so far in a single vector has limitations. Analysis shows that the models "forget" information from far away (Khandelwal et al., 2018) and generally perform worse as sentences get longer (Bahdanau et al., 2015). Gating mechanisms such as long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) can alleviate this disadvantage, but only partly.

Transformer networks remove recurrence entirely from the network. Instead, it relies on self-attention to model context. When computing the next representation for a particular token, self-attention directly references the current representation of every other token in the input. As a result, Transformers model long-distance dependencies well because they draw from all tokens regardless of distance. They do not suffer from the problems of vanishing and exploding gradients common in RNNs because the attention causes short gradient paths. Although self-attention treats all tokens equally, positional information about tokens is still captured by Transformer networks through simple positional embeddings concatenated to each token embedding. The expressivity of Transformers does have a downside, which is that they generally don't perform as well in data-poor regimes. Generally, recent context is most important in language, and Transformers lack the inductive bias favoring recency that RNNs have. However, Transformers are well suited for the transfer learning method in this thesis since they are highly scalable.

Transformers were first applied to sequence-to-sequence learning, but in this dissertation I only use Transformer encoders (I refer to these as "Transformers" for brevity) that operate on a single sequence. A Transformer network maps a sequence of input tokens $\boldsymbol{x} = [x_1, x_2, ..., x_n]$ to a sequence of vector representations $\boldsymbol{h} = [h_1, ..., h_n]$. It consists of n blocks stacked on top of each other. An overview of a Transformer is shown in Figure 2.1. Details of full encoder-decoder Transformer networks are similar to those described below. However, an encoder-decoder Transformer has a decoder network that generates the target sequence and contains an additional attention mechanism which attends over the source sequence. This decoder uses causal attention masking, where tokens are prevented from attending to future tokens, to enforce left-to-right generation. This same attention masking is used to train Transformer language models.

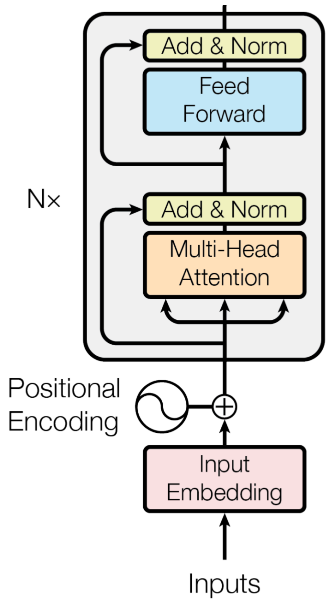


Figure 2.1: The Transformer neural architecture. The model consists of input/positional embeddings followed by N self-attention and feedforward blocks. Figure from (Vaswani et al., 2017).

Input Encoding. Before being passed into the Transformer, the input is first converted into a sequence of embeddings. The input words are tokenized using a fixed vocabulary, either representing rare words with a special `UNK` token, or more commonly using a word piece vocabulary (Sennrich et al., 2016a) that splits rare words into multiple sub-words. Then each token is represented as the concatenation of its token embedding and a position embedding. The position embedding provides the model with the ordering of the sequence. There are many choices for position embeddings, but I generally use a learned vector for each position.

Multi-Headed Attention. The key mechanism by which Transformers contextualize representations is *multi-headed attention* (see Figure 2.2). Attention (Bahdanau et al., 2015) dynamically assigns a weight to every pair of words in the sequence, indicating how much the model should "pay attention to" the first word when computing the representation of the second. Attention has been a highly successful neural network component for processing text (Luong et al., 2015b), video (Sharma et al., 2016), image (Xu et al., 2015), and speech (Chorowski et al., 2015) data.

Transformers use multiple attention heads in parallel, where each head can potentially capture a completely different word-word relation. Transformers aggregate the information from each head to produce a single output vector representation for each word in the sequence. I provide more mathematical detail below.

Given a query q and a set of n items $\{x_1, ..., x_n\}$, the attention mechanism Bahdanau et al. (2015) induces a probability distribution α over the item set, and then produces an expectation (weighted sum) of the items as the output. Intuitively, attention makes a soft selection of which item x_i is the best fit for the query q. More formally, each item x_i is represented by two vectors: key k_i and value v_i. The key k_i is used to determine the distribution α via an inner product with the query vector q, normalized by a softmax:

$$\alpha_i = \frac{\exp q^\top k_i}{\sum_{\ell=1}^{n} \exp q^\top k_\ell}$$

The output y is then the expectation over the value vectors v_i:

$$y = \sum\nolimits_{i=1}^{n} \alpha_i v_i$$

The Transformer network (Vaswani et al., 2017) uses a flavor of this mechanism called *self-attention*, where each input word plays a dual role as both a query and a selectable item. This is implemented by passing the vector representation of every word x_i through three different linear transformations, resulting in query q_i, key k_i, and value v_i vectors. Each query q_j can then attend over all the key-value pairs (k_i, v_i) in the sequence, producing a different attention distribution α^j (i.e., α_i^j denotes the attention weight towards position i from position j) and output y_j for each word, as shown in Figure 2.2.

Each head computes its own independent attention weights and output vectors; the output vectors across heads are concatenated when producing a layer's output.

Feed Forward Layer. In addition to the self-attention sub-layer, each block contains a

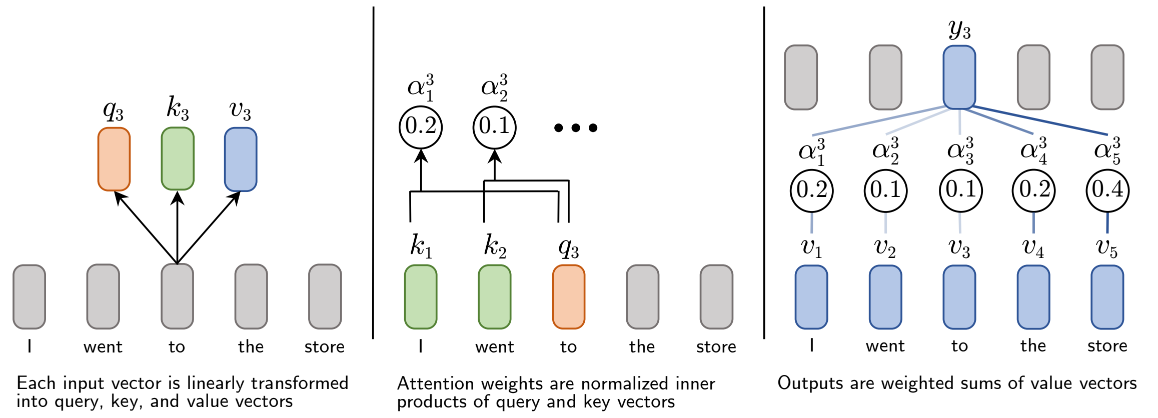

q_3
k_3
v_3
I
went
to
the
store
Each input vector is linearly transformed into query, key, and value vectors
α_1^3
α_2^3
0.2
0.1
k_1
k_2
q_3
Attention weights are normalized inner products of query and key vectors
y_3
α_3^3
α_4^3
α_5^3
0.4
v_1
v_2
v_4
v_5
Outputs are weighted sums of value vectors

Figure 2.2: An overview of the neural attention mechanism. For simplicity, here I only show a single attention head applied to a single position in the input sequence. Attention induces weights over the previous hidden states, causing the model to "pay attention to" salient information in the context.

fully connected feedforward network. The network consists of two layers, typically with a ReLU (Nair and Hinton, 2010) activation:

$$\text{FFN}(x) = W_2\text{ReLU}(W_1x + b_1) + b_2$$

The intermediary layer's size generally is larger than the input size, providing an expressive transformation while not making the other parts of the block too expensive.

Layer Norm and Residual Connection. The self-attention and feed-forward components both make use of layer normalization and residual connections. Layer normalization (Ba et al., 2016) stabilizes training by "standardizing" the layer's output so it has zero mean and unit variance. The residual connection improves gradient flow by adding a "shortcut" from the layer's input to its output, allowing the network to be deep without having exploding or vanishing gradients (He et al., 2016). If the component's input is a d-dimensional vector x

and its output is denoted as $f(x)$, the transformation is

$$\text{Output} = \text{LayerNorm}(f(x) + x) \quad \text{where}$$

$$\text{LayerNorm}(x) = a(x - \mu)/\sigma + b \qquad \mu = \frac{1}{d}\sum_{i=1}^{d} x_i \qquad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

where a and b are trainable scalars adding a learned bias and scaling factor to the layer.

2.2 Knowledge Distillation

Having discussed the primary neural architecture used in this dissertation, this section switches gears towards an important training technique for neural networks. Knowledge distillation (Ba and Caruana, 2014; Hinton et al., 2015) is a method for transferring knowledge from a "teacher" neural network to a "student" network. The key idea is to train the student to imitate the teacher's output distribution. Knowledge distillation can be applied to train an effective small model by having it learn from an accurate but computationally expensive model, such as an ensemble or model with many parameters. In this dissertation, I apply knowledge distillation to improve multi-task models by having single-task models act as teachers. I also use a form of "self-distillation" as a semi-supervised learning method.

For simplicity, I present knowledge distillation for classification problems. Let $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$ denote the training set for the classification task and $f_\theta(x)$ denote the output distribution over classes produced by a neural network with parameters θ on the input x. This distribution f_θ is a k-dimensional vector with elements summing to 1 where k is the number of classes, typically produced by a softmax layer. Standard supervised learning trains θ to minimize the loss on the training set:

$$\mathcal{L}(\theta) = \sum_{x_i, y_i \in \mathcal{D}} \ell(y_i, f_\theta(x))$$

where for classification tasks ℓ is usually cross-entropy.

Knowledge distillation trains the model to instead match the predictions of a teacher model $f'_{\theta'}$ with parameters θ':

$$\mathcal{L}(\theta) = \sum_{x_i, y_i \in \mathcal{D}} \ell(f'_{\theta'}(x), f_\theta(x))$$

This teacher network is typically trained with standard supervised learning over the data.

Interestingly, knowledge distillation improves over standard supervised training. This finding is surprising because the targets provided by the teacher network are not even guaranteed to be correct the way gold labels are. Perhaps even more surprising is the phenomenon of self-distillation. Training a network θ' and then distilling it into a new network θ improves the second network even if it has exactly the same neural architecture and hyperparameters as θ'!

Furlanello et al. (2018) investigate the surprising effectiveness of these "born-again networks" and point to two explanations. First, distillation is effective because the teacher's output distribution over classes provides more training signal than a one-hot label; Hinton et al. (2015) suggest that teacher outputs contain "dark knowledge" capturing additional information about training examples. Secondly, the output distribution implicitly provides weights for examples: on more challenging examples where the correct class isn't obvious, the model is trained to be more uncertain, with the teacher putting less weight on the correct class in its output distribution.

2.3 Multi-Task Learning

Machine learning models are generally trained to perform a single task. As discussed in the introduction, this paradigm incurs significant drawbacks in terms of *generality* (the model only learns a single task and is unlikely to perform well in other settings) and *scalability* (the model is limited by the potentially small amount of annotated data available for the task). Multi-task learning enables models to benefit from the knowledge that comes from the training signal of related tasks. Compared to single-task learning, it is also closer to the kind of learning that humans do, where drawing on prior experiences lets us quickly learn new skills. From a more practical perspective, having a single multi-task model can be easier to develop and deploy than having many single-task models. Multi-task models are at less risk for over-fitting than single-task ones because they are exposed to many diverse inputs. Caruana (1997) argues that multi-task learning provides a valuable *inductive bias* to the model. An inductive bias is anything that causes the learner to prefer some hypotheses over others. In the case of multi-task learning, the features or representations learned are encouraged to be useful for many tasks. As a result, features specific to particular tasks which are not likely to generalize well (e.g., dataset artifacts) will be less used by the model.

Pre-Neural Methods. A simple use of multi-task learning is for feature selection (Obozinski et al., 2006), where features useful for multiple tasks are preferred. However, in NLP a more common multi-task learning method is *joint prediction*. These methods train the model to predict multiple related structures. Information from a predicted structure can improve the prediction of other ones through new features (e.g., knowing a word is a proper noun makes it more likely to be a named entity) or hard constraints (e.g., a named entity cannot cross phrase boundaries). For example, Durrett and Klein (2014) train a model to jointly perform entity linking, entity typing, and coreference resolution. Entity types can improve coreference because mentions of different types will not be coreferent (e.g., the company "Freddie Mac" and the personal pronoun "him"). Similarly, coreference can improve entity typing because coreference decisions can provide new information about an entity (e.g., knowing that "Dell" corefers with "the company" means "Dell" is more likely to refer to an organization than the last name of a person).

A simple method for joint prediction is pipelining, where structures are predicted one-at-a-time with earlier predictions informing later ones. Generally. lower-level linguistic structures are predicted first. For example, it is common practice to first predict part-of-speech tags before dependency parsing a sentence for both neural and statistical models (Yamada and Matsumoto, 2003; Chen and Manning, 2014). A more sophisticated approach is to relate the structure in a probabilistic model and perform inference using belief propagation (Smith and Eisner, 2008; Durrett and Klein, 2014). Lastly, rather than predicting multiple separate models, a multi-task model can also learn single joint structure capturing multiple tasks. For example Finkel and Manning (2009) jointly perform parsing and named entity recognition by learning a grammar augmented with named entity types.

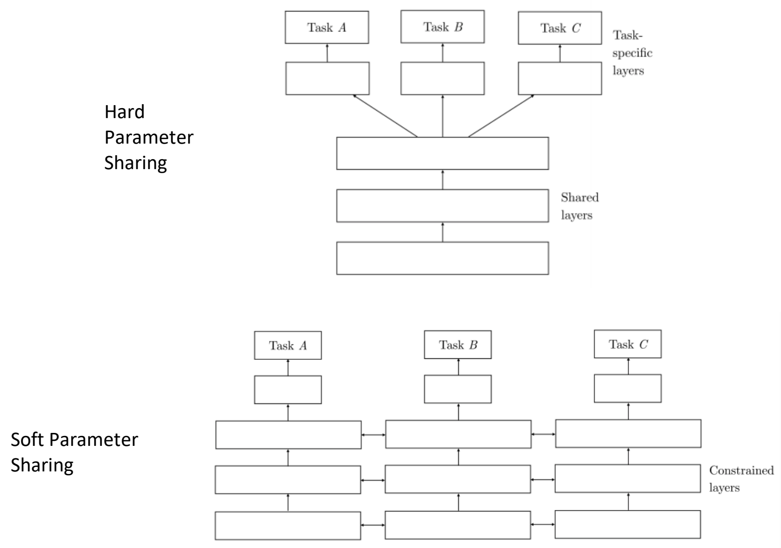


Figure 2.3: Hard vs. soft parameter sharing for neural multi-task learning. In hard parameter sharing, different tasks use the same network weights. In soft parameter sharing, weights are task-specific but are constrained or regularized to be similar, allowing for knowledge to transfer across tasks. Figure from Ruder (2017)

Neural Methods. Deep learning has fundamentally changed how multi-task learning is done because it allows for shared representations between tasks. There are two primary ways of sharing parameters. In hard parameter sharing (Caruana, 1993), the same neural parameters are used to build up features for different tasks, with additional task-specific parameters added to go from the features to the final prediction. The particular choice of which parameters are shared is a widely researched design decision. The earliest approaches simply shared word embeddings between tasks Collobert and Weston (2008). Søgaard and Goldberg (2016) and Hashimoto et al. (2017) train many-task models where the tasks are arranged hierarchically according to their linguistic level. The models are multi-layer neural networks, with prediction modules attaches to each layer. Lower layers do lower-level tasks such as part-of-speech tagging, while deeper layers in the network to more complex tasks, a strategy reminiscent of earlier pipelined NLP systems. Other works (Misra et al., 2016; Ruder et al., 2019; Liu et al., 2017) have proposed complicated architectures designed to ensure only beneficial information is shared across tasks. These generally have a mixture of shared and task-specific parameters, with regularizers encouraging cross-task features to be shared. In contrast with hard parameter sharing, in soft parameter sharing each task has its own model with its own parameters. The distance between the model parameters is then regularized in order to encourage the parameters to be similar. For example Duong et al. (2015) use an L2 penalty to improve cross-lingual dependency parsing. I use hard parameter instead of soft parameter sharing in this thesis due to the goal of efficiency. Soft parameter sharing does not scale as well because it requires more task-specific parameters for each task, which makes the model larger as the number of tasks increases.

Underlying these neural multi-task methods is the idea that shared features can be more useful than independently learned ones. First, shared features have to be effective for multiple tasks, causing the model to "focus" on more generalizable features and ignore features less useful for other tasks. Essentially sharing parameters constrains the model's hypothesis space by encouraging the features to be useful for more tasks. Feature sharing can also improve performance because some features may be easier to learn from one task than another. Hence, combining tasks provides the model access to easier learning of more features.

Generally, prior work in multi-task learning has focused on a small number of closely related tasks. For example, Luong et al. (2016) add a single auxiliary task to improve a primary one during seq2seq learning, Miwa and Bansal (2016) jointly learn entity detection and relation extraction, Rei (2017) learns language modeling with sequence tagging, Peng et al. (2017) learn three semantic parsing tasks, Zhang and Weiss (2016) jointly learn POS tagging and dependency parsing, and Søgaard and Goldberg (2016) jointly learn syntactic tagging tasks. When tasks are unrelated, performance drops considerably (Bingel and Søgaard, 2017; Plank and Alonso, 2017). In contrast, following the goal of generality, the multi-task work in this thesis generally focuses on learning many diverse tasks.

2.4 Semi-Supervised Learning

Semi-supervised learning uses unlabeled data such as text from the web to improve models on tasks of interest. The models learn on a mix of labeled and unlabeled data. Labeled examples teach the model about the task. They then generalize what they learn to unlabeled examples, which provide the model with more coverage of the data manifold – the full range of examples they may see at test time. Semi-supervised learning is particularly advantageous from the perspective of scalability. Labeled data is often scarce, but there is abundant unlabeled text on the web and in books. While training a model using multi-task learning gives it access to a bit more data, there can be orders of magnitudes more unlabeled data. This scalability is particularly important in the context of neural methods, which especially benefit from large amounts of data.

Self-Training. Many semi-supervised learning algorithms rely on variants of self-training (Scudder, 1965) (see Figure 2.4b). Historically, self-training has successfully been applied to NLP tasks such as word-sense disambiguation (Yarowsky, 1995) and parsing (McClosky et al., 2006). In each round of training, the classifier, acting as a "teacher," labels some of the unlabeled data and adds it to the training set. Then, acting as a "student," it is retrained on the new training set. This procedure can be repeated iteratively until the model converges, gradually adding more examples to the training set and expanding the range of examples the model sees. While these "pseudo-labeled" examples will be noisy, the hope is that the model will still be able to learn from them and "smooth out" the noise by processing a large enough volume of examples. Self-training uses unlabeled data to progressively unlock new features for a task. For example, a named entity recognition may learn that "Seattle" is a location by seeing it in the indicative context "I traveled to Seattle" and therefore being provided the right pseudo-label. Then, it can use that new knowledge to learn a new indicative feature (the word "population" occurring near locations) from "Seattle has a large population." Continuing to use the labeled training data is essential for avoiding the problem of semantic drift. A model learning from incorrect pseudo-labels will get worse at the task, which can lead to further incorrect pseudo-labels. Instead, self-training hopes for a beneficial feedback loop where pseudo-labels improve the model such that future pseudo-labels will be more accurate.

While alternating between generating a whole new training set and learning on it is necessary for batch training methods, this process can be inefficient with neural nets using minibatch gradient descent. A solution is to perform "online" self-training, where during each training step a minibatch of unlabeled examples is pseudo-labeled and learned from.

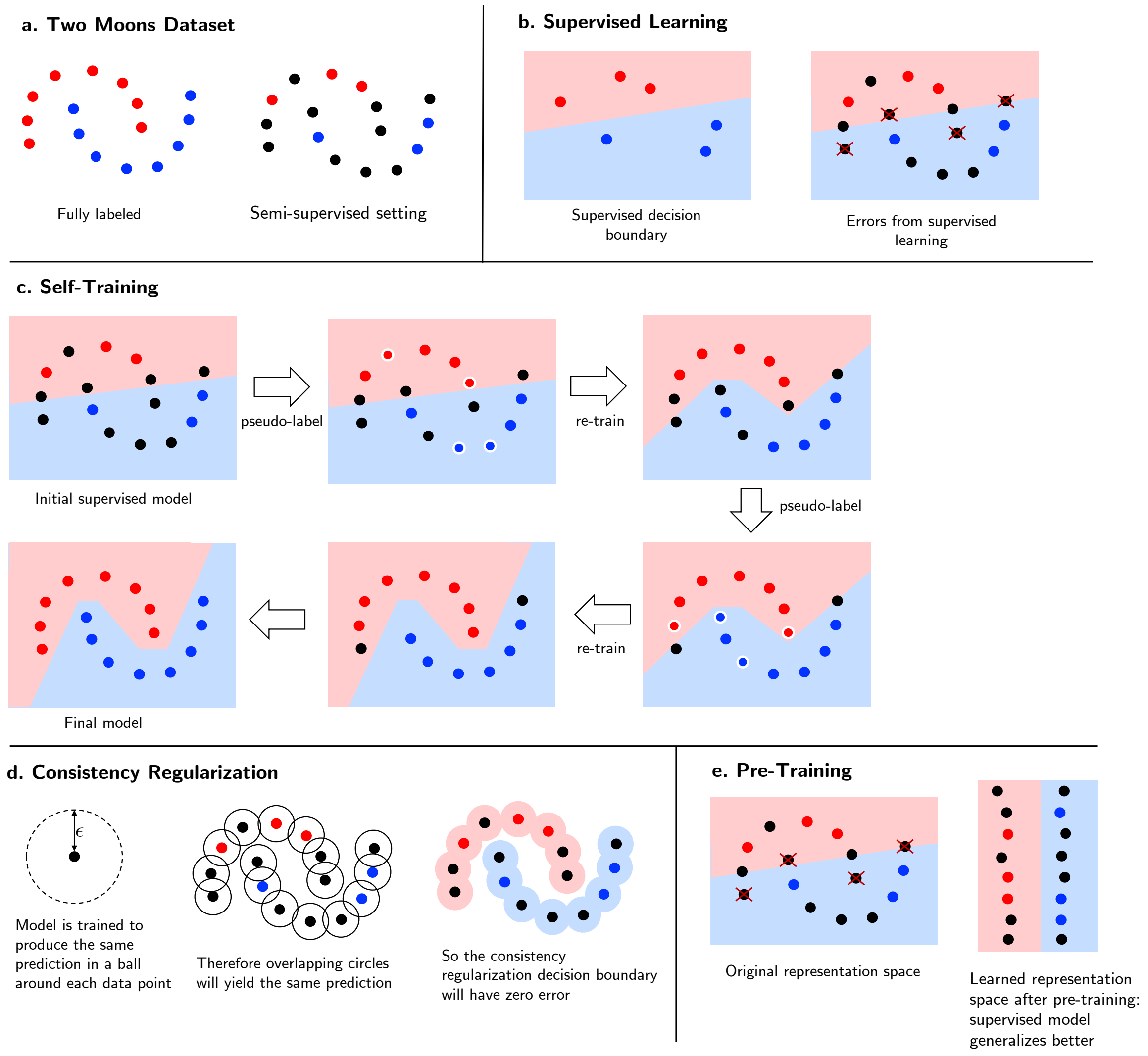

a. Two Moons Dataset
Fully labeled
Semi-supervised setting
b. Supervised Learning
Supervised decision boundary
Errors from supervised learning
c. Self-Training
Initial supervised model
pseudo-label
re-train
pseudo-label
re-train
Final model
d. Consistency Regularization
ϵ
Model is trained to produce the same prediction in a ball around each data point
Therefore overlapping circles will yield the same prediction
So the consistency regularization decision boundary will have zero error
e. Pre-Training
Original representation space
Learned representation space after pre-training: supervised model generalizes better

Figure 2.4: An illustration of how few popular semi-supervised learning methods work on the toy "two moons" dataset. Part **a** gives an overview of the dataset and **b** shows how standard supervised learning fails to learn the correct function from only the labeled examples. Parts **c**, **d**, and **e** demonstrate how self-training, consistency regularization, and pre-training can allow the model to learn a correct decision boundary for the dataset.

In particular, given a minibatch of labeled examples (x_i, y_i) and minibatch of labeled examples x_i^u, the loss function is

$$\mathcal{L}(x_i, y_i, x_i^u, \theta) = \underbrace{CE(y_i, f_\theta(x_i))}_{\substack{\text{standard} \\ \text{supervised loss}}} + \underbrace{CE(\overbrace{\text{one-hot}(\text{argmax}(f_\theta(x_i^u)))}^{\text{pseudo-label}}, f_\theta(x_i^u))}_{\text{self-training loss}}$$

where CE denotes cross-entropy loss. Many approaches (including the consistency regularization methods discussed below) train the student with soft targets from the teacher's output distribution rather than a hard one-hot label (i.e., deleting the argmax), making the procedure more akin to knowledge distillation. As discussed in Section 2.2, soft targets can provide a richer training signal to the model. Another strategy for improving self-training is using a stronger teacher than the model itself. The goal of this idea is to produce better pseudo-labels for the student. For example, tri-training (Zhou and Li, 2005; Ruder and Plank, 2018) uses an ensemble of three models as the teacher and Mean Teacher (Tarvainen and Valpola, 2017a) uses an exponential moving average of the student's weights as the teacher.

Consistency Regularization. Standard self-training appears a bit circular: the model is learning from the exact predictions it already makes, raising concerns that the model will not actually learn much new from the unlabeled data. Indeed, if we replace the one-hot pseudo-label in the self-training loss with the output distribution $f_\theta(x_i^u)$, the two terms in the cross entropy loss become the same![1] Methods using *consistency regularization* (see Figure 2.4d) address this issue. Consistency regularization adds noise (e.g., drawn from a Gaussian distribution) or applies stochastic transformations (e.g., horizontally flipping an image) to the student's inputs. Under consistency regularization, the self-training loss become $CE(f_\theta(\text{noised}(x_i^u)), f_\theta(x_i^u))$, This approach makes the learning task harder for the student network because it is learning from a corrupted input. Due to this more challenging task, the student is more likely to learn from the teacher (often itself, but without the input noise). Another view is that consistency regularization encourages the model to give consistent predictions to nearby data points. This induces "distributional smoothness" or

[1]Although in fact, this approach still works as a semi-supervised learning method called entropy minimization (Grandvalet and Bengio, 2005).

"Lipschitz continuity" in the model, i.e., a small change to the input does not change the output, regularizing the model. In fact, methods like L2 regularization can be viewed as crude ways of also inducing this sort of model smoothness.

Consistency regularization has been very successful for computer vision applications (Bachman et al., 2014; Laine and Aila, 2017; Tarvainen and Valpola, 2017b). It works especially well when the noise is selected adversarially (Miyato et al., 2016) or when combined with other smoothness regularizers such as MixUp (Zhang et al., 2018; Berthelot et al., 2019). However, stochastic input alterations are more difficult to apply to discrete data like text, making consistency regularization less used for natural language processing. Continuous noise can't be added to discrete words, and there are not simple meaning-preserving transformations like cropping or flipping for text. One solution is to add noise to the model's word embeddings (Miyato et al., 2017a). Another solution is ensuring consistent predictions across different *views*, or subsets of the input features. Co-training (Blum and Mitchell, 1998) and co-regularization (Sindhwani and Belkin, 2005), which train two models with disjoint views of the input, take this approach. On unlabeled data, each one acts as a "teacher" for the other model. The models can learn from each other because of their different views. I apply this idea to semi-supervised NLP in Cross-View Training.

2.5 Self-Supervised Learning

Semi-supervised learning allows a model to benefit from large amounts of unlabeled data. However, the model is still limited to learning a pre-specified set of tasks. To build a model for a new task, the expensive procedure of learning over the large unlabeled corpus has to be repeated again. Self-supervised representation learning uses unlabeled data in a more general-purpose way. Rather than learning representations useful for a particular task, self-supervised learning seeks to learn general representations that are transferable to a variety of tasks. Once the slow self-supervised learning phase has been performed once, the model can be reused for many tasks. Self-supervised methods construct what looks like a supervised task from the unlabeled data. For example, deleting some of the input and predicting what it was originally creates a supervised learning problem from the unlabeled data. These tasks are designed to be challenging and general, so they produce

Method	**Encoder**	**Prediction Task**	**Pre-Train Compute**	**Params**	**GLUE Score**
GloVe	Word vector	Neighboring words	7.2e15	5M-50M	68.6
ELMo	Bi-LSTM	LM	3.4e18	96M	71.2
GPT	Transformer (Dec)	LM	4.0e19	117M	78.8
BERT	Transformer	Cloze	1.9e20	335M	84.0
RoBERTa	Transformer	Cloze	3.2e21	356M	88.9
XLNet	Transformer	Any-order LM	3.8e21	360M	89.2
ALBERT	Transformer	Cloze	3.1e22	235M	89.9
T5-11b	Transformer (Enc-Dec)	Seq2Seq Cloze	4.6e22	11B	90.7
GPT-3	Transformer (Dec)	LM	5.1e23	175B	–

Table 2.1: Statistics for selected pre-training methods in NLP. Compute is measured in the total number of pre-training floating point operations. Enc-Dec means encoder-decoder model. Methods are ordered according to the publication date. While rapid progress on GLUE is apparent, the progress has been driven by larger models and increasing amount of pre-training compute.

good representations. There has been rapid growth in the size and ability of these models. See Figure 2.5 for a taxonomy of pre-trained models and Table 2.1 for statistics on some of the more significant methods.

Self-supervised learning methods typically operate in two phases. First, there is a *pre-training* phase where the model learns from the unlabeled corpus. Then during a *fine-tuning* phase, the learned model is transferred to the task of interest. Specifically, the pre-trained weights are used as initialization for a task-specific classifier, with usually a small additional randomly initialized neural network component added on top to make the final classification decisions. The goal of the pre-training is to learn neural network weights that provide a much better initialization for supervised models than random ones. The pre-trained model weights can be updated during the task-specific training, or held fixed for faster but typically less accurate training. The expensive self-supervised pre-training phase only has to occur once, and pre-trained models are relatively quick to fine-tune (often converging in 3 epochs or less) due to their initialization from effective weights. This advantage makes self-supervised pre-training much easier to use in practice than other methods. Pre-training does not have to be self-supervised – for example, CoVe (McCann et al., 2017) pre-trains LSTMs to perform machine translation. However, self-supervised

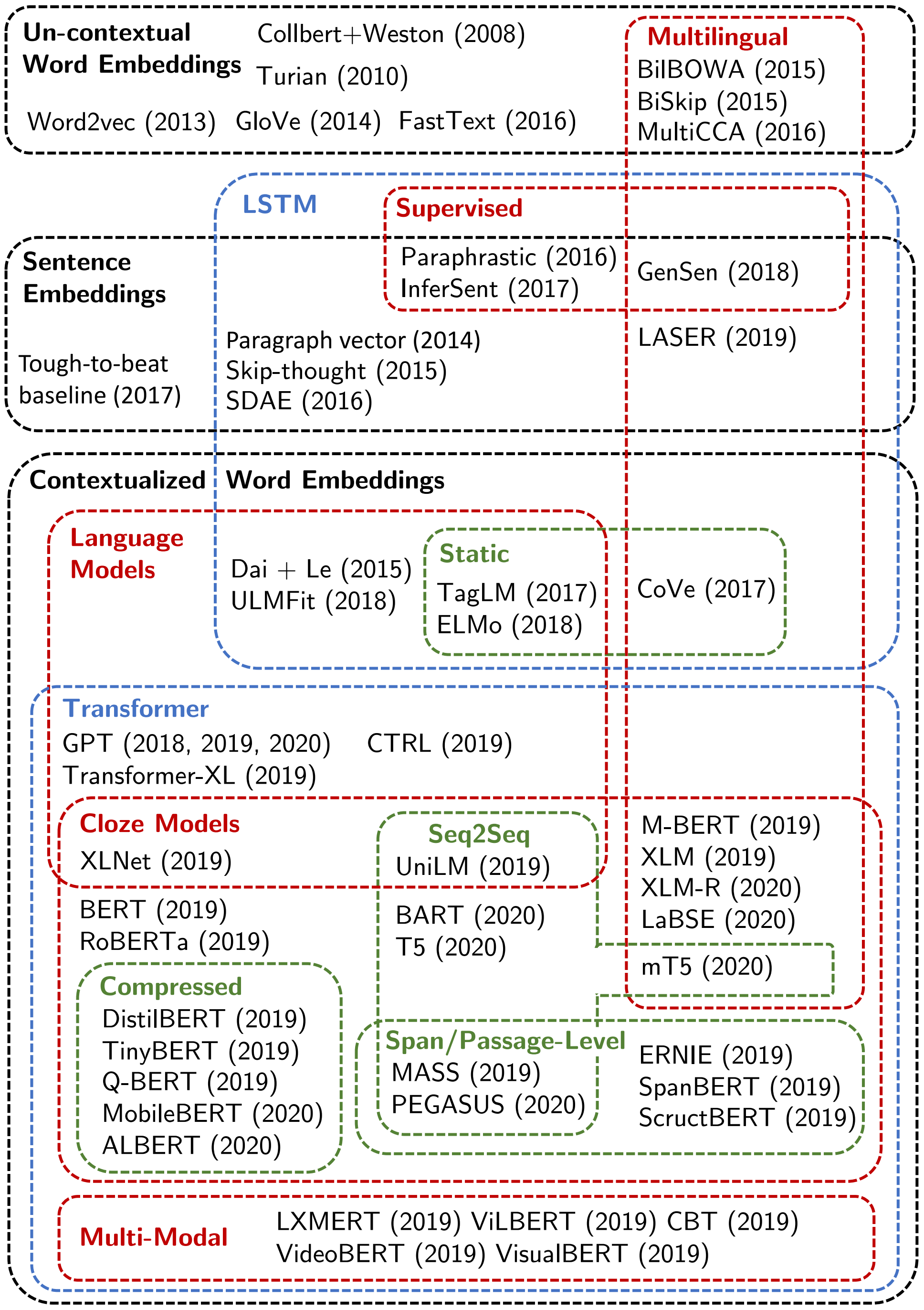


Figure 2.5: A (non-exhaustive) taxonomy of neural pre-training methods for NLP. Black regions correspond to the type of pre-trained model, blue to neural architectures, red to types of pre-training tasks, and green to other details of the models.

methods are now by far more common than supervised pre-training in NLP because of the abundance of unlabeled data.

2.5.1 Learning Word Embeddings

Early applications of self-supervised pre-training learned word embeddings: low-dimensional vector representations of words. Such embeddings became widely used in the NLP community due to their remarkable ability to capture lexical features (see Figure 2.6) despite receiving no human supervision signal. A key advantage of dense word embeddings over sparse lexical features (e.g., unigrams) is in improved generalization to rare or unseen words. A statistical sentiment analysis classifier with bag-of-words features won't learn that "abominable" has negative sentiment if the word isn't present in the training set. However, a model built from word embeddings can generalize from a related word – perhaps "terrible" and "horrible" are in the training set and have a similar embedding to "abominable."

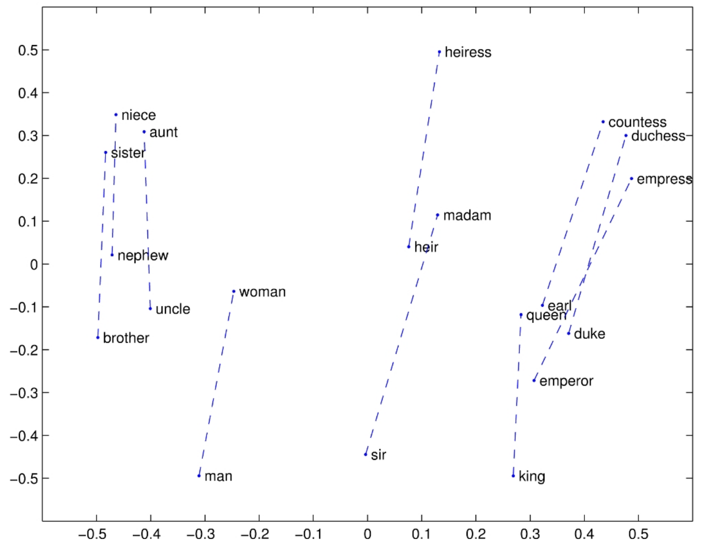


Figure 2.6: An example of lexical features learned by GloVe: a direction in the embedding space corresponds to male/female. Figure from (Pennington et al., 2014).

A pre-neural approach to dealing with this sort of data sparsity in text data is Brown clustering (Brown et al., 1992). Brown clustering groups words into clusters where words occurring in similar contexts are placed in the same cluster. These word clusters are useful as features for statistical systems. Neural methods for learning embeddings are also inspired by the distributional hypothesis that words that occur in similar contexts tend to have similar meanings. For example, the self-supervised tasks for pre-training word embeddings often rely on predicting words based on their contexts. Alternatively, they can be viewed as performing low-rank approximations of the matrix of word co-occurrences. Collobert and Weston (2008) developed the first neural word embedding method, where a neural network is trained to distinguish words in their context from negative samples chosen at random. Perhaps the best-known word embedding method is word2vec, which uses a training objective called skip-gram (Mikolov et al., 2013b). Skip-gram is simpler and faster than the approach from Collobert and Weston (2008). It learns a context vector c and word vector v for each word (these two vectors are typically summed after training, resulting in a single vector per word). Instead of using a neural network, the training objective simply encourages word vectors to have a high dot product with the context vectors of nearby words. After pre-training, the context and word vectors are summed up to get a single embedding for each word. If C is the matrix of all context vectors, the Skip-Gram objective is

$$\mathcal{L}(\theta) = \sum_{w \in \mathcal{D}} \sum_{w' \in \text{window}(w)} - \log \text{softmax}(Cv_w)_{w'}$$

Where w and w' are the indices of words in the vocabulary and window(w) refers to the words within a fixed distance (typically 5) of w. In practice, the full softmax over all context vectors is very expensive. Instead, word2vec makes use of negative sampling, a technique similar to noise-contrastive estimation. Rather than lowering the dot product with all context vectors not in the window, negative sampling selects a small set of random words to lower in score. Subsequent word embeddings such as GloVe (Pennington et al., 2014) and FastText (Bojanowski et al., 2017) use improved training objectives but with the same underlying idea. There has also been extensive research in learning cross-lingual word embeddings that are aligned across languages such that the embedding of a word is close to the embedding of its translation (Faruqui and Dyer, 2014; Luong et al., 2015a; Gouws et al., 2015).

2.5.2 Language Modeling

While methods for pre-training embeddings have been highly successful, these approaches produce a fixed representation of each word. However, in practice, language is highly contextual, with the meaning of a word depending on its neighbors. A clear illustration of this is in polysemous words with many senses: *bat* can refer to an animal, a piece of sporting equipment, or an action depending on its context. For this reason, it can be more effective to pre-train neural networks such as Transformers and LSTMs that build contextual representations of words rather than static representations. An advantage of pre-training text encoders is that the number of randomly initialized parameters in the model can be much smaller, reducing over-fitting. Furthermore, self-supervised tasks have access to much more data than supervised ones, allowing for larger text encoders than can be effectively trained on supervised datasets. Pre-training methods for text encoders typically use the contextualized representations of part of the input to predict another part of the input. Language models were among the first such pre-training task.

Language models (LMs) model the probability $p(\boldsymbol{x})$ of a piece of text $\boldsymbol{x} = [x_1, x_2, ..., x_n]$ consisting of n words. Self-supervised transfer learning methods train language models that each use a neural network with parameters θ to estimate the probability. Applying the chain rule of probability, the probability of a text can be decomposed as

$$p(\boldsymbol{x}) = p(x_1)p(x_2|x_1)p(x_3|x_1x_2)... = \prod_{t=1}^{n} p(x_t|x_{<t})$$

This decomposition is useful because it breaks up $p(\boldsymbol{x})$ into terms that are easier to estimate. While estimating $p(\boldsymbol{x})$ directly is intractable because it requires normalizing over the space of all possible sentences $\boldsymbol{x}$, the probability $p(x_t|x_{<t})$ just requires normalizing over all words in the vocabulary. A neural LM outputs an estimate $p_\theta(x_t|x_{<t})$. In other words, it is trained to predict the next token given the previous context to that token's left. First, a neural network first encodes the input sequence into a sequence of contextualized representations $\boldsymbol{h} = [h_1, ..., h_n]$ where each representation is a d-dimensional vector corresponding to an input token. The network runs left-to-right, which means each h_t only depends on previous tokens $x_{<t}$. This can be achieved by using a recurrent network such as an LSTM

(Hochreiter and Schmidhuber, 1997) or applying causal masking to a Transformer.

The LM then uses a softmax layer to estimate token probabilities:

$$p_\theta(x_t|x_{<t}) = \exp(h_t^T e_{x_t}) / \sum_{x' \in \text{vocab}} \exp(h_t^T e_{x'})$$

where e_{x_t} denotes a token embedding. The estimated probability of a sentence $\boldsymbol{x}$ is then

$$p_\theta(\boldsymbol{x}) = \prod_{t=1}^{n} p_\theta(x_t|x_{<t})$$

Given a dataset $\mathcal{D}$ of text, the models are trained to minimize the negative log-likelihood loss

$$\mathcal{L}(\theta) = \sum_{\boldsymbol{x} \in \mathcal{D}} -\log p_\theta(\boldsymbol{x}) = \sum_{\boldsymbol{x} \in \mathcal{D}} -\log \prod_{t=1}^{n} p_\theta(x_t|x_{<t}) = \sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{t=1}^{n} -\log p_\theta(x_t|x_{<t})$$

using mini-batch gradient descent.

Language models can be used to generate text one token at a time. At each step, the model outputs a token x_t given the previous context $x_{<t}$ by sampling from $p_\theta(x_t|x_{<t})$.[2] The generated token is then added to the context and the process repeats. These models are called *autoregressive* due to this process of making predictions based on their previous outputs. However, instead of generation, I focus on the use of the parameters θ or representations $\boldsymbol{x}$ to be incorporated into downstream supervised tasks. For this use, language models have a big drawback in that they are uni-directional: the representation for a particular token only depends on previous tokens. As a result, the representations are less contextual than they could be: for example, the first token of the input is not contextual at all and is therefore no more useful than a static word embedding. A solution is to train two language models, one that operates left-to-right and one that operates right-to-left. Combining both models by concatenating the left-to-right and right-to-left states results in better contextualized representations for each token.

[2]It is possible to improve generation quality by using more sophisticated methods than sampling from the distribution. Examples include using a temperature, using beam search, or using top-k sampling (Fan et al., 2018) / nucleus sampling (Holtzman et al., 2020)

Language model pre-training was first proposed by (Dai and Le, 2015), although their experiments were relatively small-scale. A significant breakthrough was ELMo (Peters et al., 2018), which (1) used both directions of language modeling, and (2) trained much larger models on much larger datasets. ELMo representations were typically held fixed (i.e., treated as static word embeddings) and passed into existing strong models for the task. Radford et al. (2018) proposed GPT (Generative Pre-training), which scaled up models further, using a large Transformer network instead of LSTMs. Furthermore, GPT (along with the prior ULMFit (Howard and Ruder, 2018) method) fine-tuned the whole LM rather than only using it to produce static feature vectors. GPT was a breakthrough because it demonstrated how these models could be easily applied to many tasks without task-specific add-ons.

As a generative task, language modeling is especially useful for pre-training text generation models such as text summarization or chit-chat dialogue systems. However, large language models such as GPT-2 (Radford et al., 2019) and GPT-3 (Brown et al., 2020) have shown that language modeling alone can train general models that learn, at least shallowly, many tasks. This property is demonstrated through few-shot or zero-shot transfer of the models to NLP datasets.

2.5.3 Cloze Modeling

While combining the representations from left-to-right and right-to-left LMs results in a bidirectional model, such a model is only bidirectional in a "shallow" way, with the left and right context of each token just being combined through concatenating states. In contrast, a Transformer encoder can be much more expressive because every token "sees" every other token during training. Having this kind of cross attention across the input is very beneficial for tasks such as reading comprehension, where it is beneficial to build up representations of the question based on the context and vice versa. Consequently, many variants of "co-attention" have been developed (Xiong et al., 2017; Seo et al., 2017) to improve reading comprehension models.

An effective pre-training task for bidirectional Transformers (i.e., Transformer encoders without causal masking) is the cloze task. The goal of the cloze task is to predict a word's

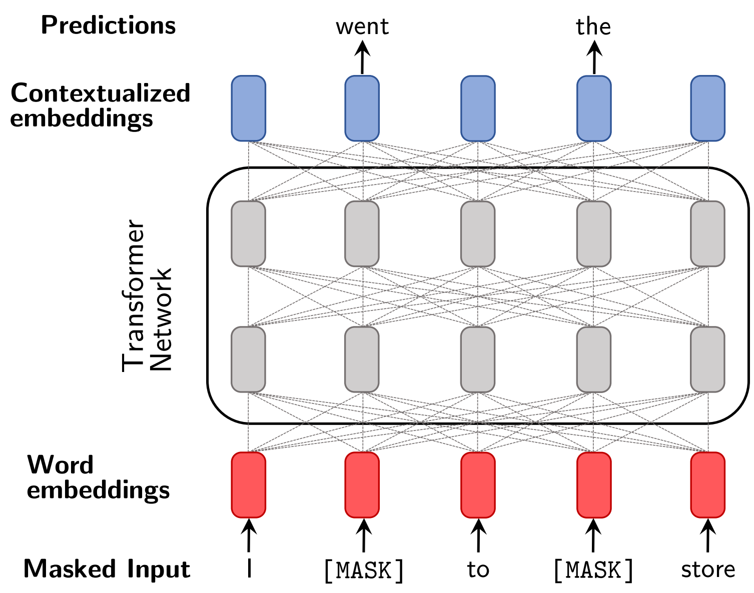


Figure 2.7: An overview of masked language modeling. First, words in the input are randomly masked out. Then a Transformer network embeds the sequence as contextualized vectors. The final representations are used to predict the identities of the masked-out input words.

identity given both its left and right context (as opposed to only one side in an LM). It is commonly used as a teaching method for people learning a second language. It also proves very effective for teaching neural models about language.

More formally, cloze models learn the probability $p_{\text{data}}(x_t|\boldsymbol{x}_{\backslash t})$ of a token x_t occurring in the surrounding context $\boldsymbol{x}_{\backslash t} = [x_1, ..., x_{t-1}, x_{t+1}, ..., x_n]$. Masked language model (MLM) training (see Figure 2.7) is the standard approach for learning in these models. They select a small subset of the unlabeled input sequence (typically 15%), mask the identities of those tokens by replacing them with a special [MASK] token, and then train the network to recover the original input. Specifically, MLMs represents the context as $[x_1$, ... , x_{t-1}, `[MASK]`, x_{t+1}, ... , $x_n]$, the input sequence with x_t replaced by a special `[MASK]` placeholder token. This masked sequence is encoded into vector representations by a Transformer network. Then the representation at position t is passed into a softmax layer to produce a distribution over tokens $p_\theta(x_t|\boldsymbol{x}_{\setminus t})$ for the position. The negative log-likelihood loss for the model over a dataset of unlabeled text $\mathcal{D}$ is

$$\mathcal{L}_{\text{MLM}}(\theta) = \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}}\frac{1}{|\boldsymbol{x}|}\sum_{t=1}^{\boldsymbol{x}} - \log p_\theta(x_t|\boldsymbol{x}_{\setminus t})$$

To improve efficiency, masked language models like BERT typically mask out k tokens at once from each example rather than only one. In particular, on each sentence $\boldsymbol{x}$, the MLM selects $\mathcal{R}$, a randomly chosen set of k unique positions $1 \leq r \leq |\boldsymbol{x}|$ to replace with `[MASK]`. The loss then becomes

$$\mathcal{L}_{\text{MLE}}(\theta) = \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}}\frac{1}{|\mathcal{S}|}\sum_{\mathcal{R}\in\mathcal{S}}\frac{1}{|\mathcal{R}|}\sum_{t\in\mathcal{R}} - \log p_\theta(x_t|\text{REPLACE}(\boldsymbol{x}, \mathcal{R}, \texttt{[MASK]}))$$

where $\text{REPLACE}(\boldsymbol{x}, \mathcal{R}, \texttt{[MASK]})$ denotes replacing the tokens at positions in $\mathcal{R}$ with `[MASK]` and $\mathcal{S} = \{\mathcal{R} : \mathcal{R} \subset [n] \wedge |\mathcal{R}| = k\}$ contains all possible selections of k unique positions in the input sequence to mask out. In practice, the second sum is approximated by a taking a single random selection for $\mathcal{R}$. These two losses are only equivalent if $-\log p_\theta(x_t|\boldsymbol{x}_{\setminus t}) = -\log p_\theta(x_t|\text{REPLACE}(\boldsymbol{x}, \mathcal{R}, \texttt{[MASK]}))$, i.e., the additional masked-out tokens do not change the model's outputs. In practice, k is set to a small value (for BERT, $k = \lceil 0.15n \rceil$, i.e., 15% of the tokens are masked out per example) to reduce this discrepancy. If many more tokens were masked out, the input would become unusable because too much context would be missing.

Cloze modeling has proven very effective as a self-supervised learning method, yielding rapid gains on NLP benchmarks. A series of masked language models, including BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019e), XLNet (Yang et al., 2019), and ALBERT (Lan et al., 2020), have pushed this training method to larger and larger scales. However, this scaling has primarily been done by increasing model size and training times. In this thesis, I propose ELECTRA, a more efficient pre-training method inspired by the cloze task. It alleviates a substantial efficiency drawback of masking, where the model only learns from 15% of input tokens per example. Given the remarkable scalability of recent pre-training methods, I argue that improving efficiency should be the primary concern of self-supervised pre-training methods because improving efficiency consistently improves downstream task performance.

2.5.4 Phrase-Level and Sentence-Level Objectives

The self-supervised learning methods discussed so far learn vector representations for individual word tokens. But what about larger units of text such as sentences? In practice, having effective word representations seems sufficient to achieve good performance on many tasks: during fine-tuning, the model can learn to combine them. In fact, state-of-the-art models such as RoBERTa (Liu et al., 2019e), XLNet (Yang et al., 2019) and T5 (Raffel et al., 2020) only use token-level tasks. Nevertheless, this section will briefly detail some of the most notable representation learning objectives for larger text units.

A simple method for producing a sentence embedding is a "bag of vectors" approach the averages or sums up word embeddings (learned using the techniques discussed in Section 2.5.1) for the constituent words. Arora et al. (2017) show that this approach is surprisingly effective and can be improved by using techniques based on PCA to assign weights to the word vectors when taking the sum. Sentence-level and paragraph-level self-supervised tasks were also explored prior to the widespread use of language model and cloze model pre-training. Paragraph vector (Le and Mikolov, 2014) learns paragraph representations that are predictive of the words in the paragraph, skip-thought (Kiros et al., 2015) learns sentence vectors that can be passed into decoders such that the decoders generate the previous or next sentence, and Sequential Denoising Autoencoders (SDAE; Hill et al. (2016)) train a sentence encoder that embeds a sentence corrupted through word deletions and swaps into a vector from which a decoder can recover the original (uncorrupted) sentence. Supervised approaches to learning sentence embeddings such as from natural language inference, (Conneau et al., 2017), paraphrase (Wieting et al., 2016), or translation (Artetxe and Schwenk, 2019) data have also been explored.

Since BERT, there have been several approaches combining cloze modeling with higher-level self-supervised tasks. BERT itself included a "next-sentence prediction" task. For this task, the input consists of two text segments concatenated together. The second segment is either the text following the first or a random segment from another document. BERT is then trained to predict from its first output vector[3], which is considered the embedding for the whole text, if the second segment is the following text or a random one. In practice, this task has actually been shown to decrease the performance of pre-trained transformers (Liu et al., 2019e; Yang et al., 2019), likely because (1) this task is very easy for most random segments and (2) the random segments provide irrelevant context that interferes with the cloze task. However, similar but more challenging tasks have been shown to improve performance slightly. ALBERT (Lan et al., 2020) uses a sentence ordering task where the two segments are either in order or switched, StructBERT (Wang et al., 2020) uses a combination of sentence order and next sentence prediction, and PEGASUS (Zhang et al., 2020) masks out important sentences in the input and trains a decoder to reconstruct them, which particularly improves text summarization performance.

There also have been self-supervised tasks proposed to learn representations for spans or chunks of text. For example, MASS (Song et al., 2019) masks out contiguous chunks of text to provide a more challenging reconstruction task to the model, and StructBERT (Wang et al., 2020) shuffles tri-grams in the input sequence and trains the model to recover the original ordering. SpanBERT (Joshi et al., 2019) masks out spans and then trains the model to predict the masked-out tokens from the hidden state bordering each masked span. It shows improvements on span selection tasks such as question answering and coreference resolution.

2.5.5 Extensions and Analysis of Pre-Trained Transformers

The effectiveness of self-supervised pre-training combined with the expressive Transformer architecture (see Section 2.1) has led to a significant paradigm shift in NLP. Their generality has greatly reduced the need for task-specific engineering: rather than designing a model for a specific task, it is now often possible to download a pre-trained model that will work well on it out-of-the-box. Pre-trained models also make NLP tasks centered around linguistic structure such as dependency parsing less relevant because these models learn much about language structure as a by-product of their self-supervised training. This finding has been observed through "BERTology," an emerging area in NLP focused

[3]This vector corresponds to a special `[CLS]` token that is prepended to the text before BERT processes it.

on analyzing pre-trained models.

One line of research examines the *outputs* of language models or cloze models on carefully chosen input sentences (Linzen et al., 2016; Khandelwal et al., 2018; Gulordava et al., 2018). For example, a model's performance at subject-verb agreement (generating the correct number of a verb far away from its subject) provides a measure of the model's syntactic ability. Another line of work investigates the internal *vector representations* of the model (Adi et al., 2017; Zhang and Bowman, 2018), often using probing classifiers. Probing classifiers are simple neural networks that take the vector representations of a pre-trained model as input and are trained to do a supervised task (e.g., part-of-speech tagging). If a probing classifier achieves high accuracy, it suggests that the input representations reflect the corresponding aspect of language (e.g., low-level syntax). These studies have demonstrated that pre-trained models such as BERT model aspects of syntax (Shi et al., 2016; Hewitt and Manning, 2019; Goldberg, 2019) or coreference (Tenney et al., 2018; Liu et al., 2019a) without explicitly being trained for the tasks. Some of my own research has looked at BERT's attention mechanisms, and found attention heads capturing aspects of syntax and coreference remarkably well (Clark et al. (2019a), see Figure 2.8). Together, these results show the remarkable effectiveness of self-supervised pre-training for learning rich representations of language.

The success of pre-training on language data has led to cross-modal self-supervised learning methods that add other modalities. For example, LXMERT (Tan and Bansal, 2019), ViLVERT (Lu et al., 2019), and VisualBERT (Li et al., 2019) learn joint representations of images and text using data such as captioned images while VideoBERT (Sun et al., 2019b) learns to represent videos and text from captioned videos. Another direction has been pre-training multi-lingual models that can be then applied to machine translation or zero-shot language transfer, where the model is fine-tuned on labeled data in one language, but then is evaluated at performing the task for other languages. These methods often make use of translation data as well as monolingual to improve performance, but still commonly employ variants of the cloze task (Lample and Conneau, 2019; Feng et al., 2020a; Xue et al., 2020).

With the goal of extending pre-trained models to text generation tasks, another line of research has developed methods for sequence-to-sequence pre-training. A simple method

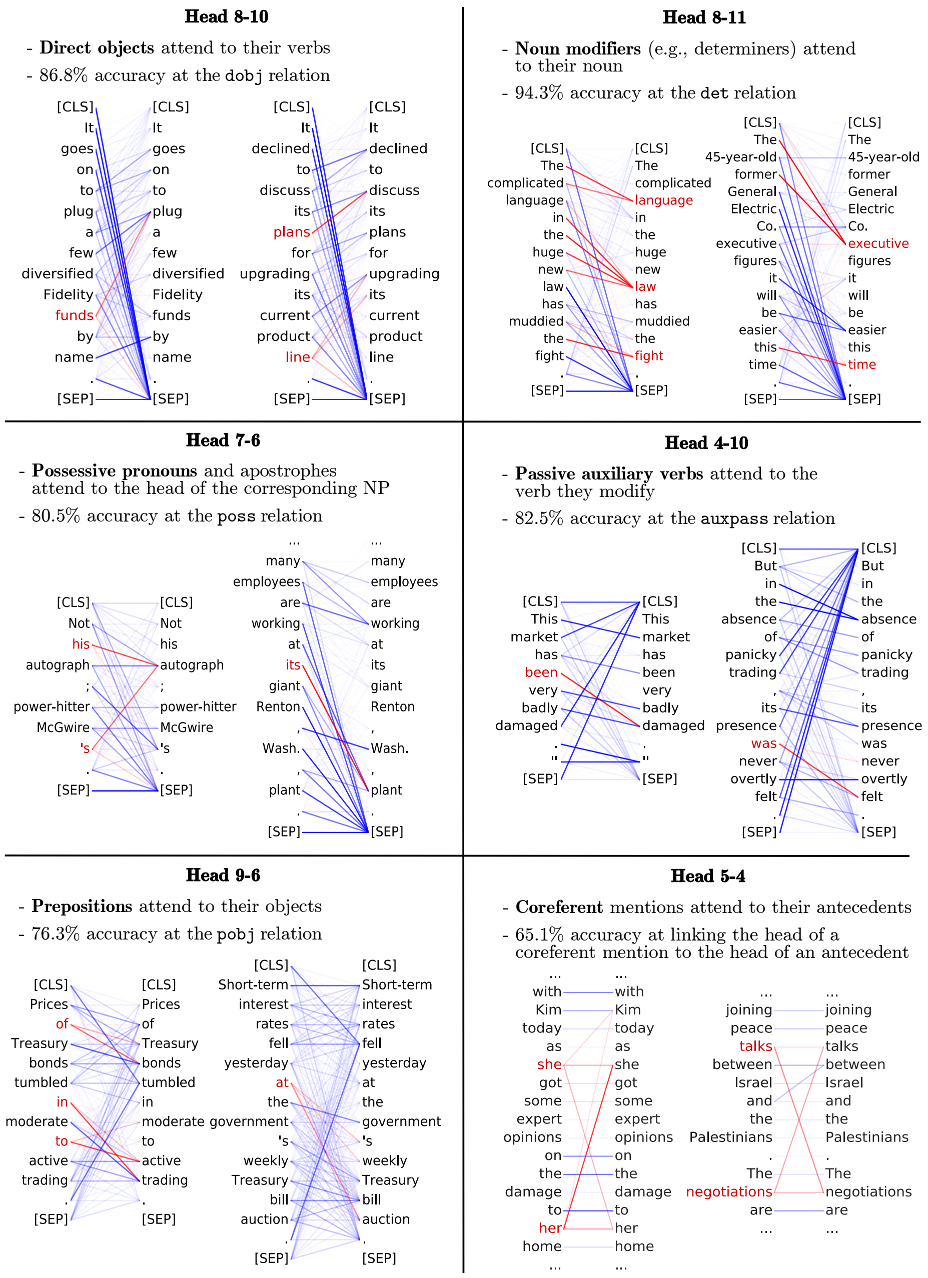

Head 8-10
- Direct objects attend to their verbs
- 86.8% accuracy at the dobj relation
Head 8-11
- Noun modifiers (e.g., determiners) attend to their noun
- 94.3% accuracy at the det relation
Head 7-6
- Possessive pronouns and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the poss relation
Head 4-10
- Passive auxiliary verbs attend to the verb they modify
- 82.5% accuracy at the auxpass relation
Head 9-6
- Prepositions attend to their objects
- 76.3% accuracy at the pobj relation
Head 5-4
- Coreferent mentions attend to their antecedents
- 65.1% accuracy at linking the head of a coreferent mention to the head of an antecedent

Figure 2.8: BERT attention heads that correspond to linguistic phenomena. The darkness of a line indicates the strength of the attention weight. All attention to/from red words is colored red to highlight attention head behaviors. Despite not being explicitly trained on these tasks, BERT's attention heads perform remarkably well, illustrating how syntax-sensitive behavior can emerge from self-supervised training alone. See Clark et al. (2019a) for more details on this analysis.

is UniLM (Dong et al., 2019), which adds an additional objective to a cloze model Transformer where for part of the sequence it (1) predicts the next token instead of the current one and (2) it uses causal masking so these predictions are left-to-right. Other approaches change the model architecture to being full encoder-decoder Transformers. These methods typically pass a masked input into the encoder like cloze models. Then the decoder is trained to predict either the original (unmasked) input sequence as in BART (Lewis et al., 2020) or only the masked-out tokens as in MASS (Song et al., 2019) and T5 (Raffel et al., 2020). A related task is gap sentences generation (Zhang et al., 2020), where input sentences are masked out and a decoder is trained to reproduce them. Pre-trained sequence-to-sequence models generally do not work better on downstream tasks than pre-trained Transformer encoders, but have the significant advantage of being applicable to text generation problems.

Inference is expensive with large pre-trained transformers, making these models difficult to use in practice. Consequently, there has been substantial research on compressing these models through distillation (Sun et al., 2020; Sanh et al., 2019; Jiao et al., 2020) or quantization (Shen et al., 2020; Zafrir et al., 2019). This thesis focuses more on training efficiency, an aspect of pre-training that also remains very costly. However, the models I present could be combined with these compression techniques to improve inference speed.

2.6 Natural Language Processing Tasks and Datasets

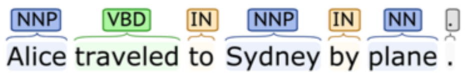
With the goal of *generality* in mind, I evaluate the transfer learning methods presented in this thesis on a diverse range of natural language processing tasks. This section details the tasks and their corresponding datasets. Statistics on the datasets are shown in Table 2.2.

2.6.1 Syntactic Tasks

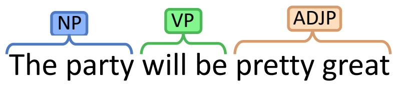
Syntactic tasks involve predicting the grammatical structure of the sentence. Syntactic tasks are generally less useful intrinsically compared to semantic tasks involving sentence meaning. Historically, syntactic analysis has been used as a step in a pipeline performing a deeper analysis of a sentence (e.g., for sentiment analysis, understanding the scope of negations is crucial). With the rise of end-to-end deep learning, this use of syntactic tasks is less common. While modeling the syntactic structure of a sentence is an essential step of natural language understanding, end-to-end systems have shown an ability to automatically learn the aspects of syntax needed for the task without human-provided labels (Blevins et al., 2018). However, syntactic analysis can still be useful for other kinds of analysis. For example, tracking the adjectives that often modify a noun over a corpus of text can be used to automatically collect properties of the object (i.e., information extraction) or what people think of the object (opinion mining). Similarly, if there is no training data for a task, syntactic structure is useful for defining a rule-based system, such as collecting subject-verb-object triples for relation extraction.

In this dissertation, I use syntactic tasks of various granularities, going from classifying words, to groups of words, to relationships between the words in a sentence. Most of the datasets are derived from the Penn Treebank (Marcus et al., 1993), which is a large corpus of Wall Street Journal articles annotated with syntax trees.

Part-of-Speech (POS) Tagging: Labeling words with their syntactic categories (e.g., determiner, adjective, etc.). POS tagging is an example of a tagging tagging task where each word in a sentence is assigned a label. I use the Wall Street Journal (WSJ) portion of the Penn Treebank (Marcus et al., 1993) and report word-level accuracy.

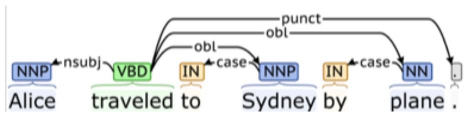


Text Chunking: Dividing a sentence into syntactically correlated parts (e.g., a noun phrase followed by a verb phrase). While the labeling is over groups of words, text chunking is often converted into a tagging task using a method such as BIO encoding (see named entity recognition below for details). I use the CoNLLL-2000 shared task data (Tjong Kim Sang and Buchholz, 2000) and report F1 over predicted chunks.



Combinatory Categorial Grammar (CCG) Supertagging: Labeling words with CCG supertags, lexical categories that encode information about the sentence's predicate-argument structure. CCG is widely used in syntactic and semantic parsing. I use data from CCGBank (Hockenmaier and Steedman, 2007) and report word-level accuracy.

Dependency Parsing: Inferring a tree-structure describing the syntax of a sentence. In a dependency parse, words in a sentence are treated as nodes in a graph. Typed directed edges connect the words, forming a tree structure describing the syntactic structure of the sentence. In particular, each word in the sentence receives exactly one in-going edge going from another word (the “head) to it (the “dependent) of a particular type (the “relation”). There is an additional special `root` node whose edge points to the head of the sentence. I use the Penn Treebank converted to Stanford Dependencies (version 3.3.0) and report labeled attachment score (LAS). LAS measures the percent of edges that have both the correct head and relation type.



2.6.2 General Language Understanding Benchmark Tasks

The next set of tasks capture a broader set of datasets covering natural language understanding (NLU). They are derived from the General Language Understanding (GLUE, Wang et al. (2019)) benchmark, which is a collection of NLU tasks. The tasks are designed to cover a diverse range of NLU, with datasets of different domains, sizes, and difficulties covering a range of the many facets of natural language understanding.

CoLA: Corpus of Linguistic Acceptability (Warstadt et al., 2018). The task is to determine whether a given sentence is grammatical or not. The dataset contains 8.5k train examples from books and journal articles on linguistic theory.

There is a problem. `Acceptable` There are a problem. `Unacceptable`

SST: Stanford Sentiment Treebank (Socher et al., 2013). The tasks is to determine if the sentence is positive or negative in sentiment. The dataset contains 67k train examples from movie reviews.

The movie drags at first, but the climactic ending is worth it.	`Positive Sentiment`	An amazingly dull film.	`Negative Sentiment`

MRPC: Microsoft Research Paraphrase Corpus (Dolan and Brockett, 2005). The task is to predict whether two sentences are semantically equivalent or not. The dataset contains 3.7k train examples from online news sources.

All of the animals escaped the zoo.
`Is a paraphrase of`
The zoo's animals broke free.

All of the animals escaped the zoo.
`Is not a paraphrase of`
The zoo's animals went on a journey.

STS: Semantic Textual Similarity (Cer et al., 2017). The tasks is to predict how semantically similar two sentences are on a 1–5 scale. The dataset contains 5.8k train examples drawn from new headlines, video and image captions, and natural language inference data.

All of the animals escaped the zoo.
`Relatedness score: 4`
The lion escaped its cage.

All of the animals escaped the zoo.
`Relatedness score: 3`
The animals went on a journey.

QQP: Quora Question Pairs (Iyer et al., 2017). The task is to determine whether a pair of questions are semantically equivalent. The dataset contains 364k train examples from the community question-answering website Quora.

What is America's largest city?
`Is a paraphrase of`
Which city in the U.S. has the biggest population?

What is America's largest city?
`Is not a paraphrase of`
What is America's capital?

MNLI: Multi-genre Natural Language Inference (Williams et al., 2018). Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis, contradicts the hypothesis, or neither. The dataset contains 393k train examples drawn from ten different sources.

All of the animals escaped the zoo.
Implies
The lion escaped its cage

All of the animals escaped the zoo.
Contradicts
The lion was left trapped in its cage.

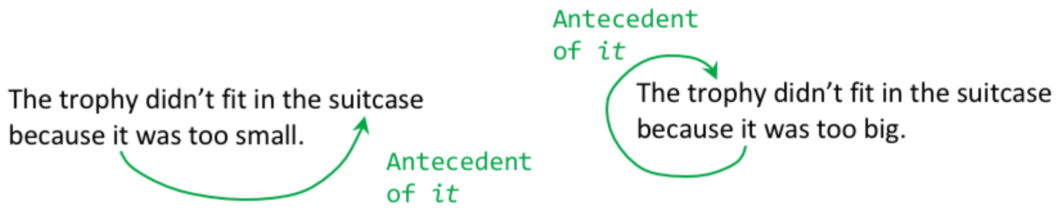
QNLI: Question Natural Language Inference; constructed from SQuAD (Rajpurkar et al., 2016). The task is to predict whether a context sentence contains the answer to a question sentence. The dataset contains 108k train examples from Wikipedia.

... France defeated Brazil in the finals.
Answers
Who won the 1998 World Cup?

... France defeated Brazil in the finals.
Does not answer
Who won the battle of Agincourt?

RTE: Recognizing Textual Entailment (Giampiccolo et al., 2007). Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis or not. The dataset contains 2.5k train examples from a series of annual textual entailment challenges.

WNLI: Winograd schema (Levesque, 2011). The goal is to correctly pick a pronoun's antecedent out of two possibilities. Winograd schema are deliberately constructed so that a small change to the sentence changes the Winograd schema (e.g., changing "small" to "large" below). This property makes the task challenging because surface-level features alone will not suffice to make a correct prediction.

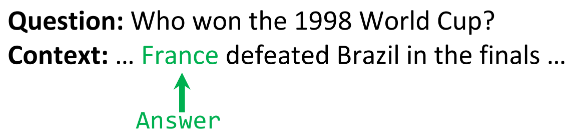

The trophy didn't fit in the suitcase because it was too small.
Antecedent of it
Antecedent of it
The trophy didn't fit in the suitcase because it was too big.

Metrics: I report Spearman correlation for STS, Matthews correlation coefficient (MCC) for CoLA and accuracy for the other tasks.

2.6.3 Question Answering

Question answering is an NLP application with potentially great value. Most people interact with the vast amount of information available on the web through information retrieval, but this has limitations. It usually requires visiting and reading the retrieved page, which is slow and poorly suited for small phone screens and spoken language interfaces (e.g., asking a question to a smartphone). Furthermore, it can be cumbersome to use when answering multi-hop reasoning questions that require multiple web searches to answer. It is also poor with highly specific information, such as answering a novel question about a particular document. General question answering systems offer an appealing alternative. In this thesis, I focus on *reading comprehension*, a kind of question answering where the answer is known to be contained in a particular document. Reading comprehension systems can be combined with retrieval to build open domain question systems or can be useful for quickly providing information on a specific document. Reading comprehension can also be considered a core test of natural language understanding: one sign of truly understanding a piece of text is being able to answer arbitrary questions about its content and meaning.

SQuAD 1.1: Stanford Question Answering Dataset (Rajpurkar et al., 2016). Given a context paragraph (e.g., part of a Wikipedia article) and a question (e.g., about the article's subject), the task is to select the span of text in the paragraph answering the question. The span is a contiguous sequence of words, so the task can be viewed as using a highlighter to denote the answer to a question, making the task easier to learn for models and easier to evaluate than generation-based question answering, where the answer can be paraphrased or synthesized from the context. The dataset contains 88k train examples from Wikipedia. I report F1 and Exact Match. EM requires that the predicted span exactly matches the human-provided one while F1 offers partial credit where a predicted span partially overlapping the gold-standard span still gets some points.

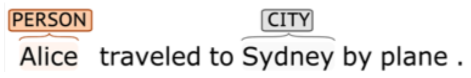

Question: Who won the 1998 World Cup?
Context: … France defeated Brazil in the finals …
Answer

SQuAD 2.0: Stanford Question Answering Dataset version 2.0 (Rajpurkar et al., 2018). This task adds additional questions to SQuAD whose answer does not exist in the context; models have to recognize when these questions occur and not return an answer for them. This task was designed so simple heuristics such as "answer a *who* question with the first person in the article" aren't always reliable, making the task more challenging. The no-answer questions were created adversarially by specifically creating misleading questions that models are likely to make a prediction on. The dataset contains 130k train examples from Wikipedia (approximately 50k unanswerable questions added on top of SQuAD 1.1). I report F1 and Exact Match.

2.6.4 Other Tasks

I lastly use named entity recognition and machine translation tasks in my experiments. While less directly related to natural language understanding, these are two of the most widely used NLP tasks in the world.

Named Entity Recognition (NER): Identifying named entities in a sentence and classifying each as a location, organization, person, or miscellaneous entity. I use the CoNLL-2003 data (Tjong Kim Sang and De Meulder, 2003) dataset. I also use the OntoNotes (Hovy et al., 2006) dataset for more fine-grained analysis of entities (FGN). This dataset contains 18 instead of 4 types, such as "Facility" and "Geo Political Entity."



NER can be converted to a tagging task using an encoding such BIO. In BIO, each word is labeled as a **B**eginning, **I**nterior, or **O**utside word of an entity. From these tags, the actual entities can be recovered. In practice, it sometimes helps to use BIOES encoding, which also marks the **E**ndings of entities or **S**ingleton entities that consist of a single token. I report entity-level F1 score, which computes the precision and recall of completely getting all the words in an entity as well as predicting the correct type.

Machine Translation: Machine translation is the task of automatically converting text from one language to another while preserving the meaning. I use the English-Vietnamese translation dataset from IWSLT 2015 (Cettolo et al., 2015), which contains translations of TED talks. I report (tokenized) BLEU scores on the tst2013 test set. BLEU measures the overlap between the predicted and reference translation. The score combines overlap measures (a variant of precision) for various numbers of n-grams ranging from unigrams (are the same words in the reference and prediction?) to 4-grams (is every 4-word chunk in the reference in the prediction?), with 4-grams providing a stricter measure, but one that captures word order better.

Corpus	|Train|	|Test|	Task	Metric	Domain
			Syntactic Tasks		
POS	38k	5.5k	Part-of-speech tagging	Acc.	News
Chunk	9k	2k	Text chunking	F1.	News
CCG	40k	2.4k	CCG-supertagging	Acc.	News
Dep	40k	2.4k	Dependency Parsing	LAS	News
			Named Entity Recognition		
NER	15k	3.7k	Named entity recognition	F1	News
FGN	29k	4k	Fine-grained NER	F1	Misc
			Single-Sentence Tasks		
CoLA	8.5k	1k	Acceptability	MCC	Misc.
SST	67k	1.8k	Sentiment	Acc.	Movie reviews
			Similarity and Paraphrase		
MRPC	3.7k	1.7k	Paraphrase Detection	Acc.	News
STS	7k	1.4k	Sentence Similarity	Spearman	Misc.
QQP	364k	391k	Question Paraphrase Detection	Acc.	Social QA
			Natural Language Inference Tasks		
MNLI	400k	20k	Natural Language Inference	Acc.	Misc.
QNLI	5.4k	1.8k	Question-Answer Entailment	Acc.	Wikipedia
RTE	3.7k	1.7k	Natural Language Inference	Acc.	News, Wikipedia
WNLI	634	146	Pronominal Anaphora	Acc.	Fiction Books
			Reading Comprehension		
SQuAD 1.1	88k	20k	Answer Selection	F1/EM	Wikipedia
SQuAD 2.0	130k	20k	Answer Selection / Answerability	F1/EM	Wikipedia
			Machine Translation		
En-Vi	400k	20k	English to Vietnamese Translation	BLEU	TED Talks

Table 2.2: Summary and basic statistics of the datasets used in this dissertation. Dataset sizes are given in terms of the number of sentences.

Chapter 3

Robust Multi-Task Learning with Born-Again Multi-Task Networks

With the goal of building general-purpose NLP systems that can perform many tasks, the most direct approach is simply to train multi-task models. When tasks are related, it is intuitive that multi-task learning can improve performance: knowledge the model learns about one task can be transferred to improve its ability at a related task. Indeed, much work in multi-task learning for NLP has focused on learning a small number of closely related tasks such as POS tagging and dependency parsing (Zhang and Weiss, 2016) or entity detection and relation extraction (Miwa and Bansal, 2016). But with the broader goal of highly general systems, I am interested in building models that can perform a large number of tasks that are potentially quite different. A many-task NLP model also has efficiency advantages: one multi-task model can be faster and easier to develop and deploy than having many single-task ones.

Ideally, multi-task learning across many tasks would improve over single-task models when tasks are closely related but never perform worse than a single-task model. After all, the additional tasks are only adding more training data to the model during learning. However, in practice, multi-task models often perform worse than their single-task counterparts when going beyond carefully selected related tasks. For example, Bingel and Søgaard (2017) train two-task models across all combinations of 10 tasks, but only find improvements in 38 of 90 task pairs and Plank and Alonso (2017) explore using a variety

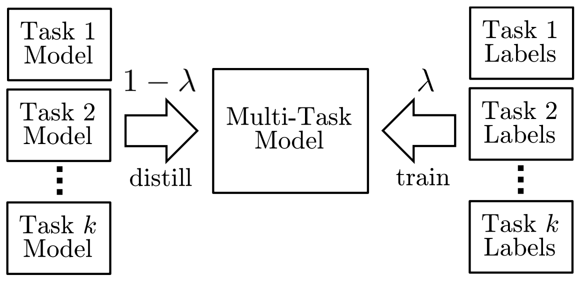


Figure 3.1: An overview of Born-Again Multi-Task Networks (BAM). The multi-task model is trained on a mixture of soft targets from single-task teacher networks (distillation) and the task labels (standard training). λ is increased linearly from 0 to 1 over the course of training.

of auxiliary tasks to improve sequence prediction, but only obtained significant improvements for 1 of 5 tasks. McCann et al. (2018) develop a benchmark of 10 natural language understanding tasks with the goal of facilitating multi-task research, but their multi-task baseline underperforms single-task ones.

In this chapter, I propose a method for *robust* multi-task learning where the multi-task model never under-performs its single-task counterparts while still achieving gains from related tasks. The key idea is to use knowledge distillation (Ba and Caruana, 2014; Hinton et al., 2015) so that a single-task model "teaches" the multi-task model. Intuitively, the multi-task model will never perform worse than the single-task one when it can learn directly how the single-task one performs the task.

Knowledge distillation transfers knowledge from a "teacher" model to a "student" model by training the student to imitate the teacher's outputs. In "born-again networks" (Furlanello et al., 2018), the teacher and student have the same neural architecture and model size, but surprisingly the student is able to surpass the teacher's accuracy. Intuitively, distillation is effective because the teacher's output distribution over classes provides more training signal than a one-hot label; Hinton et al. (2015) suggest that teacher outputs contain "dark knowledge" capturing additional information about training examples.

This work extends born-again networks to the multi-task setting. I compare Single$\rightarrow$Multi[1] born-again distillation with several other variants (Single$\rightarrow$Single and Multi$\rightarrow$Multi),

[1]I use Single$\rightarrow$Multi to indicate distilling single-task "teacher" models into a multi-task "student" model.

and also explore performing multiple rounds of distillation (Single$\rightarrow$Multi$\rightarrow$Single$\rightarrow$Multi). Furthermore, I propose a simple teacher annealing method that helps the student model outperform its teachers. Teacher annealing gradually transitions the student from learning from the teacher to learning from the gold labels. This method ensures the student gets a rich training signal early in training, but is not limited to only imitating the teacher.

My experiments multi-task fine-tune BERT (Devlin et al., 2019) to perform the tasks from the GLUE natural language understanding benchmark (Wang et al., 2019). My training method, which I call Born-Again Multi-tasking (BAM)[2], consistently outperforms standard single-task and multi-task training. Further analysis shows the multi-task models benefit from both better regularization and transfer between related tasks.

3.1 Previous Work

This section discusses several closely related previous works; for a thorough overview of prior work in multi-task learning, refer to background section 2.3. Distilling large models into small models (Kim and Rush, 2016; Mou et al., 2016) or ensembles of models into single models (Kuncoro et al., 2016; Liu et al., 2019b) has been shown to improve results for many NLP tasks. There has also been some work on using knowledge distillation to aid in multi-task learning. In reinforcement learning, knowledge distillation has been used to regularize multi-task agents (Parisotto et al., 2016; Teh et al., 2017). In NLP, Tan et al. (2019) distill single-language-pair machine translation systems into a many-language system, but they focus on multilingual rather than multi-task learning, use a more complex training procedure, and only experiment with Single$\rightarrow$Multi distillation. Several other works also explore fine-tuning BERT using multiple tasks (Phang et al., 2018; Liu et al., 2019c; Keskar et al., 2019b). However, they use only standard transfer or multi-task learning, instead focusing on finding beneficial task pairs or designing improved task-specific components on top of BERT.

[2]Code is available at `https://github.com/google-research/google-research/tree/master/bam`

3.2 Method

3.2.1 Multi-Task Setup

Model. All of my models are built on top of BERT (Devlin et al., 2019). This model passes byte-pair-tokenized (Sennrich et al., 2016b) input sentences through a Transformer network (Vaswani et al., 2017), producing a contextualized representation for each token. The vector corresponding to the first input token (for BERT this is a special token `[CLS]` that is prepended to each input sequence.) c is passed into a task-specific classifier. For classification tasks, I use a standard softmax layer: $\text{softmax}(Wc)$. For regression tasks, I normalize the labels so they are between 0 and 1 and then use a size-1 NN layer with a sigmoid activation: $\text{sigmoid}(w^T c)$. In my multi-task models, all of the model parameters are shared across tasks except for these classifiers on top of BERT, which means less than 0.01% of the parameters are task-specific. Following BERT, the token embeddings and Transformer are initialized with weights from a self-supervised pre-training phase.[3] I chose to build on top of BERT instead of train a model from scratch because (1) BERT is widely used since it produces stronger models and (2) it provides a sterner test for the benefits of multi-task learning because BERT already has a large amount of knowledge from its pre-training phase.

Training. Single-task training is performed as in Devlin et al. (2019). For multi-task training, examples of different tasks are shuffled together, even within minibatches. The summed loss across all tasks is minimized. For efficiency on TPU hardware, I run all task-specific classifiers over all examples and mask out all of the losses except the particular task the example is from. The extra computation from producing predictions for the other tasks is negligible because running a single softmax layer is very cheap compared with the rest of BERT.

[3]For BERT code and weights, see `https://github.com/google-research/bert`.

3.2.2 Knowledge Distillation

To provide a clearer understanding of multi-task distillation, I first briefly review standard single-task knowledge distillation (for details on knowledge distillation see background section 2.2). I use $\mathcal{D}_\tau = \{(x_\tau^1, y_\tau^1), ..., (x_\tau^N, y_\tau^N)\}$ to denote the training set for a task τ and $f_\tau(x, \theta)$ to denote the output for task τ produced by a neural network with parameters θ on the input x. Standard supervised learning trains θ to minimize the loss on the training set:

$$\mathcal{L}(\theta) = \sum_{x_\tau^i, y_\tau^i \in \mathcal{D}_\tau} \ell(y_\tau^i, f_\tau(x_\tau^i, \theta))$$

where for classification tasks ℓ is usually cross-entropy. Knowledge distillation trains the model to instead match the predictions of a teacher model with parameters θ':

$$\mathcal{L}(\theta) = \sum_{x_\tau^i, y_\tau^i \in \mathcal{D}_\tau} \ell(f_\tau(x_\tau^i, \theta'), f_\tau(x_\tau^i, \theta))$$

Note that my distilled networks are "born-again" in that the student has the same model architecture as the teacher, i.e., all of my models have the same prediction function f_τ for each task. For regression tasks, I train the student to minimize the L2 distance between its prediction and the teacher's instead of using cross-entropy loss.

Multi-Task Distillation. Given a set of tasks $\mathcal{T}$, I train a single-task model with parameters θ_τ on each task τ. For most experiments, I use the single-task models to teach a multi-task model with parameters θ:

$$\mathcal{L}(\theta) = \sum_{\tau \in \mathcal{T}} \sum_{x_\tau^i, y_\tau^i \in \mathcal{D}_\tau} \ell(f_\tau(x_\tau^i, \theta_\tau), f_\tau(x_\tau^i, \theta))$$

This method is denoted as Single$\rightarrow$Multi distillation in the rest of this chapter. However, I experiment with other distillation strategies as well.

Teacher Annealing. In knowledge distillation, the student is trained to imitate the teacher. This raises the concern that the student may be limited by the teacher's performance and not be able to substantially outperform the teacher. To address this, I propose *teacher annealing*, which mixes the teacher prediction with the gold label during training. Specifically,

Model	**Avg.**	**CoLA**	**SST-2**	**MRPC**	**STS-B**	**QQP**	**MNLI**	**QNLI**	**RTE**
Single	84.0	60.6	93.2	88.0	90.0	91.3	86.6	92.3	70.4
Multi	85.5	60.3	93.3	88.0	89.8	91.4	86.5	92.2	82.1
Single→Single	84.3	**61.7****	93.2	**88.7***	90.0	91.4	**86.8****	**92.5*****	70.0
Multi→Multi	85.6	60.9	93.5	88.1	89.8	**91.5***	86.7	92.3	82.0
Single→Multi	**86.0*****	**61.8****	**93.6***	**89.3****	89.7	**91.6***	**87.0*****	**92.5*****	**82.8***

Table 3.1: Comparison of distillation strategies on the GLUE dev set. *, **, and *** indicate statistically significant ($p < .05$, $p < .01$, and $p < .001$) improvements over both Single and Multi according to bootstrap hypothesis tests. For all statistical tests I use the Holm-Bonferroni method (Holm, 1979) to correct for multiple comparisons.

the term in the summation becomes

$$\ell(\lambda y^i_\tau + (1-\lambda) f_\tau(x^i_\tau, \theta_\tau), f_\tau(x^i_\tau, \theta))$$

where λ is linearly increased from 0 to 1 throughout training. Early in training, the model is mostly distilling to get as useful of a training signal as possible. Towards the end of training, the model is mostly relying on the gold-standard labels so it can learn to surpass its teachers.

3.3 Experiments

Data. I use the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019), which consists of 9 natural language understanding tasks on English data (see background section 2.6 for details on GLUE).

Training Details. Rather than simply shuffling the datasets for my multi-task models, I follow the task sampling procedure from Bowman et al. (2018), where the probability of training on an example for a particular task τ is proportional to $|\mathcal{D}_\tau|^{0.75}$. This ensures that tasks with very large datasets don't overly dominate the training.

I also use the layerwise-learning-rate trick from Howard and Ruder (2018). If layer 0 is the NN layer closest to the output, the learning rate for a particular layer d is set to

BASE_LR $\cdot \alpha^d$ (i.e., layers closest to the input get lower learning rates). The intuition is that pre-trained layers closer to the input learn more general features, so they shouldn't be altered much during training.

Hyperparameters. For single-task models, I use the same hyperparameters as in the original BERT experiments except I pick a layerwise-learning-rate decay α of 1.0 or 0.9 on the dev set for each task. For multi-task models, I train the model for longer (6 epochs instead of 3) and with a larger batch size (128 instead of 32), using $\alpha = 0.9$ and a learning rate of 1e-4. All models use the BERT-Large pre-trained weights.

Reporting Results. Dev set results report the average score (Spearman correlation for STS, Matthews correlation for CoLA, and accuracy for the other tasks) on all GLUE tasks except WNLI, for which methods can't outperform a majority baseline. Results show the median score of at least 20 trials with different randoms seeds. I found that using a large number of trials is essential because results can vary significantly for different runs. For example, standard deviations in score are over ± 1 for CoLA, RTE, and MRPC for multi-task models. Single-task standard deviations are even larger.

3.4 Results

Main Results. I compare models trained with single-task learning, multi-task learning, and several varieties of distillation in Table 3.1. While standard multi-task training improves over single-task training for RTE (likely because it is closely related to MNLI), there is no improvement on the other tasks. In contrast, Single$\rightarrow$Multi knowledge distillation improves or matches the performance of the other methods on all tasks except STS, the only regression task in GLUE. I believe distillation does not work well for regression tasks because there is no distribution over classes passed on by the teacher to aid learning.

The gain for Single$\rightarrow$Multi over Multi is larger than the gain for Single$\rightarrow$Single over Single, suggesting that distillation works particularly well in combination with multi-task learning. Interestingly, Single$\rightarrow$Multi works substantially better than Multi$\rightarrow$Multi distillation. I speculate it may help that the student is exposed to a diverse set of teachers

Model	**GLUE score**
BERT-Base (Devlin et al., 2019)	78.5
BERT-Large (Devlin et al., 2019)	80.5
BERT on STILTs (Phang et al., 2018)	82.0
MT-DNN (Liu et al., 2019c)	82.2
Span-Extractive BERT on STILTs (Keskar et al., 2019b)	82.3
Snorkel MeTaL ensemble	83.2
MT-DNN$_{KD}$* (Liu et al., 2019b)	83.7
BERT-Large + BAM	82.3

Table 3.2: Comparison of BAM with previous methods on the GLUE test set. *MT-DNN$_{KD}$ is distilled from a diverse ensemble of models.

in the same way ensembles benefit from a diverse set of models, but future work is required to fully understand this phenomenon. In addition to the models reported in the table, I also trained Single$\rightarrow$Multi$\rightarrow$Single$\rightarrow$Multi models. However, the difference with Single$\rightarrow$Multi was not statistically significant, suggesting there is little value in multiple rounds of distillation.

Overall, a key benefit of my method is robustness: while standard multi-task learning produces mixed results, Single$\rightarrow$Multi distillation consistently outperforms standard single-task and multi-task training. I also note that in some trials single-task training resulted in models that score quite poorly (e.g., less than 91 for QQP or less than 70 for MRPC), while the multi-task models have more dependable performance.

Test Set Results. I compare against previous work by submitting to the GLUE leaderboard. I use Single$\rightarrow$Multi distillation. Following the procedure used by BERT, I train multiple models and submit the one with the highest average dev set score to the test set. BERT trained 10 models for each task (80 total); I trained 20 multi-task models. Results are shown in Table 3.2.

BAM outperforms or matches existing published results that do not rely on ensembling. However, due to the variance between trials discussed under "Reporting Results," I think these test set numbers should be taken with a grain of salt, as they only show the performance of individual training runs (which is further complicated by tricks such as dev set model selection). I believe significance testing over multiple trials would be needed to have

Model	**Avg. Score**
Multi	85.5
+Single-Task Fine-Tuning	+0.3
Single$\rightarrow$Multi	86.0
+Single-Task Fine-Tuning	+0.1

Table 3.3: Combining multi-task training with single-task fine-tuning. Improvements are statistically significant ($p < .01$) according to Mann-Whitney U tests.

a definitive comparison.

Single-Task Fine-Tuning. A crucial difference distinguishing my work from the STILTs, Snorkel MeTaL, and MT-DNN$_{KD}$ methods in Table 3.2 is that I do not single-task fine-tune my model. That is, I do not further train the model on individual tasks after the multi-task training finishes. While single-task fine-tuning improves results, I think to some extent it defeats the purpose of multi-task learning: the result of training is one model for each task instead of a model that can perform all of the tasks. Compared to having many single-task models, a multi-task model is simpler to deploy, faster to run, and arguably more scientifically interesting from the perspective of building general language-processing systems.

I evaluate the benefits of single-task fine-tuning in Table 3.3. Single-task fine-tuning initializes models with multi-task-learned weights and then performs single-task training. Hyperparameters are the same as for my single-task models except I use a smaller learning rate of 1e-5. While single-task fine-tuning unsurprisingly improves results, the gain on top of Single$\rightarrow$Multi distillation is small, reinforcing the claim that distillation provides many of the benefits of single-task training while producing a single unified model instead of many task-specific models.

Ablation Study. I show the importance of teacher annealing and the other training tricks in Table 3.4. I found them all to significantly improve scores. Using pure distillation without teacher annealing (i.e., fixing $\lambda = 0$) performs no better than standard multi-task learning, demonstrating the importance of the proposed teacher annealing method.

Comparing combinations of tasks. Training on a large number of tasks is known to help

Model	**Avg. Score**
Single→Multi	86.0
No layer-wise LRs	−0.3
No task sampling	−0.4
No teacher annealing: $\lambda = 0$	−0.5
No teacher annealing: $\lambda = 0.5$	−0.3

Table 3.4: Ablation Study. Differences from Single→Multi are statistically significant ($p < .001$) according to Mann-Whitney U tests.

Trained Tasks	**RTE score**
RTE	70.0
RTE + MNLI	83.4
RTE + QQP + CoLA + SST	75.1
All GLUE	82.8

Table 3.5: Which tasks help RTE? Pairwise differences are statistically significant ($p < .01$) according to Mann-Whitney U tests.

regularize multi-task models (Ruder, 2017). A related benefit of multi-task learning is the transfer of learned "knowledge" between closely related tasks. I investigate these two benefits by comparing several models on the RTE task, including one trained with a very closely related task (MNLI, a much large textual entailment dataset) and one trained with fairly unrelated tasks (QQP, CoLA, and SST). I use Single→Multi distillation (Single→Single in the case of the RTE-only model). Both sets of auxiliary tasks improve RTE performance, suggesting that both benefits are playing a role in improving multi-task models. Interestingly, RTE + MNLI alone slightly outperforms the model performing all tasks, perhaps because training on MNLI, which has a very large dataset, is already enough to sufficiently regularize the model.

3.5 Discussion

I have shown that Single$\rightarrow$Multi distillation combined with teacher annealing produces results consistently better than standard single-task or multi-task training. Achieving robust multi-task gains across many tasks has remained elusive in previous research, so this method makes multi-task learning more broadly useful within NLP. In particular, it opens the door to easily training models on many tasks without needing to think carefully about the tasks and worrying about task interference. However, with the exception of closely related tasks with small datasets (e.g., MNLI helping RTE), the overall size of the gains from my multi-task method are small compared to the gains provided by transfer learning from self-supervised tasks (i.e., BERT). It remains to be fully understood to what extent "self-supervised pre-training is all you need" and where transfer/multi-task learning from supervised tasks can still provide substantial value. Overall, my feeling is that a small number of examples from a related task still might fade to insignificance in their utility compared to billions of tokens of unlabeled text, especially because the unsupervised objectives performed over unlabeled text tend to be very rich (e.g., text generation) compared to supervised tasks (e.g., binary classification of sentences on a specific phenomenon like sentiment). However, this work shows that even on top of BERT there are large gains to be had when datasets are small and tasks are closely related.

Chapter 4

Scalable Semi-Supervised Learning with Cross-View Training

As explained in Chapter 3, multi-task learning enables NLP models to learn from more data: instead of a single human-labeled dataset, they are trained on many. Nevertheless, creating human-labeled datasets is costly, with many consisting of less than one hundred examples. Due to the range of different phenomena in language and vast amounts of world knowledge needed for challenging NLP tasks, it is unlikely there will ever be enough labeled data to capture it all. Luckily, unlabeled text data is abundant on the web. Given how deep learning methods thrive when given many examples, can we use this unlabeled text to provide our models with much more data?

This chapter draws inspiration from traditional semi-supervised learning techniques such as self-training and co-training to develop a powerful algorithm for using unlabeled data to improve models. These methods have historically been effective for NLP (Yarowsky, 1995; McClosky et al., 2006), but have been less commonly employed with neural models. Specifically, I will present Cross-View Training (CVT), a self-training algorithm that works well with deep learning models because it takes advantage of the ability for neural networks to have shared representations. CVT is much faster than concurrent pre-training methods such as ELMo and achieves excellent results on a variety of NLP tasks.

Currently, the dominant way of using unlabeled data for NLP systems is pre-training (see Section 2.5). These methods perform unsupervised representation learning on a large corpus of unlabeled data followed by supervised training. I developed CVT before the widespread adoption of pre-trained text encoders. Although pre-training has since become the primary way of using unlabeled data, I believe there still is value in other semi-supervised learning approaches.

In particular, a disadvantage of pre-training is that the first representation learning phase does not take advantage of labeled data – the model attempts to learn generally effective representations rather than ones that are targeted towards a particular task. This is a limitation in efficiency: the model has to essentially learn much more than it actually needs for the specific task of interest. Semi-supervised learning algorithms like self-training do not suffer from this problem because they continually learn about a task on a mix of labeled and unlabeled data and focus much more narrowly on particular task(s).

In self-training, the model learns as normal on labeled examples. On unlabeled examples, the model acts as both a "teacher" that makes predictions about the examples and a "student" that is trained on those predictions. Although this process can achieve gains by allowing the learning of new predictive words or features from contexts, it is somewhat tautological: the model already produces the predictions it is being trained on. Methods in computer vision addresses this by adding noise to the student's input, training the model so it is robust to input perturbations (Sajjadi et al., 2016; Wei et al., 2018). However, applying noise is difficult for discrete inputs like text.

As a solution, I take inspiration from multi-view learning (Blum and Mitchell, 1998; Xu et al., 2013) and train the model to produce consistent predictions across different *views* of the input. Instead of only training the full model as a student, CVT adds auxiliary prediction modules – neural networks that transform vector representations into predictions – to the model and also trains them as students. The input to each student prediction module is a subset of the model's intermediate representations corresponding to a restricted view of the input example. For example, one auxiliary prediction module for sequence tagging is attached to only the "forward" LSTM in the model's first Bi-LSTM layer, so it makes predictions without seeing any tokens to the right of the current one.

CVT works by improving the model's representation learning. The auxiliary prediction modules can learn from the full model's predictions because the full model has a better, unrestricted view of the input. As the auxiliary modules learn to make accurate predictions despite their restricted views of the input, they improve the quality of the representations they are built on top of. This in turn improves the full model, which uses the same shared representations. In short, my method combines the idea of representation learning on unlabeled data with classic self-training.

CVT can be applied to a variety of tasks and neural architectures, but I focus on sequence modeling tasks where the prediction modules are attached to a shared Bi-LSTM encoder. I propose auxiliary prediction modules that work well for sequence taggers, graph-based dependency parsers, and sequence-to-sequence models. I evaluate my approach on English dependency parsing, combinatory categorial grammar supertagging, named entity recognition, part-of-speech tagging, and text chunking, as well as English-to-Vietnamese machine translation. CVT improves over previous results on all these tasks. Furthermore, CVT can easily and effectively be combined with the multi-task learning we previously explored through adding additional prediction modules for the different tasks on top of the shared Bi-LSTM encoder. Training a unified model to jointly perform all of the tasks except machine translation improves results while decreasing the total training time. The resulting model[1] is comparable to ELMo (Peters et al., 2018) in terms of architecture and unlabeled data source, but scores better while training 10x faster. This efficient learning demonstrates the utility of self-training as an alternative to pre-training in NLP.

4.1 Method

I first present Cross-View Training and describe how it can be combined effectively with multi-task learning. See Figure 4.1 for an overview of the training method.

4.1.1 Cross-View Training

Let $\mathcal{D}_l = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ represent a labeled dataset and $\mathcal{D}_{ul} = \{x_1$, x_2, ..., $x_M\}$ represent an unlabeled dataset. I use $p_\theta(y|x_i)$ to denote the output distribution over classes produced by the model with parameters θ on input x_i. During CVT, the model

[1]Code is available at `https://github.com/tensorflow/models/tree/master/research/cvt_text`

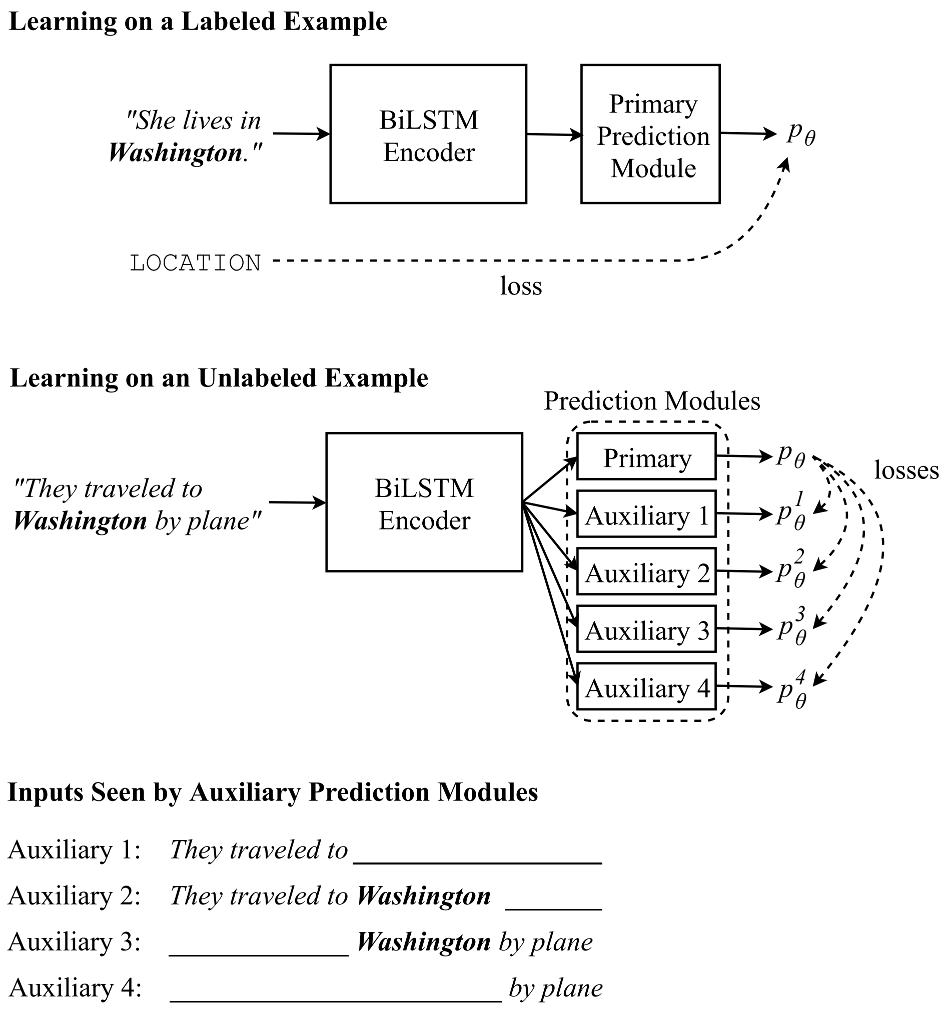

Learning on a Labeled Example
"She lives in **Washington**."
BiLSTM Encoder
Primary Prediction Module
p_θ
LOCATION
loss
Learning on an Unlabeled Example
Prediction Modules
"They traveled to **Washington** by plane"
BiLSTM Encoder
Primary
Auxiliary 1
Auxiliary 2
Auxiliary 3
Auxiliary 4
p_θ
p_θ^1
p_θ^2
p_θ^3
p_θ^4
losses
Inputs Seen by Auxiliary Prediction Modules
Auxiliary 1: They traveled to ____
Auxiliary 2: They traveled to **Washington** ____
Auxiliary 3: ____ **Washington** by plane
Auxiliary 4: ____ by plane

Figure 4.1: An overview of Cross-View Training. The model is trained with standard supervised learning on labeled examples. On unlabeled examples, auxiliary prediction modules with different views of the input are trained to agree with the primary prediction module. This particular example shows CVT applied to named entity recognition. From the labeled example, the model can learn that “Washington” usually refers to a location. Then, on unlabeled data, auxiliary prediction modules are trained to reach the same prediction without seeing some of the input. In doing so, they improve the contextual representations produced by the model, for example, learning that “traveled to” is usually followed by a location.

alternates learning on a minibatch of labeled examples and learning on a minibatch of unlabeled examples. For labeled examples, CVT uses standard cross-entropy loss:

$$\mathcal{L}_{\text{sup}}(\theta) = \frac{1}{|\mathcal{D}_l|} \sum_{x_i, y_i \in \mathcal{D}_l} CE(y_i, p_\theta(y|x_i))$$

CVT adds k auxiliary prediction modules to the model, which are used when learning on unlabeled examples. A prediction module is usually a small neural network (e.g., a hidden layer followed by a softmax layer). Each one takes as input an intermediate representation $h^j(x_i)$ produced by the model (e.g., the outputs of one of the LSTMs in a Bi-LSTM model). It outputs a distribution over labels $p_\theta^j(y|x_i)$. Each h^j is chosen such that it only uses a part of the input x_i; the particular choice can depend on the task and model architecture. I propose variants for several tasks in Section 4.2. The auxiliary prediction modules are only used during training; the test-time prediction come from the primary prediction module that produces p_θ.

On an unlabeled example, the model first produces soft[2] targets $p_\theta(y|x_i)$ by performing inference. CVT trains the auxiliary prediction modules to match the primary prediction module on the unlabeled data by minimizing

$$\mathcal{L}_{\text{CVT}}(\theta) = \frac{1}{|\mathcal{D}_{ul}|} \sum_{x_i \in \mathcal{D}_{ul}} \sum_{j=1}^{k} D(p_\theta(y|x_i), p_\theta^j(y|x_i))$$

where D is a distance function between probability distributions (I use KL divergence). I hold the primary module's prediction $p_\theta(y|x_i)$ fixed during training (i.e., I do not back-propagate through it) so the auxiliary modules learn to imitate the primary one, but not vice versa. CVT works by enhancing the model's representation learning. As the auxiliary modules train, the representations they take as input improve so they are useful for making predictions even when some of the model's inputs are not available. This in turn improves the primary prediction module, which is built on top of the same shared representations.

I combine the supervised and CVT losses into the total loss, $\mathcal{L} = \mathcal{L}_{\text{sup}} + \mathcal{L}_{\text{CVT}}$, and minimize it with stochastic gradient descent. In particular, I alternate minimizing $\mathcal{L}_{\text{sup}}$ over a minibatch of labeled examples and minimizing $\mathcal{L}_{\text{CVT}}$ over a minibatch of unlabeled

[2]As in knowledge distillation, soft targets (i.e., distributions over classes) tend to work better than one-hot hard targets for semi-supervised learning.

examples.

For most neural networks, adding a few additional prediction modules is computationally cheap compared to the portion of the model building up representations (such as an RNN or CNN). Therefore my method contributes little overhead to training time over other self-training approaches for most tasks. CVT does not change inference time or the number of parameters in the fully-trained model because the auxiliary prediction modules are only used during training.

4.1.2 Combining CVT with Multi-Task Learning

To make the model more general-purpose, CVT can easily be combined with multi-task learning by adding additional prediction modules for the other tasks on top of the shared Bi-LSTM encoder. During supervised learning, I randomly select a task and then update $\mathcal{L}_{\text{sup}}$ using a minibatch of labeled data for that task. When learning on the unlabeled data, I optimize $\mathcal{L}_{\text{CVT}}$ jointly across all tasks at once, first running inference with all the primary prediction modules and then learning from the predictions with all the auxiliary prediction modules. As before, the model alternates training on minibatches of labeled and unlabeled examples.

Examples labeled across many tasks are useful for multi-task systems to learn from, but most datasets are only labeled with one task. A benefit of multi-task CVT is that the model creates (artificial) all-tasks-labeled examples from unlabeled data. This significantly improves the model's data efficiency and training time. Since running prediction modules is computationally cheap, computing $\mathcal{L}_{\text{CVT}}$ is not much slower for many tasks than it is for a single one. However, I find the all-tasks-labeled examples substantially speed up model convergence. For example, my model trained on six tasks takes about three times as long to converge as the average model trained on one task, a 50% decrease in total training time.

4.2 Applying Cross-View Training

CVT relies on auxiliary prediction modules that have restricted views of the input. In this section, I describe specific constructions of the auxiliary prediction modules that are effective for sequence tagging, dependency parsing, and sequence-to-sequence learning.

4.2.1 Bi-LSTM Sentence Encoder

All of my models use a two-layer CNN-BiLSTM (Chiu and Nichols, 2016; Ma and Hovy, 2016) sentence encoder. It takes as input a sequence of words $x_i = [x_i^1, x_i^2, ..., x_i^T]$. First, each word is represented as the sum of an embedding vector and the output of a character-level Convolutional Neural Network, resulting in a sequence of vectors $v = [v^1, v^2, ..., v^T]$. The encoder applies a two-layer bidirectional LSTM (Graves and Schmidhuber, 2005) to these representations. The first layer runs a Long Short-Term Memory unit (Hochreiter and Schmidhuber, 1997) in the forward direction (taking v^t as input at each step t) and the backward direction (taking v^{T-t+1} at each step) to produce vector sequences $[\overrightarrow{\boldsymbol{h}}_1^1, \overrightarrow{\boldsymbol{h}}_1^2, ... \overrightarrow{\boldsymbol{h}}_1^T]$ and $[\overleftarrow{\boldsymbol{h}}_1^1, \overleftarrow{\boldsymbol{h}}_1^2, ... \overleftarrow{\boldsymbol{h}}_1^T]$. The output of the Bi-LSTM is the concatenation of these vectors: $h_1 = [\overrightarrow{\boldsymbol{h}}_1^1 \oplus \overleftarrow{\boldsymbol{h}}_1^1, ..., \overrightarrow{\boldsymbol{h}}_1^T \oplus \overleftarrow{\boldsymbol{h}}_1^T]$. The second Bi-LSTM layer works the same, producing outputs h_2, except it takes h_1 as input instead of v.

4.2.2 CVT for Sequence Tagging

In sequence tagging, each token x_i^t has a corresponding label y_i^t. The primary prediction module for sequence tagging produces a probability distribution over classes for the t^{th} label using a one-hidden-layer neural network applied to the corresponding encoder outputs:

$$p(y^t|x_i) = \texttt{NN}(h_1^t \oplus h_2^t) = \text{softmax}(U \cdot \text{ReLU}(W(h_1^t \oplus h_2^t)) + b)$$

The auxiliary prediction modules take $\overrightarrow{\boldsymbol{h}}_1(x_i)$ and $\overleftarrow{\boldsymbol{h}}_1(x_i)$, the outputs of the forward and backward LSTMs in the first[3] Bi-LSTM layer, as inputs. I add the following four

[3]Modules taking inputs from the second Bi-LSTM layer would not have restricted views because information about the whole sentence gets propagated through the first layer.

auxiliary prediction modules to the model (see Figure 4.2):

$$\begin{aligned}
p_\theta^{\text{fwd}}(y^t|x_i) &= \text{NN}^{\text{fwd}}(\overrightarrow{\boldsymbol{h}}_1^t(x_i)) \\
p_\theta^{\text{bwd}}(y^t|x_i) &= \text{NN}^{\text{bwd}}(\overleftarrow{\boldsymbol{h}}_1^t(x_i)) \\
p_\theta^{\text{future}}(y^t|x_i) &= \text{NN}^{\text{future}}(\overrightarrow{\boldsymbol{h}}_1^{t-1}(x_i)) \\
p_\theta^{\text{past}}(y^t|x_i) &= \text{NN}^{\text{past}}(\overleftarrow{\boldsymbol{h}}_1^{t+1}(x_i))
\end{aligned}$$

The "forward" module makes each prediction without seeing the right context of the current token. The "future" module makes each prediction without the right context or the current token itself. Therefore it works like a neural language model that, instead of predicting which token comes next in the sequence, predicts which class of token comes next. The "backward" and "past" modules are analogous.

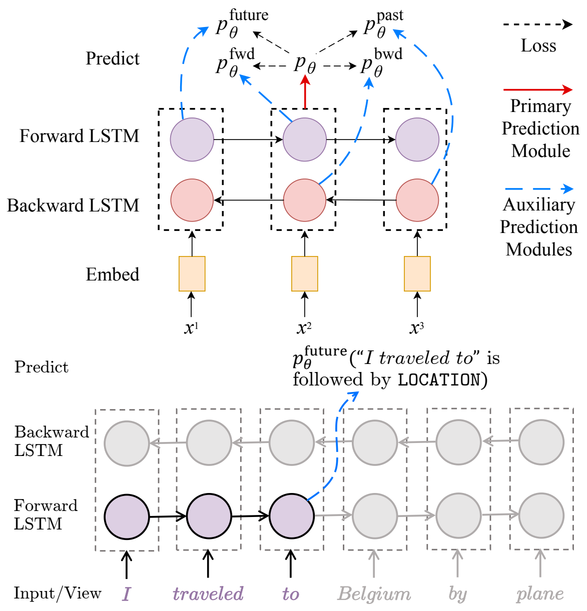


Figure 4.2: Above: auxiliary prediction modules for sequence tagging models. Each one sees a restricted view of the input. For example, the "future" prediction module does not see any context to the right of the current token or the current token itself when predicting that token's label. For simplicity, I only show a one layer Bi-LSTM encoder and only show the model's predictions for a single time step. Below: an illustration of how the prediction modules take advantage of the neural network architecture to create restricted input views.

4.2.3 CVT for Dependency Parsing

In a dependency parse, words in a sentence are treated as nodes in a graph. Typed directed edges connect the words, forming a tree structure describing the syntactic structure of the sentence. In particular, each word x_i^t in a sentence $x_i = x_i^1, ..., x_i^T$ receives exactly one in-going edge (u, t, r) going from a word x_i^u or additional root token (called the "head") to it (the "dependent") of type r (the "relation"). I use a graph-based dependency parser similar to the one from Dozat and Manning (2017). This treats dependency parsing as a classification task where the goal is to predict which in-going edge $y_i^t = (u, t, r)$ connects to each word x_i^t.

First, the representations produced by a Bi-LSTM encoder for the candidate head and dependent are passed through separate hidden layers. A bilinear classifier applied to these representations produces a score for each candidate edge. Lastly, these scores are passed through a softmax layer to produce probabilities. Mathematically, the probability of an edge is given as:

$$p_\theta((u,t,r)|x_i) \propto e^{s(h_1^u(x_i) \oplus h_2^u(x_i), h_1^t(x_i) \oplus h_2^t(x_i), r)}$$

where s is the scoring function:

$$\begin{aligned} s(z_1, z_2, r) = \ & \text{ReLU}(W_{\text{head}} z_1 + b_{\text{head}})(W_r + W) \\ & \text{ReLU}(W_{\text{dep}} z_2 + b_{\text{dep}}) \end{aligned}$$

The bilinear classifier uses a weight matrix W_r specific to the candidate relation as well as a weight matrix W shared across all relations. Note that unlike in most prior work, my dependency parser only takes words as inputs, not also part-of-speech tags.

I add four auxiliary prediction modules to my model for cross-view training:

$$\begin{aligned} p_\theta^{\text{fwd-fwd}}((u,t,r)|x_i) &\propto e^{s^{\text{fwd-fwd}}(\overrightarrow{\boldsymbol{h}}_1^u(x_i), \overrightarrow{\boldsymbol{h}}_1^t(x_i), r)} \\ p_\theta^{\text{fwd-bwd}}((u,t,r)|x_i) &\propto e^{s^{\text{fwd-bwd}}(\overrightarrow{\boldsymbol{h}}_1^u(x_i), \overleftarrow{\boldsymbol{h}}_1^t(x_i), r)} \\ p_\theta^{\text{bwd-fwd}}((u,t,r)|x_i) &\propto e^{s^{\text{bwd-fwd}}(\overleftarrow{\boldsymbol{h}}_1^u(x_i), \overrightarrow{\boldsymbol{h}}_1^t(x_i), r)} \\ p_\theta^{\text{bwd-bwd}}((u,t,r)|x_i) &\propto e^{s^{\text{bwd-bwd}}(\overleftarrow{\boldsymbol{h}}_1^u(x_i), \overleftarrow{\boldsymbol{h}}_1^t(x_i), r)} \end{aligned}$$

Each one has some missing context (not seeing either the preceding or following words) for the candidate head and candidate dependent.

4.2.4 CVT for Sequence-to-Sequence Learning

For sequence-to-sequence tasks, I use an LSTM encoder-decoder model with attention (Sutskever et al., 2014; Bahdanau et al., 2015). Each example consists of an input (source) sequence $x_i = x_i^1, ..., x_i^T$ and output (target) sequence $y_i = y_i^1, ..., y_i^K$. The encoder's representations are passed into an LSTM decoder using a bilinear attention mechanism (Luong et al., 2015b). In particular, at each time step t the decoder computes an attention distribution over source sequence hidden states as $\alpha_j \propto e^{h^j W_\alpha \bar{h}^t}$ where $\bar{h}^t$ is the decoder's current hidden state. The source hidden states weighted by the attention distribution form a context vector: $c_t = \sum_j \alpha_j h^j$. Next, the context vector and current hidden state are combined into an attention vector $a_t = \tanh(W_a[c_t, h_t])$. Lastly, a softmax layer predicts the next token in the output sequence: $p(y_i^t|y_i^{<t}, x_i) = \text{softmax}(W_s a_t)$.

I add two auxiliary decoders when applying CVT. The auxiliary decoders share embedding and LSTM parameters with the primary decoder, but have different parameters for the attention mechanisms and softmax layers. For the first one, I restrict its view of the input by applying attention dropout, randomly zeroing out a fraction of its attention weights. The second one is trained to predict the next word in the target sequence rather than the current one: $p_\theta^{\text{future}}(y_i^t|y_i^{<t}, x_i) = \text{softmax}(W_s^{\text{future}} a_{t-1}^{\text{future}})$. Since there is no target sequence for unlabeled examples, I cannot apply teacher forcing to get an output distribution over the vocabulary from the primary decoder at each time step. Instead, I produce hard targets for the auxiliary modules by running the primary decoder with beam search on the input sequence. This idea has previously been applied to sequence-level knowledge distillation by Kim and Rush (2016) and makes the training procedure similar to back-translation (Sennrich et al., 2016a).

4.3 Experiments

I compare Cross-View Training against several strong baselines on seven tasks: Combinatory Categorial Grammar (CCG) Supertagging, Text Chunking, Named Entity Recognition (NER), Fine-Grained NER (FGN), Part-of-Speech (POS) Tagging, Dependency Parsing, and Machine Translation. See background section 2.6 for more details on the tasks. I use the 1 Billion Word Language Model Benchmark (Chelba et al., 2014) as a pool of unlabeled sentences for semi-supervised learning.

4.3.1 Model Details

I apply dropout during training, but not when running the primary prediction module to produce soft targets on unlabeled examples. In addition to the auxiliary prediction modules listed in Section 4.2, I find it slightly improves results to add another one that sees the whole input rather than a subset (but unlike the primary prediction module, does have dropout applied to its representations). Unless indicated otherwise, my models have LSTMs with 1024-sized hidden states and 512-sized projection layers.

Sequence Tagging. For Chunking and Named Entity Recognition, I use a BIOES tagging scheme. I apply label smoothing (Szegedy et al., 2016; Pereyra et al., 2017) with a rate of 0.1 to the target labels when training on the labeled data.

Dependency Parsing. I omit punctuation from evaluation, which is standard practice for the PTB-SD 3.3.0 dataset. ROOT is represented with a fixed vector $h_{\texttt{ROOT}}$ instead of using a vector from the encoder, but otherwise dependencies coming from ROOT are scored the same way as the other dependencies.

Machine Translation. I apply dropout to the output of each LSTM layer in the decoder. My implementation is heavily based off of a Google NMT Tutorial[4] (Luong et al., 2017). I attribute my significantly better results to using pre-trained word embeddings, a character-level CNN, a larger model, stronger regularization, and better hyperparameter tuning. Target words occurring 5 or fewer times in the train set are replaced with a `UNK` token (but not during evaluation). I use a beam size of 10 when performing beam search. I found it

[4] `https://github.com/tensorflow/nmt`

slightly beneficial to apply label smoothing with a rate of 0.1 to the teacher's predictions (unlike my other tasks, the teacher only provides hard targets to the students for translation).

Multi-Task Learning. Several of my datasets are constructed from the Penn Treebank. However, I treat them as separate rather than providing examples labeled across multiple tasks to my model during supervised training. Furthermore, the Penn Treebank tasks do not all use the same train/dev/test splits. I ensure the training split of one task never overlaps the evaluation split of another by discarding the overlapping examples from the train sets.

Other Details. I apply dropout (Hinton et al., 2012) to the word embeddings and outputs of each Bi-LSTM. I use an exponential-moving-average (EMA) of the model weights from training for the final model; I found this to slightly improve accuracy and significantly reduce the variance in accuracy between models trained with different random initializations. The model is trained using SGD with momentum (Polyak, 1964; Sutskever et al., 2013). Word embeddings are initialized with GloVe vectors (Pennington et al., 2014) and fine-tuned during training. The full set of model hyperparameters are listed in Table 4.1.

Parameter	Value
Word Embeddings Initializition	300d GloVe 6B
Character Embedding Size	50
Character CNN Filter Widths	[2, 3, 4]
Character CNN Num Filters	300 (100 per filter width)
Encoder LSTM sizes	1024 for the first layer, 512 for the second one
Encoder LSTM sizes, "Large" model	4096 for the first layer, 2048 for the second one
LSTM projection layer size	512
Hidden layer sizes	512
Dropout	0.5 for labeled examples, 0.8 for unlabeled examples
EMA coefficient	0.998
Learning rate	$0.5/(1 + 0.005t^{0.5})$ (t is number of SGD updates so far)
Momentum	0.9
Batch size	64 sentences

Table 4.1: Hyperparameters for the model.

4.3.2 Baselines

I compare CVT with the following other semi-supervised learning algorithms. Unless indicated otherwise, baselines were run with the same architecture and hyperparameters as the CVT model.

Word Dropout. In this method, I only train the primary prediction module. When acting as a teacher it is run as normal, but when acting as a student, I randomly replace some of the input words with a `REMOVED` token with probability 0.1. This is similar to CVT in that it exposes the model to a restricted view of the input. However, it is less data efficient. By carefully designing the auxiliary prediction modules, it is possible to train the auxiliary prediction modules to match the primary one across many different views of the input a once, rather than just one view at a time.

Virtual Adversarial Training (VAT). VAT (Miyato et al., 2016) works like word dropout, but adds noise to the word embeddings of the student instead of dropping out words. Notably, the noise is chosen adversarially so it most changes the model's prediction. This method was applied successfully to semi-supervised text classification by Miyato et al. (2017a). I set the norm of the perturbation to be 1.5 for CCG, 1.0 for Dependency Parsing, and 0.5 for the other tasks (these values worked best on the dev sets). Otherwise, the implementation is as described in (Miyato et al., 2017a); I based my implementation off of their code[5]. I was unable to successfully apply VAT to machine translation, perhaps because the student is provided hard targets for that task.

ELMo. ELMo incorporates the representations from a large separately-trained language model into a task-specific model. My implementation follows Peters et al. (2018). When combining ELMo with multi-task learning, I allow each task to learn its own weights for the ELMo embeddings going into each prediction module. I found applying dropout to the ELMo embeddings was crucial for achieving good performance. I did not directly compare with later pre-trained models such as BERT because this work pre-dates them.

[5]`https://github.com/tensorflow/models/tree/master/research/adversarial_text`

Method	CCG	Chunk	NER	FGN	POS	Dep. Parse		Translate
	Acc.	F1	F1	F1	Acc.	UAS	LAS	BLEU
Shortcut LSTM (Wu et al., 2017)	95.1				97.53			
ID-CNN-CRF (Strubell et al., 2017)			90.7	86.8				
JMT† (Hashimoto et al., 2017)		95.8			97.55	94.7	92.9	
TagLM* (Peters et al., 2017)		96.4	91.9					
ELMo* (Peters et al., 2018)			92.2					
Biaffine (Dozat and Manning, 2017)						95.7	94.1	
Stack Pointer (Ma et al., 2018)						95.9	94.2	
Stanford (Luong and Manning, 2015)								23.3
Google (Luong et al., 2017)								26.1
Supervised	94.9	95.1	91.2	87.5	97.60	95.1	93.3	28.9
Virtual Adversarial Training*	95.1	95.1	91.8	87.9	97.64	95.4	93.7	–
Word Dropout*	95.2	95.8	92.1	88.1	97.66	95.6	93.8	29.3
ELMo*	95.8	96.5	92.2	88.5	97.72	96.2	94.4	29.3
ELMo + Multi-task*†	95.9	96.8	92.3	88.4	**97.79**	96.4	94.8	–
CVT*	95.7	96.6	92.3	88.7	97.70	95.9	94.1	**29.6**
CVT + Multi-task*†	96.0	96.9	92.4	88.4	97.76	96.4	94.8	–
CVT + Multi-task + Large*†	**96.1**	**97.0**	**92.6**	**88.8**	97.74	**96.6**	**95.0**	–

Table 4.2: Comparison of CVT and earlier approaches on the test sets. I report the mean score over 5 runs. Standard deviations in score are around 0.1 for NER, FGN, and translation, 0.02 for POS, and 0.05 for the other tasks. See the appendix for results with them included. The +Large model has four times as many hidden units as the others, making it similar in size to the models when ELMo is included. * denotes semi-supervised and † denotes multi-task.

Furthermore, they use Transformer networks instead of LSTMs and larger amounts of unlabeled data, making the comparison less direct. I applied dropout to the ELMo embeddings before they are incorporated into the rest of the model. When training the multi-task ELMo model, each prediction module has its own set of softmax-normalized weights (s_j^{task} in (Peters et al., 2018)) for the ELMo embeddings going into the task-specific prediction modules. All tasks share the same s_j weights for the ELMo embeddings going into the shared Bi-LSTM encoder.

4.3.3 Results

Results are shown in Table 4.2. CVT on its own outperforms or is comparable to the best previously published results on all tasks. Figure 4.3 shows an example win for CVT over supervised learning.

Of the prior results listed in Table 4.2, only TagLM and ELMo are semi-supervised. These methods first train an enormous language model on unlabeled data[6] and incorporate the representations produced by the language model into a supervised classifier. My base models use 1024 hidden units in their LSTMs (compared to 4096 in ELMo), require fewer training steps (around one pass over the billion-word benchmark rather than 10 passes), and do not require a pipelined training procedure. Therefore, although they perform on par with ELMo, they are faster and simpler to train. Increasing the size of my CVT+Multi-task model so it has 4096 units in its LSTMs like ELMo improves results further so they are significantly better than the ELMo+Multi-task ones, with the model still training over 10 times faster than ELMo.

CVT even holds up well when compared with BERT, a subsequent model using a large Transformer architecture and much more unlabeled data. BERT-Large takes well over 500x the compute of CVT to train, but only performs slightly better at named entity recognition (92.8 F1; Devlin et al. (2019)), dependency parsing (95.3 LAS; Zhou and Hai (2019)), part-of-speech tagging (97.8 accuracy; Jiang et al. (2020a)), and text chunking (97.3 F1; Liu et al. (2019d)). The fact that an “old fashioned” LSTM can come close to the performance of BERT on these tasks suggests that performance is quite saturated, but also speaks to the effectiveness of CVT as a semi-supervised learning method.

CVT + Multi-Task. I train a single shared-encoder CVT model to perform all of the tasks except machine translation (as it is quite different and requires more training time than the other ones). Multi-task learning improves results on all of the tasks except fine-grained NER, sometimes by large margins. Prior work on many-task NLP such as Hashimoto et al. (2017) uses complicated architectures and training algorithms. My result shows that simple parameter sharing can be enough for effective many-task learning when the model is big and trained on a large amount of data.

[6] In fact, they use the same billion-word benchmark as we do for unlabeled data.

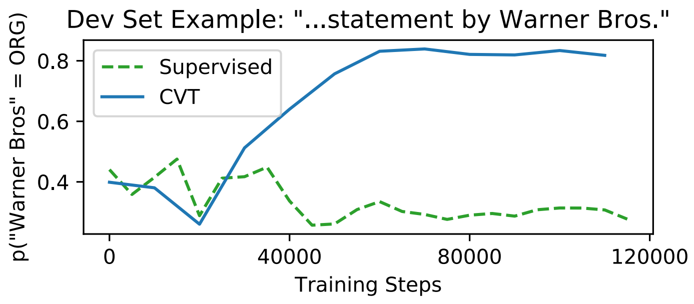


Figure 4.3: An NER example that CVT classifies correctly but supervised learning does not. “Warner” only occurs as a last name in the train set, so the supervised model classifies “Warner Bros” as a person. The CVT model also mistakenly classifies “Warner Bros” as a person to start with, but as it sees more of the unlabeled data (in which “Warner” occurs thousands of times) it learns that “Warner Bros” is an organization.

Interestingly, multi-task learning works better in conjunction with CVT than with ELMo. I hypothesize that the ELMo models quickly fit to the data primarily using the ELMo vectors, which perhaps hinders the model from learning effective representations that transfer across tasks. I also believe CVT alleviates the danger of the model “forgetting” one task while training on the other ones, a well-known problem in many-task learning (Kirkpatrick et al., 2017). During multi-task CVT, the model makes predictions about unlabeled examples across all tasks, creating (artificial) all-tasks-labeled examples, so the model does not only see one task at a time. In fact, multi-task learning plus self training is similar to the Learning without Forgetting algorithm (Li and Hoiem, 2016), which trains the model to keep its predictions on an old task unchanged when learning a new task. To test the value of all-tasks-labeled examples, I trained a multi-task CVT model that only computes $\mathcal{L}_{\text{CVT}}$ on one task at a time (chosen randomly for each unlabeled minibatch) instead of for all tasks in parallel. The one-at-a-time model performs substantially worse (see Table 4.3).

Model Generalization. In order to evaluate how my models generalize to the dev set from the train set, I plot the dev vs. train accuracy for my different methods as they learn (see Figure 4.4). Both CVT and multi-task learning improve model generalization: for the same train accuracy, the models get better dev accuracy than purely supervised learning. Interestingly, CVT continues to improve in dev set accuracy while close to 100% train accuracy for CCG, Chunking, and NER, perhaps because the model is still learning from

Model	CCG	Chnk	NER	FGN	POS	Dep.
CVT-MT	95.7	97.4	96.0	86.7	97.74	94.4
w/out all-labeled	95.4	97.1	95.6	86.3	97.71	94.1

Table 4.3: Dev set performance of multi-task CVT with and without producing all-tasks-labeled examples.

unlabeled data even when it has completely fit to the train set. I also show results for a smaller multi-task + CVT model. Although it generalizes at least as well as the larger one, it halts making progress on the train set earlier. This suggests it is important to use sufficiently large neural networks for multi-task learning: otherwise the model does not have the capacity to fit to all the training data.

Auxiliary Prediction Module Ablation. I briefly explore which auxiliary prediction modules are more important for the sequence tagging tasks in Table 4.4. I find that both kinds of auxiliary prediction modules improve performance, but that the future and past modules improve results more than the forward and backward ones, perhaps because they see a more restricted and challenging view of the input.

Model	CCG	Chnk	NER	FGN	POS
Supervised	94.8	95.5	95.0	86.0	97.59
CVT	95.6	97.0	95.9	87.3	97.66
no fwd/bwd	–0.1	–0.2	–0.2	–0.1	–0.01
no future/past	–0.3	–0.4	–0.4	–0.3	–0.04

Table 4.4: Ablation study on auxiliary prediction modules for sequence tagging.

Training Models on Small Datasets. I explore how CVT scales with dataset size by varying the amount of training data the model has access to. Unsurprisingly, the improvement of CVT over purely supervised learning grows larger as the amount of labeled data decreases (see Figure 4.5, left). Using only 25% of the labeled data, my approach already performs as well or better than a fully supervised model using 100% of the training data, demonstrating that CVT is particularly useful on small datasets.

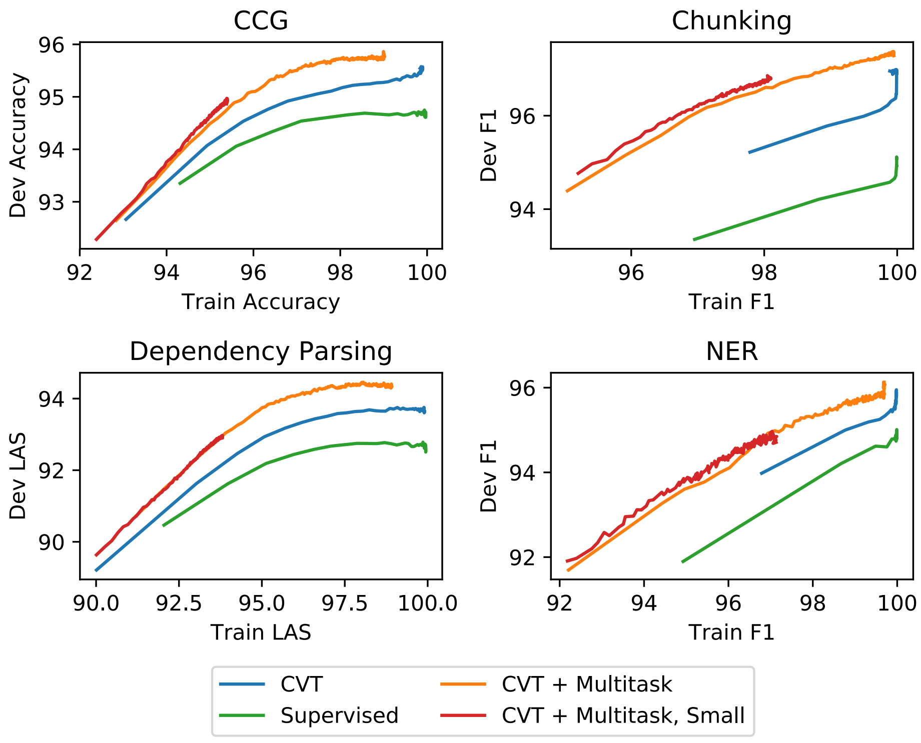


Figure 4.4: Dev set vs. Train set accuracy for various methods. The "small" model has 1/4 the LSTM hidden state size of the other ones (256 instead of 1024).

Training Larger Models. Most sequence taggers and dependency parsers in prior work use small LSTMs (hidden state sizes of around 300) because larger models yield little to no gains in performance (Reimers and Gurevych, 2017). I found my own supervised approaches also do not benefit greatly from increasing the model size. In contrast, when using CVT accuracy scales better with model size (see Figure 4.5, right). This finding suggests the appropriate semi-supervised learning methods may enable the development of larger, more sophisticated models for NLP tasks with limited amounts of labeled data.

Generalizable Representations. Lastly, I explore training the CVT+multi-task model on five tasks, freezing the encoder, and then only training a prediction module on the sixth task. This tests whether the encoder's representations generalize to a new task not seen during its training. Only training the prediction module is very fast because (1) the encoder (which is by far the slowest part of the model) has to be run over each example only once

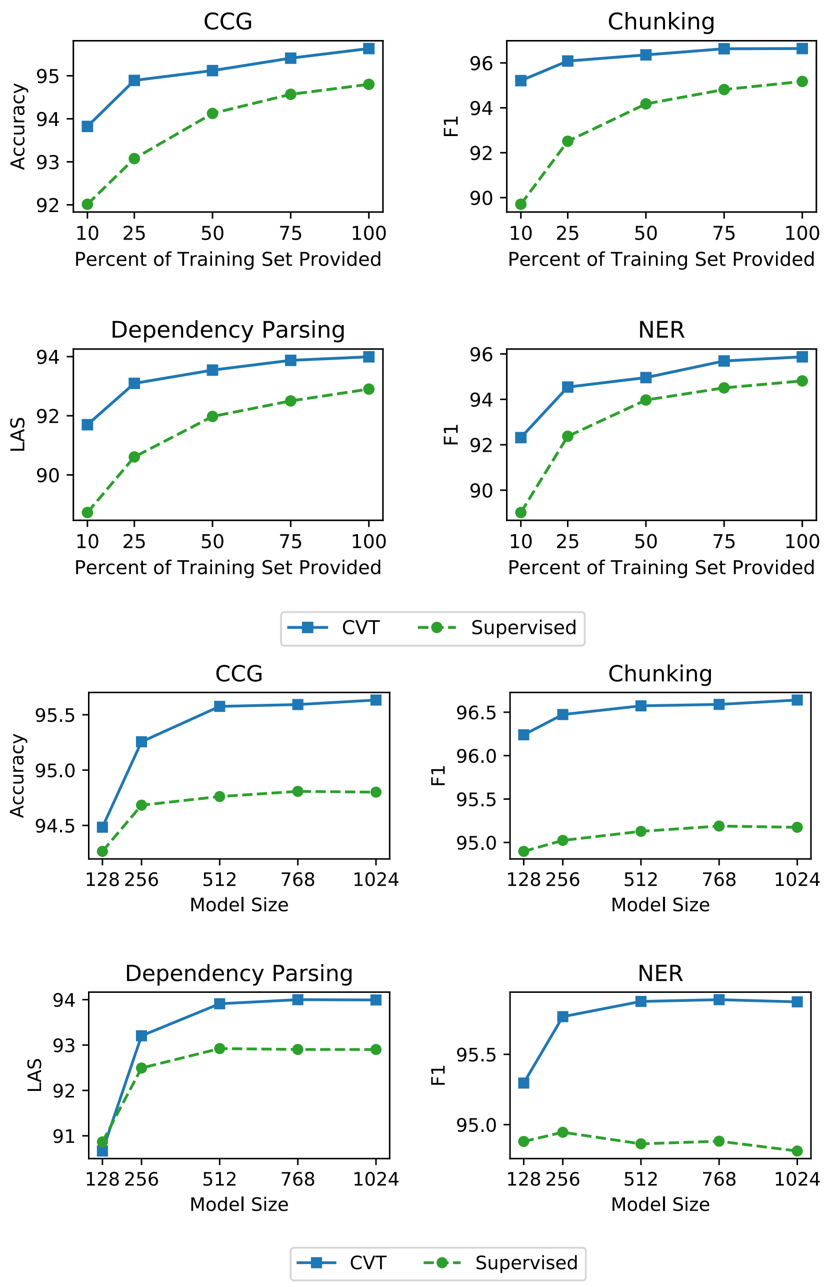


Figure 4.5: Above: Dev set performance vs. percent of the training set provided to the model. Below: Dev set performance vs. model size. The x axis shows the number of hidden units in the LSTM layers; the projection layers and other hidden layers in the network are half that size. Points correspond to the mean of three runs.

and (2) I do not backpropagate into the encoder. Results are shown in Table 4.5.

Model	CCG	Chnk	NER	FGN	POS	Dep.
Supervised	94.8	95.6	95.0	86.0	97.59	92.9
CVT-MT frozen	95.1	96.6	94.6	83.2	97.66	92.5
ELMo frozen	94.3	92.2	91.3	80.6	97.50	89.4

Table 4.5: Comparison of single-task models on the dev sets. "CVT-MT frozen" means I pre-train a CVT + multi-task model on five tasks, and then train only the prediction module for the sixth. "ELMo frozen" means I train prediction modules (but no LSTMs) on top of ELMo embeddings.

Training only a prediction module on top of multi-task representations works remarkably well, outperforming ELMo embeddings and sometimes even a vanilla supervised model, showing the multi-task model is building up effective representations for language. In particular, the representations could be used like skip-thought vectors (Kiros et al., 2015) to quickly train models on new tasks without slow representation learning.

4.4 CVT for Image Recognition

Although the focus of my work is on NLP, I also applied CVT to image recognition and found it performs competitively with previous methods. Most of the semi-supervised image recognition approaches I compare against rely on the inputs being continuous, so they would be difficult to apply to text. More specifically, consistency regularization methods (Sajjadi et al., 2016; Laine and Aila, 2017; Miyato et al., 2017b) rely on adding continuous noise and applying image-specific transformations like cropping to inputs, GANs (Salimans et al., 2016; Wei et al., 2018) are very difficult to train on text due to its discrete nature, and mixup (Zhang et al., 2018; Verma et al., 2019) requires a way of smoothly interpolating between different inputs.

Approach. My image recognition models are based on Convolutional Neural Networks, which produce a set of features $H(x_i) \in \mathbb{R}^{n \times n \times d}$ from an image x_i. The first two dimensions of H index into the spatial coordinates of feature vectors and d is the size of the feature vectors. For shallower CNNs, a particular feature vector corresponds to a region of the input image. For example, $H_{0,0}$ would be a d-dimensional vector of features extracted from the upper left corner. For deeper CNNs, a particular feature vector would be extracted from the whole image, but still only use a "region" of the representations from an earlier layer. The CNNs in my experiments are all in the first category.

The primary prediction layers of my CNNs take as input the mean of H over the first two dimensions, which results in a d-dimensional vector that is fed into a softmax layer:

$$p_\theta(y|x_i) = \text{softmax}(W\texttt{global\_average\_pool}(H) + b)$$

CVT adds n^2 auxiliary prediction layers to the top of the CNN. The j^{th} layer takes a single feature vector as input:

$$p^j_\theta(y|x_i) = \text{softmax}(W^j H_{\lfloor j/n \rfloor, j \bmod n} + b_j)$$

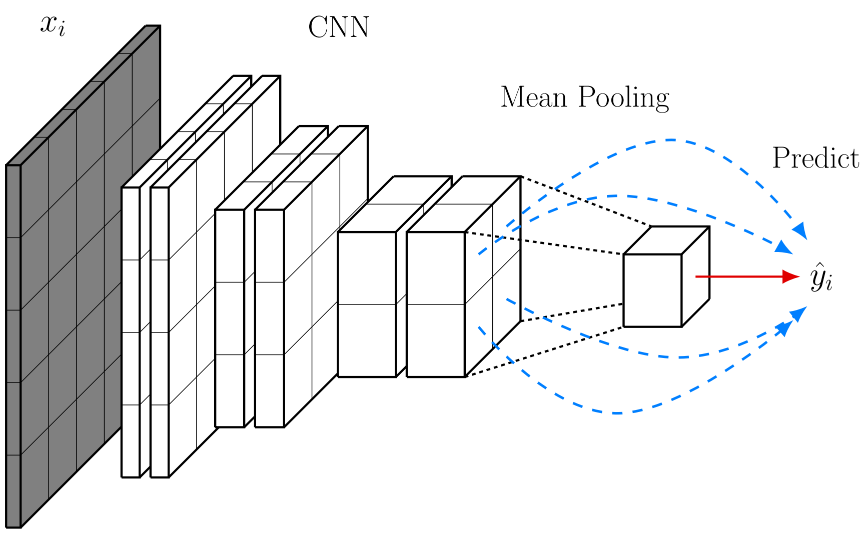


Figure 4.6: Auxiliary prediction modules (dashed blue arrows) and primary prediction module (solid red arrow) for image recognition. Each auxiliary module sees a restricted patch of the input image.

Data. I evaluated my models on the CIFAR-10 dataset (Krizhnevsky and Hinton, 2009) . Following previous work, I make the datasets semi-supervised by only using the provided labels for a subset of the examples in the training set; the rest are treated as unlabeled examples.

Model. I use the convolutional neural network from Miyato et al. (2017b), adapting their TensorFlow implementation[7]. Their model contains 9 convolutional layers and 2 max pooling layers. See Appendix D of Miyato et al.'s paper for more details.

I add 36 auxiliary softmax layers to the 6×6 collection of feature vectors produced by the CNN. Each auxiliary layer sees a patch of the image ranging in size from 21×21 pixels (the corner) to 29×29 pixels (the center) of the 32×32 pixel images. For some experiments, I combine CVT with standard consistency regularization by adding a perturbation (e.g., a small random vector) to the student's inputs when computing $\mathcal{L}_{\text{CVT}}$.

Results. The results are shown in Table 4.6. Unsurprisingly, adding continuous noise to the inputs works much better with images, where the inputs are naturally continuous, than with language. Therefore I see much better results from VAT on semi-supervised CIFAR-10 compared to on my NLP tasks. However, I still find incorporating CVT improves over models without CVT. My CVT + VAT models are competitive with current start-of-the-art approaches. I found the gains from CVT are larger when no data augmentation is applied, perhaps because random translations of the input expose the model to different "views" in a similar manner as with CVT.

Model Analysis. To understand why CVT produces better results, I compare the behavior of the VAT and CVT (with adversarial noise) models trained on CIFAR-10. First, I record the average absolute value of each feature vector produced by the CNNs when they run over the test set. As shown in the left of Figure 4.7, the CVT model has higher activation strengths for the feature vectors corresponding to the edges of the image. I hypothesize that the VAT model fits to the data while primarily using the center of the image, where the most discriminative information is contained. This results in less effective feature vectors for the outside regions. In contrast, the model with CVT must learn meaningful representations for the edge regions in order to train the corresponding auxiliary softmax layers. As these feature vectors are more useful, their magnitudes become larger so they contribute more to the final representation produced by the global average pool.

[7] `https://github.com/takerum/vat_tf`

Method	CIFAR-10	CIFAR-10+
	4000 labels	
GAN (Salimans et al., 2016)	–	18.63 ± 2.32
Stochastic Transformations (Sajjadi et al., 2016)	–	11.29 ± 0.24
Π model (Laine and Aila, 2017)	16.55 ± 0.29	12.36 ± 0.31
Temporal Ensemble (Laine and Aila, 2017)	–	12.16 ± 0.24
Mean Teacher (Tarvainen and Valpola, 2017b)	–	12.31 ± 0.28
Complement GAN (Dai et al., 2017)	14.41 ± 0.30	–
VAT (Miyato et al., 2017b)	13.15	10.55
VAdD (Park et al., 2018)	–	11.68 ± 0.19
VAdD + VAT (Park et al., 2018)	–	**10.07** ± **0.11**
SNGT + Π model (Luong et al., 2017)	13.62 ± 0.17	11.00 ± 0.36
SNGT + VAT (Luong et al., 2017)	**12.49** ± **0.36**	**9.89** ± **0.34**
Consistency + WGAN (Wei et al., 2018)	–	**9.98** ± **0.21**
Manifold Mixup (Verma et al., 2019)	–	**10.26** ± **0.32**
Supervised	23.61 ± 0.60	19.61 ± 0.56
VAT (ours)	13.29 ± 0.33	10.90 ± 0.31
CVT, no input perturbation	14.63 ± 0.20	12.44 ± 0.27
CVT, random input perturbation	13.80 ± 0.30	11.10 ± 0.26
CVT, adversarial input perturbation	**12.01** ± **0.11**	**10.11** ± **0.15**

Table 4.6: Error rates on semi-supervised CIFAR-10. I report means and standard deviations from 5 runs. CIFAR-10+ refers to results where data augmentation (random translations of the input image) was applied. For some of my models I add a random or adversarially chosen perturbation to the student model's inputs, which is done in most consistency regularization methods.

To compare the discriminatory power of the feature vectors, I freeze the weights of the CNNs and add auxiliary softmax layers that are trained from scratch. I then measure the accuracies of the added layers (see the center and right of Figure 4.7). Unsurprisingly, the VAT model, which only learns representations that will be useful after the average pool, has much lower accuracies from individual feature vectors. The difference is particularly striking in the sides and corners, where CVT accuracies are around 50% higher (they are about 25% higher in the center). This finding further indicates that CVT is improving the model's representations, particularly for the outside parts of images.

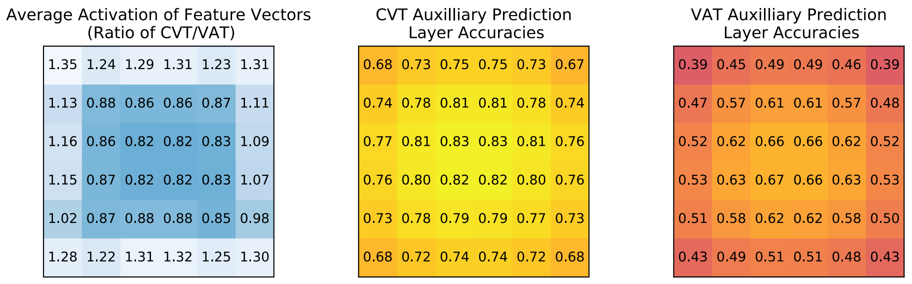


Figure 4.7: Left: Ratio between the average activation of feature vectors from final layer of the CVT CNNs divided by the average from the VAT CNNs. Each square in a grid represents a single feature vector. Brighter means the feature vectors from the CVT model are more activated. Center, Right: Accuracy of prediction layers taking a single feature vector as input. The CVT model makes more use of the outside of the image and produces better representations for those regions

4.5 Negative Results

I briefly describe a few ideas I implemented that did not seem to be effective in initial experiments. Note these findings are from early one-off experiments. I did not pursue them further after my first attempts did not pan out, so it is possible that some of these approaches could be effective with the proper adjustments and tuning.

- **Hard vs soft targets:** Classic self-training algorithms train the student model with one-hot "hard" targets corresponding to the teacher's highest probability prediction. In my experiments, this decreased performance compared to using soft targets. This finding is consistent with research on knowledge distillation (Hinton et al., 2015; Furlanello et al., 2018) where soft targets also work notably better than hard targets.

- **Confidence thresholding:** Classic self-training often only trains the student on a subset of the unlabeled examples on which the teacher has confident predictions (i.e., the output distribution has low entropy). I tried both "hard" (where the student ignores low-confidence examples) and "soft" (where examples are weighted according to the teacher's confidence) versions of this for training my models, but they did not seem to improve performance.

- **Mean Teacher:** The Mean Teacher method (Tarvainen and Valpola, 2017b) tracks an exponential moving average (EMA) of model weights, which are used to produce targets for the students. The idea is that these targets may be better quality due to a self-ensembling effect. However, I found this approach to have little to no benefit in my experiments, although using EMA model weights at test time did improve results slightly.

- **Purely supervised CVT:** Lastly, I explored adding cross-view losses to purely supervised classifiers. I hoped that adding auxiliary softmax layers with different views of the input would act as a regularizer on the model. However, I found little to no benefit from this approach. This negative result suggests that the gains from CVT are from the improved semi-supervised learning mechanism, not the additional prediction layers regularizing the model.

4.6 Conclusion

At their core, many semi-supervised learning algorithms rely on some form of self-training where models learn from their own predictions on unlabeled data. Although providing the model with new contexts it can learn from, standard self-training is somewhat circular: the model is being trained on precisely what it already predicts. Many semi-supervised learning algorithms for computer vision alleviate this issue through noising the input, but noising discrete data like text is difficult. I instead propose using different *views* of the input to help models effectively learn from their own predictions on unlabeled examples. The use of views makes it reminiscent of co-training, but CVT works particularly well with neural networks because it uses shared representations to transfer knowledge across views effectively. CVT pairs well with multi-task learning, where the model can create examples pseudo-labeled across all tasks, increasing sample efficiency because the model learns all the tasks simultaneously. Experiments show that CVT is effective at sequence tagging, dependency parsing, and machine translation, outperforming ELMo while requiring less than 10% of the compute and even performing competitively with BERT. This considerable efficiency advantage over pre-training comes from learning representations that are targeted towards specific tasks of interest as opposed to learning representations that are broadly useful through a highly general task such as language modeling. Together, these results show the value of traditional semi-supervised learning approaches despite the rise of pre-training in NLP. Indeed, subsequent work has shown that self-training can effectively be combined with pre-training to improve models (Du et al., 2020).

While CVT achieves great results, it is not always easy to apply. The model only learns a pre-set collection of tasks; using it for a new task requires the model be re-trained from scratch. Furthermore, the models require unlabeled examples to self-label. Although this is fine for the tasks I apply CVT to such as dependency parsing, where any sentence can be an unlabeled example, this sort of data is not available for other tasks such as question answering. Therefore in the next chapter, I switch focus to self-supervised pre-training while still following the goal of improving efficiency over previous methods.

Chapter 5

Efficient Self-Supervised Pre-Training with ELECTRA

So far, the methods I have presented train the model on a pre-specified set of tasks. An ideal transfer learning method would be more general, developing representations that can be transferred to almost any NLP task instead of only a few. Self-supervised representation learning moves towards this goal. These methods learn about language purely from unlabeled text and then can be quickly fine-tuned on NLP tasks. Although highly performant on many tasks, these methods are also computationally expensive to run. This chapter presents a self-supervised pre-training method that is far more efficient than previous approaches.

As discussed in background section 2.5, most existing pre-training methods rely on variants of masked language modeling. These can be viewed as learning denoising autoencoders (Vincent et al., 2008). They select a small subset of the unlabeled input sequence (typically 15%), mask the identities of those tokens (e.g., BERT (Devlin et al., 2019) replaces them with a special `[MASK]` token) or attention to those tokens (e.g., XLNet (Yang et al., 2019) uses attention masking to generate a subset of the sequence in a random order), and then train the network to recover the original input. While more effective than conventional language-model pre-training due to learning bidirectional representations, these masked language modeling (MLM) approaches incur a substantial compute cost because the network only learns from 15% of the tokens per example.

As an alternative, I propose *replaced token detection,* a pre-training task in which the model learns to distinguish real input tokens from plausible but synthetically generated replacements. Instead of masking, my method corrupts the input by replacing some tokens with samples from a noise distribution, which is typically the output of a small "generator" network such as a masked language model. This corruption procedure solves a mismatch in BERT (although not in XLNet) where the network sees artificial `[MASK]` tokens during pre-training but not when being fine-tuned on downstream tasks. I then pre-train the network as a discriminator that predicts for every token whether it is an original or a replacement. In contrast, MLM trains the network as a generator that predicts the original identities of the corrupted tokens. A key advantage of my discriminative task is that the model learns from *all* input tokens instead of just the small masked-out subset, making it more computationally efficient. Although this approach is reminiscent of training the discriminator of a GAN, the method is not adversarial in that the generator producing corrupted tokens is trained independently with maximum likelihood due to the difficulty of applying GANs to text (Caccia et al., 2020); the main challenge is that it is impossible to backpropagate through the discrete sampling of fake tokens from the noise distribution, which means sample-inefficient reinforcement learning methods must be used instead.

On the surface, replaced token detection looks quite different from masked language modeling, raising the question of whether it will lead to models learning the same rich representations of syntax and semantics captured by BERT. However, I show that in fact replaced token detection is essentially learning the same underlying task as BERT: the cloze task of predicting a token from its surrounding context. BERT implements the cloze task through masking tokens and then estimating their probabilities with an output softmax. Replaced token detection instead can be viewed as training an energy-based model (EBM) to perform the cloze task. EBMs learn an energy function that assigns low energy values to inputs in the data distribution and high energy values to other inputs. They are flexible because they do not have to compute normalized probabilities. In particular, the energy-based cloze model does not use an output softmax to estimate the probability of a token given its context like BERT. Instead, it produces a scalar energy score for each token where a low energy indicates the token is likely given its context. Energy based models cannot be trained with standard maximum-likelihood training because they produce unnormalized scores rather than probabilities. A popular alternative is noise-contrastive estimation (NCE;

Gutmann and Hyvärinen (2010)), which trains the EBM as a binary classifier distinguishing data vs. noise sample. An approximation of NCE training for the energy-based cloze model yields the replaced token detection task.

As in prior work, I apply replaced token detection to pre-train Transformer text encoders (Vaswani et al., 2017) that can be fine-tuned on downstream tasks. In particular, I develop two models using slight variants of replaced token detection. ELECTRA, which stands for "Efficiently Learning an Encoder that Classifies Token Replacements Accurately," uses replaced token detection with a standard binary classifier as the discriminator. Electric uses a slightly more complicated classifier, which makes it a true energy-based cloze model. Electric can more easily estimate the cloze probability of a token given its context, but performs slightly worse when fine-tuned on downstream tasks. I propose several extensions for these models that improve performance, as well as exploring alternative training algorithms. Then through a series of ablations, I show that learning from all input positions causes ELECTRA and Electric to train much faster than BERT.

Most previous pre-training methods require large amounts of compute to be effective, raising concerns about their cost, accessibility (most academic research labs and NLP practitioners do not have access to the hardware necessary for training these models), and even their environmental impact (Strubell et al., 2019). Since pre-training with more compute almost always results in better downstream accuracies, I argue an important consideration for pre-training methods should be compute efficiency as well as absolute downstream performance. After all, better downstream results can be achieved through just training bigger models without any improvements to the underlying method. From this viewpoint, I train ELECTRA and Electric models of various sizes and evaluate their downstream performance vs. their compute requirement. In particular, I run experiments on the GLUE natural language understanding benchmark (Wang et al., 2019) and SQuAD question answering benchmark (Rajpurkar et al., 2016). ELECTRA substantially outperforms MLM-based methods such as BERT and XLNet given the same model size, data, and compute (see Figure 5.1). It typically trains at least 4x faster than masked language models, meaning equivalent accuracy can be achieved at 1/4 the cost.

ELECTRA's efficiency gains hold at both ends of the compute spectrum. At small scale, I build an ELECTRA-Small model that can be trained on 1 GPU in 4 days.[1] ELECTRA-Small outperforms a comparably small BERT model by 5 points on GLUE, and even outperforms the much larger GPT model (Radford et al., 2018). My approach also works well at large scale, where I train an ELECTRA-Large model that performs comparably to RoBERTa (Liu et al., 2019e) and XLNet (Yang et al., 2019), despite having fewer parameters and using 1/4 of the compute for training. Training ELECTRA-Large further results in an even stronger model that outperforms ALBERT (Lan et al., 2020) on GLUE and improves all previous works on SQuAD 2.0. The Electric variant of my method slightly under-performs ELECTRA on GLUE and SQuAD. However, Electric is particularly useful in its ability to efficiently produce pseudo-likelihood scores (Salazar et al., 2020) for text: It is better at re-ranking the outputs of a speech recognition system than GPT-2 (Radford et al., 2019) because it is bidirectional and is much faster at re-ranking than BERT because it scores all input tokens simultaneously rather than having to be run multiple times with different tokens masked out. Taken together, my results indicate that the discriminative task of distinguishing real data from challenging negative samples is more compute-efficient and parameter-efficient than existing generative approaches for language representation learning.[2]

5.1 Method

This section details the replaced token detection pre-training task. First, I derive replaced token detection from the cloze task. I call the resulting pre-trained models "Electric" because of their connection to energy-based models. A few alterations to Electric results in ELECTRA ("Encoder Learned Efficiently by Classifying Token Replacements Accurately"), a more effective model that is the primary focus of this chapter.

[1]It has 1/20th the parameters and requires 1/135th the pre-training compute of BERT-Large.

[2]Code and pre-trained weights for this work are available at `https://github.com/google-research/electra`.

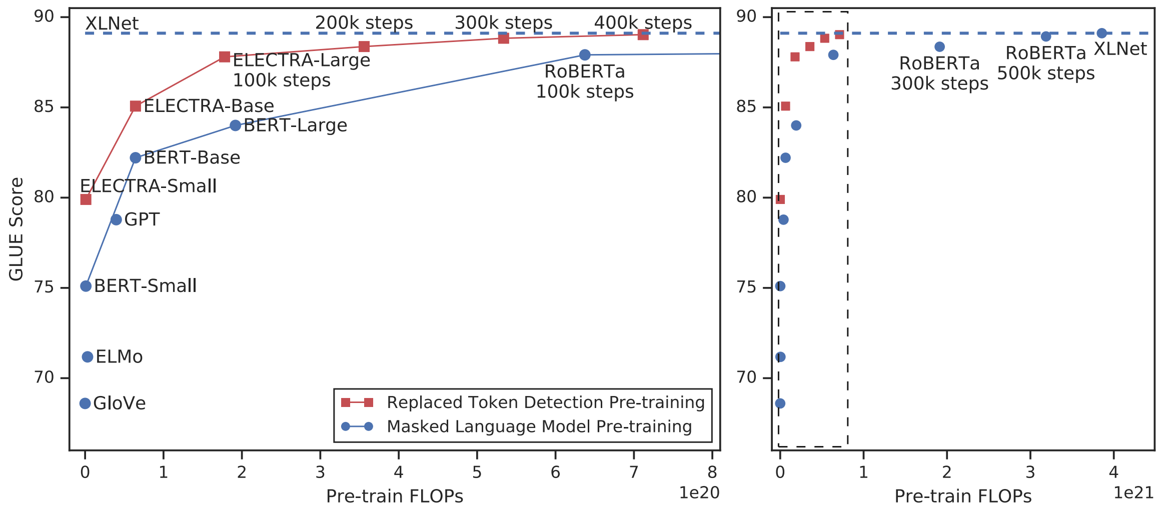


Figure 5.1: Replaced token detection pre-training consistently outperforms masked language model pre-training given the same compute budget. The left figure is a zoomed-in view of the dashed box.

5.1.1 Masked Language Models

Cloze models learn the probability $p_{\text{data}}(x_t|\boldsymbol{x}_{\backslash t})$ of a token x_t occurring in the surrounding context $\boldsymbol{x}_{\backslash t} = [x_1, ..., x_{t-1}, x_{t+1}, ..., x_n]$. BERT and related pre-training methods (Baevski et al., 2019; Liu et al., 2019e; Lan et al., 2020) implement this task through masking. Typically the context is represented as the input sequence with x_t replaced by a special `[MASK]` placeholder token. This masked sequence is encoded into vector representations by a Transformer network (Vaswani et al., 2017). Then the representation at position t is passed into a softmax layer to produce a distribution over tokens $p_\theta(x_t|\boldsymbol{x}_{\backslash t})$ for the position. See background section 2.5 for more details on the cloze task.

5.1.2 Electric: An Energy-Based Cloze Model

Electric also models $p_{\text{data}}(x_t|\boldsymbol{x}_{\backslash t})$, but does not use masking or a softmax layer. Electric first maps the unmasked input $\boldsymbol{x} = [x_1, ..., x_n]$ into contextualized vector representations $\boldsymbol{h}(\boldsymbol{x}) = [\boldsymbol{h}_1, ..., \boldsymbol{h}_n]$ using a Transformer network. The model assigns a given position t an

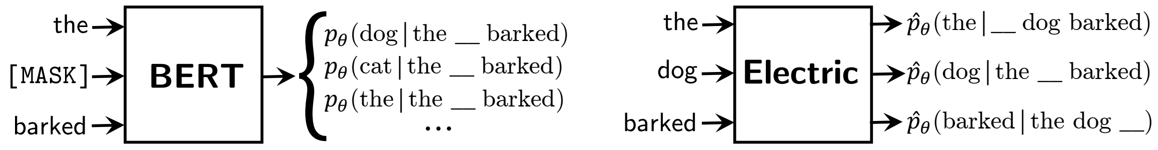

the
[MASK]
barked
BERT
p_θ(dog | the __ barked)
p_θ(cat | the __ barked)
p_θ(the | the __ barked)
...
the
dog
barked
Electric
$\hat{p}_\theta$(the | __ dog barked)
$\hat{p}_\theta$(dog | the __ barked)
$\hat{p}_\theta$(barked | the dog __)

Figure 5.2: Comparison of BERT and Electric. Both model the probability of a token given its surrounding context, but BERT produces a full output distribution over tokens only for masked positions while Electric produces un-normalized probabilities (but no full distribution) for all input tokens.

energy score

$$E(\boldsymbol{x})_t = \boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x})_t$$

using a learned weight vector $\boldsymbol{w}$. The energy function defines a distribution over the possible tokens at position t, which is obtained by exponentiating and normalizing E:

$$p_\theta(x_t|\boldsymbol{x}_{\backslash t}) = \exp\left(-E(\boldsymbol{x})_t\right)/Z(\boldsymbol{x}_{\backslash t}) = \frac{\exp\left(-E(\boldsymbol{x})_t\right)}{\sum_{x' \in \mathcal{V}} \exp\left(-E(\textsc{replace}(\boldsymbol{x}, t, x'))_t\right)}$$

where $\textsc{replace}(\boldsymbol{x}, t, x')$ denotes replacing the token at position t with x' and $\mathcal{V}$ is the vocabulary, in practice usually word pieces (Sennrich et al., 2016b). Unlike with BERT, which produces the probabilities for all possible tokens x' using a softmax layer, a candidate x' is passed in as *input* to the Transformer. As a result, computing p_θ is prohibitively expensive because the partition function $Z_\theta(\boldsymbol{x}_{\backslash t})$ requires running the Transformer $|\mathcal{V}|$ times.

5.1.3 NCE Loss

As computing the exact likelihood is intractable, training energy-based models such as Electric with standard maximum-likelihood estimation is not possible. Instead, I use (conditional) Noise-Contrastive Estimation (NCE) (Gutmann and Hyvärinen, 2010; Ma and Collins, 2018), which provides a way of efficiently training an un-normalized model that does not compute $Z_\theta(\boldsymbol{x}_{\backslash t})$. NCE learns the parameters of a model by training it to perform a binary classification task where samples from the data distribution have to be distinguished from samples generated by a noise distribution $q(x_t|\boldsymbol{x}_{\setminus t})$. I discuss the noise distribution in more detail in Section 5.1.5. First, consider the un-normalized output

$$\hat{p}_\theta(x_t|\boldsymbol{x}_{\setminus t}) = \exp\left(-E(\boldsymbol{x})_t\right)$$

The NCE binary classification task directly trains this un-normalized probability to match the data distribution, which removes the need to compute $Z_\theta(\boldsymbol{x}_{\setminus t})$. Operationally, NCE can be viewed as follows:

- A positive data point is a text sequence $\boldsymbol{x}$ from the data and position in the sequence t. In other words, a data token x_t in its context.
- A negative data point is the same except x_t, the token at position t, is replaced with a noise token $\hat{x}_t$ sampled from q. In other words, a context from the data with a fake token $\hat{x}_t$ replacing x_t.
- Define a binary classifier D (the ”discriminator”) that estimates the probability of a data point being positive as

$$D(\boldsymbol{x})_t = \frac{n \cdot \hat{p}_\theta(x_t|\boldsymbol{x}_{\setminus t})}{n \cdot \hat{p}_\theta(x_t|\boldsymbol{x}_{\setminus t}) + k \cdot q(x_t|\boldsymbol{x}_{\setminus t})}$$

- The binary classifier is trained to distinguish positive vs negative data points using standard maximum likelihood estimation, with k negatives sampled for every n positive data points.

Formally, the NCE loss is

$$\begin{aligned}\mathcal{L}(\theta) =& n \cdot \mathop{\mathbb{E}}_{\boldsymbol{x},t}\left[-\log \frac{n \cdot \hat{p}_\theta(x_t|\boldsymbol{x}_{\setminus t})}{n \cdot \hat{p}_\theta(x_t|\boldsymbol{x}_{\setminus t}) + k \cdot q(x_t|\boldsymbol{x}_{\setminus t})}\right] + \\ & k \cdot \mathop{\mathbb{E}}_{\substack{\boldsymbol{x},t \\ \hat{x}_t \sim q}}\left[-\log \frac{k \cdot q(\hat{x}_t|\boldsymbol{x}_{\setminus t})}{n \cdot \hat{p}_\theta(\hat{x}_t|\boldsymbol{x}_{\setminus t}) + k \cdot q(\hat{x}_t|\boldsymbol{x}_{\setminus t})}\right] \\ =& -\mathop{\mathbb{E}}_{\substack{\boldsymbol{x},t \\ \hat{x}_t \sim q}}\left[n \log D(\boldsymbol{x})_t + k \log(1 - D(\textsc{replace}(\boldsymbol{x}, t, \hat{x}_t))_t)\right]\end{aligned}$$

Where $\boldsymbol{x}$ is sampled from the data distribution, t is sampled uniformly at random from $[n]$ where n is the length of $\boldsymbol{x}$, and $\text{REPLACE}(\boldsymbol{x}, t, \hat{x}_t)$ denotes replacing x_t, the token at position t, with $\hat{x}_t$

This loss is minimized when $\hat{p}_\theta$ matches the data distribution p_{data}. To see why this is true, we can first compute the gradient of the loss with respect to θ. To be concise, I will use p_{data}, $\hat{p}_\theta$, and q to denote $p_{\text{data}}(\cdot|\boldsymbol{x}_{\backslash t})$, $\hat{p}_\theta(\cdot|\boldsymbol{x}_{\backslash t})$, and $q(\cdot|\boldsymbol{x}_{\backslash t})$ where $\cdot$ is either x_t or $\hat{x}_t$. The gradient is

$$\begin{aligned}
&\nabla_\theta \mathcal{L}(\theta) = -\nabla_\theta \left(n \mathop{\mathbb{E}}_{\boldsymbol{x},t} \left[\log \frac{n\hat{p}_\theta}{n\hat{p}_\theta + kq} \right] + k \mathop{\mathbb{E}}_{\boldsymbol{x},t,\hat{x}} \left[\log \frac{kq}{n\hat{p}_\theta + kq} \right] \right) = \\
&-\nabla_\theta \sum_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{|\boldsymbol{x}|} \frac{p_{\text{data}}(\boldsymbol{x})}{|\boldsymbol{x}|} \left(np_{\text{data}} \log \frac{n\hat{p}_\theta}{n\hat{p}_\theta + kq} + kq \log \frac{kq}{n\hat{p}_\theta + kq} \right) = \\
&-\sum_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{|\boldsymbol{x}|} \frac{p_{\text{data}}(\boldsymbol{x})}{|\boldsymbol{x}|} (\nabla_\theta \hat{p}_\theta) \left(np_{\text{data}} \frac{n\hat{p}_\theta + kq}{n\hat{p}_\theta} \frac{n\hat{p}_\theta + kq - n\hat{p}_\theta}{(n\hat{p}_\theta + kq)^2} - kq \frac{n\hat{p}_\theta + kq}{kq} \frac{kq}{(n\hat{p}_\theta + kq)^2} \right) = \\
&-\sum_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{|\boldsymbol{x}|} \frac{p_{\text{data}}(\boldsymbol{x})}{|\boldsymbol{x}|} (\nabla_\theta \hat{p}_\theta) \left(np_{\text{data}} \frac{kq}{n\hat{p}_\theta(n\hat{p}_\theta + kq)} - \frac{kq}{(n\hat{p}_\theta + kq)} \right)
\end{aligned}$$

where $\mathcal{X}$ consists of all possible input sequences. The second expression comes from expanding the expectation, the third from computing the derivative, and the last from simplifying. We can find when this loss is minimized by evaluating when the gradient is equal to 0.

$$\begin{aligned}
&np_{\text{data}} \frac{kq}{n\hat{p}_\theta(n\hat{p}_\theta + kq)} - \frac{kq}{(n\hat{p}_\theta + kq)} = 0 \\
&\frac{np_{\text{data}}}{n\hat{p}_\theta} - 1 = 0 \\
&\hat{p}_\theta = p_{\text{data}}
\end{aligned}$$

Since $\hat{p}_\theta$ is learning to match the normalized distribution p_{data}, the model learns to be self-normalized such that $Z_\theta(\boldsymbol{x}_{\backslash t}) = 1$. If the model is self-normalized, we have $\hat{p}_\theta(x_t|\boldsymbol{x}_{\backslash t}) = p_\theta(x_t|\boldsymbol{x}_{\backslash t})$, which means $\hat{p}_\theta$ becomes a good estimate of p_θ as the model learns.

Algorithm 1 Naive NCE loss estimation

```
Given: Input sequence $\boldsymbol{x}$ of length $n$, number of negative samples to take $k$, noise distribution $q$, and model $\hat{p}_\theta$.
1. Define the binary classifier $D(\boldsymbol{x})_t = \frac{n \cdot \hat{p}_\theta(x_t|\boldsymbol{x}_{\setminus t})}{n \cdot \hat{p}_\theta(x_t|\boldsymbol{x}_{\setminus t}) + k \cdot q(x_t|\boldsymbol{x}_{\setminus t})}$
2. Initialize the loss as $\sum_{t=1}^{n} -\log D(\boldsymbol{x})_t$.
3. Create $k$ negative samples. For each one, sample a position in $\boldsymbol{x}$ to replace according to $t \sim$ unif$\{1, n\}$ and then sample a noise token according to $\hat{x}_t \sim q(x_t|\boldsymbol{x}_{\setminus t})$.
4. For each negative sample $(t, \hat{x}_t)$, add $-\log\left(1 - D(\textsc{replace}(\boldsymbol{x}, t, \hat{x}_t))_t\right)$ to the loss.
Runtime: $k+1$ Transformer passes (once for the $n$ positive samples, one each for the $k$ negative samples).
```

Algorithm 2 Efficient approximate NCE loss estimation (Replaced Token Detection)

```
Given: Input sequence $\boldsymbol{x}$ of length $n$, number of negative samples to take $k$, noise distribution $q$, and model $\hat{p}_\theta$.
1. Define the binary classifier $D(\boldsymbol{x})_t = \frac{(n-k) \cdot \hat{p}_\theta(x_t|\boldsymbol{x}_{\setminus t})}{(n-k) \cdot \hat{p}_\theta(x_t|\boldsymbol{x}_{\setminus t}) + k \cdot q(x_t|\boldsymbol{x}_{\setminus t})}$
2. Pick $k$ unique random positions in the input sequence $\mathcal{R} = \{r_1, ..., r_k\}$ where each $r_i$ is $1 \leq r_i \leq n$ to replace with noise tokens.
3. Simultaneously replace the tokens at all $k$ random positions with negative samples $\hat{X}$ from $q$: $\hat{x}_i \sim q(x_i|\boldsymbol{x}_{\setminus i})$ for $i \in R$, $\boldsymbol{x}^{\text{noised}} = \textsc{replace}(\boldsymbol{x}, \mathcal{R}, \hat{X})$.
4. For each position $t = 1$ to $n$: add to the loss
    $-\log D(\boldsymbol{x}^{\text{noised}})_t$              if $t \in R$
    $-\log\left(1 - D(\boldsymbol{x}^{\text{noised}})_t\right)$   otherwise
Runtime: One Transformer pass for $n-k$ positive samples and $k$ negative samples.
```

5.1.4 Training Algorithm

To minimize the loss, the expectations can be approximated by sampling. Algorithm 1 shows how the loss could be computed for a single sequence $\boldsymbol{x}$. Taking the gradient of this estimated loss produces an unbiased estimate of $\nabla_\theta \mathcal{L}(\theta)$. However, this algorithm is computationally expensive to run, since it requires $k+1$ forward passes through the Transformer to compute the $\hat{p}_\theta$s (once for the positive samples and once for each negative sample). I propose a much more efficient approach that replaces k input tokens with noise samples *simultaneously* shown in Algorithms 2. This algorithm requires just one pass through the Transformer for k noise samples and $n-k$ data samples. However, this procedure only truly minimizes $\mathcal{L}$ if (1) $D(\boldsymbol{x}^{\text{noised}})_t = D(\boldsymbol{x})_t$ for positive samples (i.e., replacing some other tokens with noise tokens does not change the discriminator score for non-replaced tokens) and (2) $D(\boldsymbol{x}^{\text{noised}})_t = D(\text{REPLACE}(\boldsymbol{x}, t, \hat{x}_t))_t$ for negative samples (i.e., replacing some other tokens with noise tokens does not change the discriminator score for a single noise token). Equivalently, this procedure assumes that $\hat{p}_\theta(x_t|\boldsymbol{x}_{\backslash t}) = \hat{p}_\theta(x_t|\boldsymbol{x}^{\text{noised}}_{\backslash t})$. To apply this efficiency trick I am making the assumption they are approximately equal, which I argue is reasonable because (1) I choose a small k of $\lceil 0.15n \rceil$, so there are few negative samples that may change D and (2) q is trained to be close to the data distribution (see below) so these negative samples are less likely to dramatically alter $\boldsymbol{x}$. This efficiency trick is analogous to BERT masking out multiple tokens per input sequence instead of just one.

5.1.5 Variants

Algorithm 2 gives rise to the replaced token detection task: k input tokens are replaced by the noise distribution and the binary classification task is to distinguish the original data tokens from these fakes. Here I discuss the choice of noise distribution q, as well as presenting ELECTRA, a version of Electric that performs slightly better in practice.

NCE vs. Negative Sampling. The NCE binary classifier (discriminator) presented above makes use of the noise distribution q when making predictions. It can be written as

$$D(\boldsymbol{x})_t = \frac{k \cdot q(x|\boldsymbol{x}_{\backslash t})}{n \cdot \exp(-E(\boldsymbol{x})_t) + k \cdot q(x|\boldsymbol{x}_{\backslash t})} = \sigma\left(E(\boldsymbol{x})_t + \log\left(\frac{k \cdot q(x|\boldsymbol{x}_{\backslash t})}{n}\right)\right)$$

where σ denotes the sigmoid function. A simpler binary classifier that does not need access to the noise distribution can be created by instead just adding a sigmoid layer on top of the Transformer:

$$D'(\boldsymbol{x})_t = \sigma\left(E(\boldsymbol{x})_t\right)$$

Not needing access to q is desirable because it allows for more effective choices of q (see below). With D', the training method becomes *negative sampling* instead of noise-contrastive estimation. Negative sampling has been widely effective in NLP, especially for learning word embeddings (Mikolov et al., 2013b). Negative sampling trains the model to learn when a token is more likely to come from the data distribution p_{data} or noise distribution q,

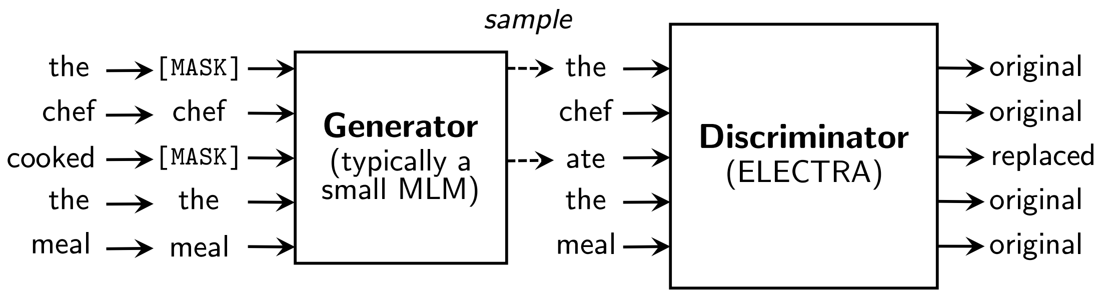


Figure 5.3: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but I usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, I train the generator with maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. After pre-training, I throw out the generator and only fine-tune the discriminator (the ELECTRA model) on downstream tasks.

On the other hand, NCE only results in the model learning p_{data} because q is passed into the model directly. ELECTRA uses negative sampling while Electric uses NCE.

Choice of Noise Distribution. The noise distribution q can be any function that produces a distribution over tokens given their context. However, some choices of q can cause the model to learn much faster than others. Intuitively, a simple noise distribution (e.g., uniform at random) provides a very easy task for the model where it won't learn effectively. Most replaced tokens will be obviously wrong, not requiring a deep understanding of the text to discern as fake. To create a challenging q, NCE commonly uses the output distribution of a model trained to match p_{data} (Gutmann and Hyvärinen, 2010; Wang and Ou, 2018).

I take this approach for most of the replaced token detection models I train: q comes from a separate "generator" neural network trained to match p_{data}. However, unlike Electric, these generators use an output softmax so they can efficiently produce a full distribution over tokens given the context. One choice of generator is simply a masked language model. Once a set of random tokens have been chosen to be replaced by noise samples, they are first replaced with `[MASK]`. The generator then predicts their original identities with an output softmax. Lastly, the `[MASK]` tokens are replaced by samples from the softmax distribution. ELECTRA uses this choice of generator. See Figure 5.3 for ELECTRA's training procedure.

A masked language model does not work as the generator for Electric because the NCE binary classifier needs access to q for every input token. However, a MLM only produces a q distribution for the k tokens that will be replaced. Instead, for Electric I use a two-tower cloze model as proposed by Baevski et al. (2019). The model runs two Transformers T_{LTR} and T_{RTL} over the input sequence. These Transformers apply causal masking so one processes the sequence left-to-right and the other operates right-to-left. The model's predictions come from a softmax layer applied to the concatenated states of the two Transformers:

$$\overrightarrow{\boldsymbol{h}} = T_{\text{LTR}}(\boldsymbol{x}), \quad \overleftarrow{\boldsymbol{h}} = T_{\text{RTL}}(\boldsymbol{x})$$
$$q(x_t|\boldsymbol{x}_{\setminus t}) = \text{softmax}(\boldsymbol{W}[\overrightarrow{\boldsymbol{h}}_{t-1}, \overleftarrow{\boldsymbol{h}}_{t+1}])_{x_t}$$

Note that the *previous* rather than current states of the Transformers are used to prevent the model from having direct access to the token it is learning to predict. A two-tower cloze model is more accurate than a language model because it uses context to both sides of each token. However, it is less accurate than a masked language model because it is not "deeply bidirectional"; the left and right contexts of a token only interact when their states are concatenated as opposed to being able to attend to each other. The inferior noise distribution is a disadvantage of using NCE (Electric) instead of negative sampling (ELECTRA). Typically the generator is much smaller than the main discriminator network performing the binary classification I further explore using simple generators such as uniformly picking fake tokens at random from the vocabulary and different sizes for the generator network in Section 5.3.2.

The generator is trained using standard maximum likelihood estimation over the data. It is trained simultaneously with the discriminator: after predicting the identities of the tokens selected to be replaced, its parameters are updated using gradient descent on the loss from those predictions. Simultaneous training proved to be more effective than a two-stage training process where first the generator is trained and then the discriminator (see Sec 5.3). With the discriminator learning to identify fakes produced by a generator network, the NCE training is reminiscent of a generative adversarial network (GAN; (Goodfellow et al., 2014)). However, there is a key difference: the generator is trained with maximum likelihood rather than being trained adversarially to fool the discriminator. Adversarially training the generator is challenging because it is impossible to backpropagate through the discrete operation of sampling replacements from the generator. Although I experimented with circumventing this issue by using reinforcement learning to train the generator (see Section 5.3.4), this performed worse than maximum-likelihood training. Furthermore, I do not supply the generator with a noise vector as input, as is typical with a GAN. After pre-training, I throw out the generator and fine-tune the discriminator on downstream tasks; the generator is only there to improve the training of the main model, not to be useful on its own.

Electric vs. ELECTRA. Electric uses noise-contrastive estimation for training and a two-tower cloze model as the generator, meaning it is directly training an energy-based model to perform cloze modeling. ELECTRA on the other hand uses a more straightforward sigmoid binary classifier, which means it can employ a more powerful masked language model as the generator network. An advantage of Electric is that it directly provides (un-normalized) probabilities $\hat{p}_\theta$ for tokens, making it useful for applications such as re-ranking the outputs of text generation systems. A disadvantage of Electric is that it has to use a less effective noise model. The differences between ELECTRA and Electric are summarized below:

Model	Noise Dist.	Binary Classifier	Training Method
Electric	Two-Tower Cloze Model	$\sigma\left(E(\boldsymbol{x})_t + \log\left(\frac{k \cdot q(x \mid \boldsymbol{x}_{\setminus t})}{n}\right)\right)$	NCE
ELECTRA	Masked LM	$\sigma(E(\boldsymbol{x})_t)$	Negative Sampling

5.2 Model Details and Experimental Setup

The next sections cover extensive experiments evaluating and analysing the replaced token detection task. Most experiments focus on ELECTRA because it generally performs better.

5.2.1 Data

I evaluate fine-tuning pre-trained models on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019) and Stanford Question Answering (SQuAD) dataset (Rajpurkar et al., 2016). GLUE contains a variety of tasks covering textual entailment (RTE and MNLI) question-answer entailment (QNLI), paraphrase (MRPC), question paraphrase (QQP), textual similarity (STS), sentiment (SST), and linguistic acceptability (CoLA). The evaluation metrics are Spearman correlation for STS, Matthews correlation for CoLA, and accuracy for the other GLUE tasks; I generally report the average score over all tasks. For SQuAD, I evaluate on versions 1.1, in which models select the span of text answering a question, and 2.0, in which some questions are unanswerable by the passage. I use the standard evaluation metrics of Exact-Match (EM) and F1 scores. See background section 2.6 for more details on the tasks. For most experiments, I pre-train on the same data as BERT, which consists of 3.3 Billion tokens from Wikipedia and BooksCorpus (Zhu et al., 2015). However, for my ELECTRA-Large model I pre-trained on the data used for XLNet (Yang et al., 2019), which extends the BERT dataset to 33B tokens by including data from ClueWeb (Callan et al., 2009), CommonCrawl, and Gigaword (Parker et al., 2011). All of the pre-training and evaluation is on English data, although I think it would be interesting to apply my methods to multilingual data in the future.

5.2.2 Counting FLOPs

With the goal of measuring efficiency, many of my results measure the compute used by the models. I chose to measure compute usage in terms of floating point operations (FLOPs) because it is a measure agnostic to the particular hardware, low-level optimizations, etc. However, it is worth noting that in some cases abstracting away hardware details is a drawback because hardware-centered optimizations can be key parts of a model's design, such as the speedup ALBERT (Lan et al., 2020) gets by tying weights and thus reducing communication overhead between TPU workers. I used TensorFlow's FLOP-counting capabilities[3] and checked the results with by-hand computation. I made the following assumptions:

- An "operation" is a mathematical operation, not a machine instruction. For example,

[3]See `https://www.tensorflow.org/api_docs/python/tf/profiler`

an `exp` is one op like an `add`, even though in practice the `exp` might be slower. I believe this assumption does not substantially change compute estimates because matrix-multiplies dominate the compute for most models. Similarly, I count matrix-vector multiplies (using an $m \times n$ matrix) as $2 * m * n$ FLOPs instead of $m * n$ as one might if considering fused multiply-add operations.

- The backwards pass takes the same number of FLOPs as the forward pass. This assumption is not exactly right (e.g., for softmax cross entropy loss the backward pass is faster), but importantly, the forward/backward pass FLOPs really are the same for matrix-multiplies, which is most of the compute anyway.

- I assume "dense" embedding lookups (i.e., multiplication by a one-hot vector). In practice, sparse embedding lookups are much slower than constant time; on some hardware accelerators dense operations are actually faster than sparse lookups.

5.2.3 Pre-Training Details

The following details apply to ELECTRA and Electric models as well as BERT baselines. I mostly use the same hyperparameters as BERT. I use dynamic token masking with the masked positions decided on-the-fly instead of during preprocessing. Also, I did not use the next sentence prediction objective proposed in the original BERT paper, as subsequent work has suggested it does not improve scores (Yang et al., 2019; Liu et al., 2019e). For my ELECTRA-Large model, I used a higher mask percent (25 instead of 15) because I noticed the generator was achieving high accuracy with 15% masking, resulting in very few replaced tokens. I searched for the best learning rate for the Base and Small models out of [1e-4, 2e-4, 3e-4, 5e-4] in early experiments. Otherwise I did no hyperparameter tuning beyond the experiments in Section 5.3. The full set of hyperparameters are listed in Table 5.1.

5.2.4 Fine-Tuning Details

For fine-tuning ELECTRA and Electric on GLUE, I add simple linear classifiers on top of ELECTRA. For SQuAD, I add the question-answering module from XLNet on top

Hyperparameter	**Small**	**Base**	**Large**
Number of layers	12	12	24
Hidden Size	256	768	1024
FFN inner hidden size	1024	3072	4096
Attention heads	4	12	16
Attention head size	64	64	64
Embedding Size	128	768	1024
Generator Size (multiplier for hidden-size, FFN-size, and num-attention-heads)	1/4	1/3	1/4
Negative sample percent	15	15	25
Learning Rate Decay	Linear	Linear	Linear
Warmup steps	10000	10000	10000
Learning Rate	5e-4	2e-4	2e-4
Adam ϵ	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999
Attention Dropout	0.1	0.1	0.1
Dropout	0.1	0.1	0.1
Weight Decay	0.01	0.01	0.01
Batch Size	128	256	2048
Train Steps (BERT/ELECTRA)	1.45M/1M	1M/766K	464K/400K

Table 5.1: Pre-train hyperparameters. I also train an ELECTRA-Large model for 1.75M steps (other hyperparameters are identical).

of ELECTRA/Electric, which is slightly more sophisticated than BERT's in that it jointly rather than independently predicts the start and end positions and has an "answerability" classifier added for SQuAD 2.0. Some of my evaluation datasets are small, which means accuracies of fine-tuned models can vary substantially depending on the random seed. I therefore report the median of 10 fine-tuning runs from the same pre-trained checkpoint for each result. Unless stated otherwise, results are on the dev set.

For Large-sized models, I used the hyperparameters from Clark et al. (2019b) for the most part. However, after noticing that RoBERTa (Liu et al., 2019e) uses more training epochs (up to 10 rather than 3) I searched for the best number of train epochs out of [10, 3] for each task. For SQuAD, I decreased the number of train epochs to 2 to be consistent

Hyperparameter	GLUE Value
Learning Rate	3e-4 for Small, 1e-4 for Base, 5e-5 for Large
Adam ϵ	1e-6
Adam β_1	0.9
Adam β_2	0.999
Layerwise LR decay	0.8 for Base/Small, 0.9 for Large
Learning rate decay	Linear
Warmup fraction	0.1
Attention Dropout	0.1
Dropout	0.1
Weight Decay	0
Batch Size	32
Train Epochs	10 for RTE and STS, 2 for SQuAD, 3 for other tasks

Table 5.2: Fine-tune hyperparameters

with BERT and RoBERTa. For Base-sized models I searched for a learning rate out of [3e-5, 5e-5, 1e-4, 1.5e-4] and the layer-wise learning-rate decay out of [0.9, 0.8, 0.7], but otherwise used the same hyperparameters as for Large models. I found the small models benefit from a larger learning rate and searched for the best one out of [1e-4, 2e-4, 3e-4, 5e-3]. With the exception of number of train epochs, I used the same hyperparameters for all tasks. In contrast, previous research on GLUE such as BERT, XLNet, and RoBERTa separately searched for the best hyperparameters for each task. I expect my results would improve slightly if I performed the same sort of additional hyperparameter search. The full set of hyperparameters is listed in Table 5.2.

Following BERT, I do not show results on the WNLI GLUE task for the dev set results, as it is difficult to beat even the majority classifier using a standard fine-tuning-as-classifier approach. For the GLUE test set results, I apply the standard tricks used by many of the GLUE leaderboard submissions including RoBERTa (Liu et al., 2019e), XLNet (Yang et al., 2019), and ALBERT (Lan et al., 2020). Specifically:

- For RTE and STS I use intermediate task training (Phang et al., 2018), starting from an ELECTRA checkpoint that has been fine-tuned on MNLI. For RTE, I found it helpful to combine this with a lower learning rate of 2e-5.

- For WNLI, I follow the trick described in Liu et al. (2019e) where I extract candidate antecedents for the pronoun using rules and train a model to score the correct antecedent highly. However, different from Liu et al. (2019e), the scoring function is not based on MLM probabilities. Instead, I fine-tune ELECTRA's discriminator so it assigns high scores to the tokens of the correct antecedent when the correct antecedent replaces the pronoun. For example, if the Winograd schema is "the trophy could not fit in the suitcase because it was too big," I train the discriminator so it gives a high score to "trophy" in "the trophy could not fit in the suitcase because the trophy was too big" but a low score to "suitcase" in "the trophy could not fit in the suitcase because the suitcase was too big."

- For each task I ensemble the best 10 of 30 models fine-tuned with different random seeds but initialized from the same pre-trained checkpoint.

While these tricks do improve scores, they make having clear scientific comparisons more difficult because they require extra work to implement, require lots of compute, and make results less apples-to-apples because different papers implement the tricks differently. For my SQuAD 2.0 test set submission, I fine-tuned 20 models from the same pre-trained checkpoint and submitted the one with the best dev set score.

5.3 Model Extensions

I attempt to improve replaced token detection by proposing and evaluating several extensions to the model. Unless stated otherwise, these experiments train an ELECTRA model with the same network size and training data as BERT-Base.

5.3.1 Weight Sharing

One possibility for improving the efficiency of the pre-training by sharing weights between the generator and discriminator. If the generator and discriminator are the same size, all of the Transformer weights can be tied. However, I found it to be more efficient to have a small generator, in which case I only share the embeddings (both the token and positional embeddings) of the generator and discriminator. In this case I use embeddings the size of the discriminator's hidden states; I added linear layers to the generator to project the embeddings into generator-hidden-sized representations. The "input" and "output" token embeddings of the generator are always tied as in BERT.

To have a direct comparison with all weights being tied, I compared the weight tying strategies when the generator is the same size as the discriminator. I train these models for 500k steps. GLUE scores are 83.6 for no weight tying, 84.3 for tying token embeddings, and 84.4 for tying all weights. I hypothesize that ELECTRA benefits from tied token embeddings because masked language modeling is particularly effective at learning these representations: while the discriminator only updates tokens that are present in the input or are sampled by the generator, the generator's softmax over the vocabulary densely updates all token embeddings. On the other hand, tying all encoder weights caused little improvement while incurring the significant disadvantage of requiring the generator and discriminator to be the same size. Based on these findings, I use tied embeddings for further experiments in this paper.

5.3.2 Smaller Generators

If the generator and discriminator are the same size, training ELECTRA would take around twice as much compute per step as training a masked language model like BERT. I suggest using a smaller generator to reduce this factor. Specifically, I make models smaller by decreasing the layer sizes while keeping the other hyperparameters constant. I also explore using an extremely simple "unigram" generator that samples fake tokens according their frequency in the train corpus. GLUE scores for differently-sized generators and discriminators are shown in Figure 5.4. All models are trained for 500k steps, which puts the smaller generators at a disadvantage in terms of compute because they require less compute per training step. Nevertheless, I find that models work best with generators 1/4–1/2 the size of the discriminator. I speculate that having too strong of a generator may pose a too-challenging task for the discriminator, preventing it from learning as effectively. In particular, the discriminator may have to use many of its parameters modeling the generator rather than the actual data distribution. Further experiments in this paper use the best

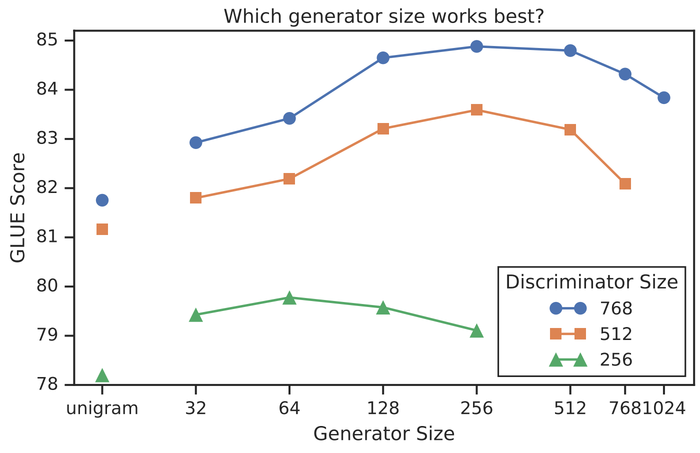


Figure 5.4: GLUE scores for different generator/discriminator sizes (number of hidden units). Interestingly, having a generator smaller than the discriminator improves results.

generator size found for the given discriminator size.

5.3.3 Two-Stage Training Algorithms

Lastly, I explore two other training algorithms for ELECTRA, although these did not end up improving results. First, I experiment with using the following two-stage training procedure instead of simultaneously training the generator and discriminator.:

1. Train only the generator for n steps.
2. Initialize the weights of the discriminator with the weights of the generator. Then train the discriminator for n steps, keeping the generator's weights frozen.

Note that the weight initialization in this procedure requires having the same size for the generator and discriminator. I found that without the weight initialization the discriminator would sometimes fail to learn at all beyond the majority class, perhaps because the generator started so far ahead of the discriminator. Joint training on the other hand naturally provides a curriculum for the discriminator where the generator starts off weak but gets better throughout training.

5.3.4 Adversarial Training Algorithm

I also explored training the generator adversarially as in a GAN, using reinforcement learning to accommodate the discrete operations of sampling from the generator. In particular I train the noise distribution (generator) q to maximize the binary classifier (discriminator) loss $\mathcal{L}$. As the generator is used just to train the discriminator rather than as a useful generative model of its own, this idea is really an instance of Adversarial Contrastive Estimation (Bose et al., 2018) rather than Generative Adversarial Training. It is not possible to adversarially train the generator by backpropagating through the discriminator (e.g., as in a GAN trained on images) due to the discrete sampling from the generator, so I use reinforcement learning instead.

The generator is different from most text generation models in that it is non-autoregressive: predictions are made independently. In other words, rather than taking a sequence of actions where each action generates a token, the generator takes a single giant action of generating all tokens simultaneously, where the probability for the action factorizes as the product of generator probabilities for each token. To deal with this enormous action space, I make the following simplifying assumption: that the discriminator's prediction $D(\boldsymbol{x}^{\text{noised}})_t$ depends only on the token x_t and the non-replaced tokens $\{x_i : i \notin \mathcal{R}\}$, i.e., it does not depend on other generated tokens $\{\hat{x}_i : i \in \mathcal{R} \wedge i \neq t\}$ where $\mathcal{R}$ is the set of indices of tokens that will be replaced. This isn't too bad of an assumption because a relatively small number of tokens are replaced, and it greatly simplifies credit assignment when using reinforcement learning. Notationally, I show this assumption by (in a slight abuse of notation) by writing $D(\hat{x}_t|\boldsymbol{x}^{\text{masked}})$ for the discriminator predicting whether the generated token $\hat{x}_t$ equals the original token x_t given the masked context $\boldsymbol{x}^{\text{masked}}$ (as opposed to the whole noised context $\boldsymbol{x}^{\text{noised}}$). A useful consequence of this assumption is that the discriminator score for non-replaced tokens ($D(x_t|\boldsymbol{x}^{\text{masked}})$ for $t \notin \mathcal{R}$) is independent of the generator samples because I am assuming it does not depend on any replaced token. Therefore these tokens can be ignored when training G to maximize $\mathcal{L}$. Learning the generator parameters seeks to find

$$\arg\max_q \mathcal{L} = \arg\max_q \mathop{\mathbb{E}}_{\boldsymbol{x},\mathcal{R},\hat{X}} \left(\sum_{t=1}^{n} -\mathbb{1}(t \notin \mathcal{R}) \log D(\boldsymbol{x}^{\text{noised}})_t - \right.$$
$$\left. \mathbb{1}(t \in \mathcal{R}) \log(1 - D(\boldsymbol{x}^{\text{noised}})_t) \right)$$

Using the simplifying assumption, I approximate the above by finding the argmax of

$$\mathop{\mathbb{E}}_{\boldsymbol{x},\mathcal{R},\hat{X}} \left(\sum_{t\in\mathcal{R}} -\mathbb{1}(t \notin \mathcal{R}) \log D(\hat{x}_t|\boldsymbol{x}^{\text{masked}}) - \mathbb{1}(t \in \mathcal{R}) \log(1 - D(\hat{x}_t|\boldsymbol{x}^{\text{masked}})) \right)$$
$$= \mathop{\mathbb{E}}_{\boldsymbol{x},\mathcal{R}} \sum_{t\in\mathcal{R}} \mathop{\mathbb{E}}_{\hat{x}_t\sim q} R(\hat{x}_t, \boldsymbol{x})$$
$$\text{where } R(\hat{x}_t, \boldsymbol{x}) = \begin{cases} -\log D(\hat{x}_t|\boldsymbol{x}^{\text{masked}}) & \text{if } t \notin \mathcal{R} \\ -\log(1 - D(\hat{x}_t|\boldsymbol{x}^{\text{masked}})) & \text{if } t \in \mathcal{R} \end{cases}$$

In short, the simplifying assumption allows us to decompose the loss over the individual generated fake tokens, changing the expectation over all replacements $\hat{X}$ to an expectation over individual replacements $\hat{x}_t$. I cannot directly find $\arg\max_q$ using gradient ascent because it is impossible to backpropagate through discrete sampling of $\hat{x}$. Instead, I use policy gradient reinforcement learning (Williams, 1992). In particular, I use the REINFORCE gradient

$$\nabla_{\theta_q} \mathcal{L} \approx \mathop{\mathbb{E}}_{\boldsymbol{x},\mathcal{R}} \sum_{t\in\mathcal{R}} \mathop{\mathbb{E}}_{\hat{x}_t\sim q} \nabla_{\theta_q} \log q(\hat{x}_t|\boldsymbol{x}^{\text{masked}})[R(\hat{x}_t, \boldsymbol{x}) - b(\boldsymbol{x}^{\text{masked}}, t)]$$

where b is a learned baseline implemented as $b(\boldsymbol{x}^{\text{masked}}, t) = -\log \text{sigmoid}(w^T h_q(\boldsymbol{x}^{\text{masked}})_t)$ and $h_q(\boldsymbol{x}^{\text{masked}})$ are the outputs of the generator's Transformer encoder. The baseline is trained with cross-entropy loss to match the reward for the corresponding position. I approximate the expectations with a single sample and learn θ_q with gradient ascent. Despite receiving no explicit feedback about which generated tokens are correct, I found the adversarial training resulted in a fairly accurate generator (for a 256-hidden-size generator, the adversarially trained one achieves 58% accuracy at masked language modeling while the

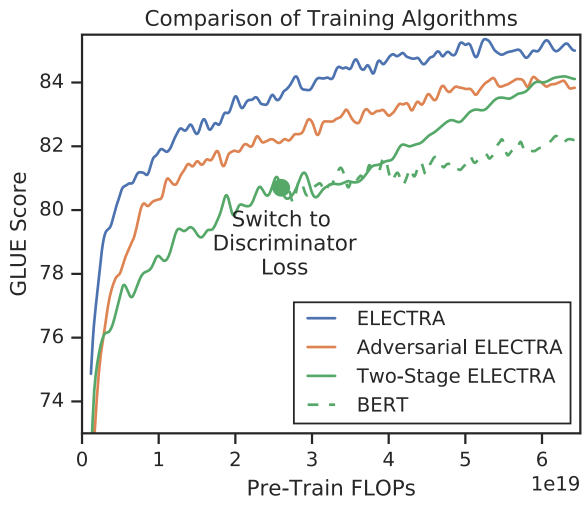


Figure 5.5: Comparison of different training algorithms. As my focus is on efficiency, the x-axis shows FLOPs rather than train steps (e.g., ELECTRA is trained for fewer steps than BERT because it includes the generator).

same sized MLE generator gets 65%).

5.3.5 Comparison of Training Algorithms

Results for the two alternative training algorithms are shown in Figure 5.5. During two-stage training, downstream task performance notably improves after the switch from the generative to the discriminative objective, but does not end up outscoring joint training. Although still outperforming BERT, I found adversarial training to underperform maximum-likelihood training. Further analysis suggests the gap is caused by two problems with adversarial training. First, the adversarial generator is simply worse at masked language modeling; it achieves 58% accuracy at masked language modeling compared to 65% accuracy for an MLE-trained one. I believe the worse accuracy is mainly due to the poor sample efficiency of reinforcement learning when working in the large action space of generating text. Secondly, the adversarially trained generator produces a low-entropy output distribution where most of the probability mass is on a single token, which means there is not much diversity in the generator samples. Both of these problems have been observed in GANs for text in prior work (Caccia et al., 2020).

5.4 Main Experiments

Having improved the model with empirically-validated extensions, I now turn to the main experiments evaluating the performance of ELECTRA and Electric. These cover their performance when fine-tuned on downstream tasks, as well as some analysis investigating why the approach is effective.

5.4.1 Fine-tuning Small ELECTRA Models

As a goal of this work is to improve the efficiency of pre-training, I develop a small ELECTRA model that can be quickly trained on a single GPU. Starting with the BERT-Base hyperparameters, I shortened the sequence length (from 512 to 128), reduced the batch size (from 256 to 128), reduced the model's hidden dimension size (from 768 to 256), and used smaller token embeddings (from 768 to 128). To provide a fair comparison, I also train a BERT-Small model using the same hyperparameters. I train BERT-Small for 1.5M steps, so it uses the same training FLOPs as ELECTRA-Small, which was trained for 1M steps.[4] In addition to BERT, I compare against two less resource-intensive pre-training methods based on language modeling: ELMo (Peters et al., 2018) and GPT (Radford et al., 2018).[5] I also show results for a base-sized ELECTRA model comparable to BERT-Base.

ELECTRA-Small performs remarkably well given its size, achieving a higher GLUE score than other methods using substantially more compute and parameters. For example, it scores 5 points higher than a comparable BERT-Small model and even outperforms the much larger GPT model. ELECTRA-Small is trained mostly to convergence, with models trained for even less time (as little as 6 hours) still achieving reasonable performance. While small models distilled from larger pre-trained Transformers can also achieve good GLUE scores (Sun et al., 2020; Jiao et al., 2020), these models require first expending substantial compute to pre-train the larger teacher model. The results also demonstrate the strength of ELECTRA at a moderate size; my base-sized ELECTRA model substantially outperforms BERT-Base and even outperforms BERT-Large (which gets 84.0 GLUE score). I hope ELECTRA's ability to achieve strong results with relatively little compute will broaden the

[4]ELECTRA requires more FLOPs per step because it consists of the generator as well as the discriminator.
[5]GPT is similar in size to BERT-Base, but is trained for fewer steps.

Model	Train / Infer FLOPs	Speedup	Params	Train Time	GLUE Score
ELMo	3.3e18 / 2.6e10	19x / 1.2x	96M	14d on 3 GTX 1080	71.2
GPT	4.0e19 / 3.0e10	1.6x / 0.97x	117M	25d on 8 P6000	78.8
BERT-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100	75.1
BERT-Base	6.4e19 / 2.9e10	1x / 1x	110M	4d on 16 TPUv3s	82.2
ELECTRA-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100	79.9
50% trained	7.1e17 / 3.7e9	90x / 8x	14M	2d on 1 V100	79.0
25% trained	3.6e17 / 3.7e9	181x / 8x	14M	1d on 1 V100	77.7
12.5% trained	1.8e17 / 3.7e9	361x / 8x	14M	12h on 1 V100	76.0
6.25% trained	8.9e16 / 3.7e9	722x / 8x	14M	6h on 1 V100	74.1
ELECTRA-Base	6.4e19 / 2.9e10	1x / 1x	110M	4d on 16 TPUv3s	85.1

Table 5.3: Comparison of small models on the GLUE dev set. BERT-Small/Base are my implementation and use the same hyperparameters as ELECTRA-Small/Base. Infer FLOPs assumes single length-128 input. Training times should be taken with a grain of salt as they are for different hardware and with sometimes un-optimized code. ELECTRA performs well even when trained on a single GPU, scoring 5 GLUE points higher than a comparable BERT model and even outscoring the much larger GPT model.

accessibility of developing and applying pre-trained models in NLP.

5.4.2 Fine-Tuning Large ELECTRA Models

I train big ELECTRA models to measure the effectiveness of the replaced token detection pre-training task at the large scale of existing state-of-the-art pre-trained Transformers. My ELECTRA-Large models are the same size as BERT-Large but are trained for much longer. In particular, I train a model for 400k steps (ELECTRA-400K; roughly 1/4 the pre-training compute of RoBERTa) and one for 1.75M steps (ELECTRA-1.75M; similar compute to RoBERTa). I use a batch size 2048 and the XLNet pre-training data. I note that although the XLNet data is similar to the data used to train RoBERTa, the comparison is not entirely direct. As a baseline, I trained my own BERT-Large model using the same hyperparameters and training time as ELECTRA-400K.

Results on the GLUE dev set are shown in Table 5.4. ELECTRA-400K performs comparably to RoBERTa and XLNet. However, it took less than 1/4 of the compute to train

Model	**Train FLOPs**	**CoLA**	**SST**	**MRPC**	**STS**	**QQP**	**MNLI**	**QNLI**	**RTE**	**Avg.**
BERT	1.9e20 (0.27x)	60.6	93.2	88.0	90.0	91.3	86.6	92.3	70.4	84.0
RoBERTa-100K	6.4e20 (0.90x)	66.1	95.6	**91.4**	92.2	92.0	89.3	94.0	82.7	87.9
RoBERTa-500K	3.2e21 (4.5x)	68.0	96.4	90.9	92.1	92.2	90.2	94.7	86.6	88.9
XLNet	3.9e21 (5.4x)	69.0	**97.0**	90.8	92.2	92.3	90.8	94.9	85.9	89.1
BERT (ours)	7.1e20 (1x)	67.0	95.9	89.1	91.2	91.5	89.6	93.5	79.5	87.2
ELECTRA-400K	7.1e20 (1x)	**69.3**	96.0	90.6	92.1	**92.4**	90.5	94.5	86.8	89.0
ELECTRA-1.75M	3.1e21 (4.4x)	69.1	96.9	90.8	**92.6**	**92.4**	**90.9**	**95.0**	**88.0**	**89.5**

Table 5.4: Comparison of large models on the GLUE dev set. ELECTRA and RoBERTa are shown for different numbers of pre-training steps, indicated by the numbers after the dashes. ELECTRA performs comparably to XLNet and RoBERTa when using less than 1/4 of their pre-training compute and outperforms them when given a similar amount of pre-training compute. BERT dev results are from Clark et al. (2019b). All models are comparable in terms of number of parameters.

ELECTRA-400K as it did to train RoBERTa and XLNet, demonstrating that ELECTRA's sample-efficiency gains hold at large scale. Training ELECTRA for longer (ELECTRA-1.75M) results in a model that outscores them on most GLUE tasks while still requiring less pre-training compute. Surprisingly, my baseline BERT model scores notably worse than RoBERTa-100K, suggesting my models may benefit from more hyperparameter tuning or using the RoBERTa training data. ELECTRA's gains hold on the GLUE test set (see Table 5.5), although these comparisons are less apples-to-apples due to the additional tricks employed by the models.

Results on SQuAD are shown in Table 5.6. Consistent, with the GLUE results, ELECTRA scores better than masked-language-modeling-based methods given the same compute resources. For example, ELECTRA-400K outperforms RoBERTa-100k and my BERT baseline, which use similar amounts of pre-training compute. ELECTRA-400K also performs comparably to RoBERTa-500K despite using less than 1/4th of the compute. Unsurprisingly, training ELECTRA longer improves results further: ELECTRA-1.75M scores higher than previous models on the SQuAD 2.0 benchmark. ELECTRA-Base also yields

Model	Train FLOPs	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	WNLI	Avg.*
BERT	1.9e20 (0.06x)	60.5	94.9	85.4	86.5	89.3	86.7	92.7	70.1	65.1	79.8
RoBERTa	3.2e21 (1.02x)	67.8	96.7	89.8	91.9	90.2	90.8	95.4	88.2	89.0	88.1
ALBERT	3.1e22 (10x)	69.1	**97.1**	**91.2**	92.0	90.5	**91.3**	–	89.2	91.8	89.0
XLNet	3.9e21 (1.26x)	70.2	**97.1**	90.5	**92.6**	90.4	90.9	–	88.5	**92.5**	89.1
ELECTRA	3.1e21 (1x)	**71.7**	**97.1**	90.7	92.5	**90.8**	**91.3**	**95.8**	**89.8**	**92.5**	**89.5**

Table 5.5: GLUE test-set results for large models. Models in this table incorporate additional tricks such as ensembling to improve scores. Some models do not have QNLI scores because they treat QNLI as a ranking task, which has been disallowed by the GLUE benchmark. To compare against these models, I report the average score excluding QNLI (Avg.*). "ELECTRA" and "RoBERTa" refer to the fully-trained ELECTRA-1.75M and RoBERTa-500K models.

Model	Train FLOPs	Params	SQuAD 1.1 dev		SQuAD 2.0 dev		SQuAD 2.0 test	
			EM	F1	EM	F1	EM	F1
BERT-Base	6.4e19 (0.09x)	110M	80.8	88.5	–	–	–	–
BERT	1.9e20 (0.27x)	335M	84.1	90.9	79.0	81.8	80.0	83.0
SpanBERT	7.1e20 (1x)	335M	88.8	94.6	85.7	88.7	85.7	88.7
XLNet-Base	6.6e19 (0.09x)	117M	81.3	–	78.5	–	–	–
XLNet	3.9e21 (5.4x)	360M	**89.7**	**95.1**	87.9	**90.6**	87.9	90.7
RoBERTa-100K	6.4e20 (0.90x)	356M	–	94.0	–	87.7	–	–
RoBERTa-500K	3.2e21 (4.5x)	356M	88.9	94.6	86.5	89.4	86.8	89.8
ALBERT	3.1e22 (44x)	235M	89.3	94.8	87.4	90.2	88.1	90.9
BERT (ours)	7.1e20 (1x)	335M	88.0	93.7	84.7	87.5	–	–
ELECTRA-Base	6.4e19 (0.09x)	110M	84.5	90.8	80.5	83.3	–	–
ELECTRA-400K	7.1e20 (1x)	335M	88.7	94.2	86.9	89.6	–	–
ELECTRA-1.75M	3.1e21 (4.4x)	335M	**89.7**	94.9	**88.0**	**90.6**	**88.7**	**91.4**

Table 5.6: Results on SQuAD for non-ensemble models.

strong results, scoring substantially better than BERT-Base and XLNet-Base, and even surpassing BERT-Large according to most metrics. ELECTRA generally performs better at SQuAD 2.0 than 1.1. Perhaps replaced token detection, in which the model distinguishes real tokens from plausible fakes, is particularly transferable to the answerability classification of SQuAD 2.0, in which the model must distinguish answerable questions from fake unanswerable questions.

5.4.3 Efficiency Analysis

I have suggested that posing the training objective over a small subset of tokens makes masked language modeling inefficient. However, it isn't entirely obvious that this is the case. After all, the model still receives a large number of input tokens even though it predicts only a small number of masked tokens. To better understand where the gains from ELECTRA are coming from, I compare a series of other pre-training objectives that are designed to be a set of "stepping stones" between BERT and ELECTRA.

- **ELECTRA 15%**: This model is identical to ELECTRA except the discriminator loss only comes from a random 15% of the tokens in the input sequence.
- **Replace MLM**: This objective is the same as masked language modeling except instead of replacing masked-out tokens with `[MASK]`, they are replaced with tokens from a generator model. This objective tests to what extent ELECTRA's gains come from solving the discrepancy of exposing the model to `[MASK]` tokens during pre-training but not fine-tuning.
- **All-Tokens MLM**: Like in Replace MLM, masked tokens are replaced with generator samples. Furthermore, the model predicts the identity of all tokens in the input, not just ones that were masked out. I found it improved results to train this model with an explicit copy mechanism that outputs a copy probability D for each token using a sigmoid layer. The model's output distribution puts D weight on the input token plus $1 - D$ times the output of the MLM softmax. This model is essentially a combination of BERT and ELECTRA. Note that without generator replacements, the model would trivially learn to make predictions from the vocabulary for `[MASK]` tokens and copy the input for other ones.

Results are shown in Table 5.7. First, I find that ELECTRA is greatly benefiting from having a loss defined over all input tokens rather than just a subset: ELECTRA 15% performs much worse than ELECTRA. Secondly, I find that BERT performance is being slightly harmed from the pre-train fine-tune mismatch from `[MASK]` tokens, as Replace MLM

Model	ELECTRA	All-Tokens MLM	Replace MLM	ELECTRA 15%	BERT
GLUE score	85.0	84.3	82.4	82.4	82.2

Table 5.7: Compute-efficiency experiments (see text for details).

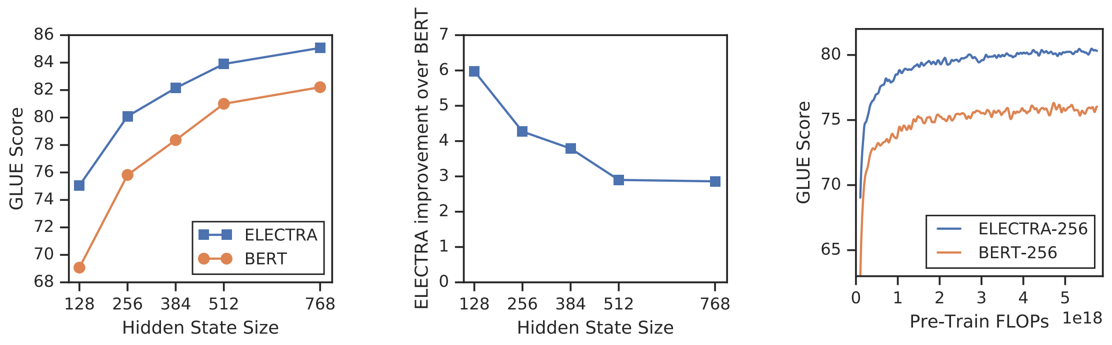


Figure 5.6: Left and Center: Comparison of BERT and ELECTRA performance for different model sizes. Right: A small ELECTRA model converges to higher downstream accuracy than BERT, showing the improvement comes from more than just faster training.

slightly outperforms BERT. BERT (including my implementation) already includes a trick to help with the pre-train/fine-tune discrepancy: masked tokens are replaced with a random token 10% of the time and are kept the same 10% of the time. However, my results suggest these simple heuristics are insufficient to fully solve the issue. Lastly, I find that All-Tokens MLM, the generative model that makes predictions over all tokens instead of a subset, closes most of the gap between BERT and ELECTRA. In total, these results suggest a large amount of ELECTRA's improvement can be attributed to learning from all tokens and a smaller amount can be attributed to alleviating the pre-train fine-tune mismatch.

The improvement of ELECTRA over All-Tokens MLM suggests that the ELECTRA's gains come from more than just faster training. I study this further by comparing BERT to ELECTRA for various model sizes (see Figure 5.6, left). I find that the gains from ELECTRA grow larger as the models get smaller. The small models are trained fully to convergence (see Figure 5.6, right), showing that ELECTRA achieves higher downstream accuracy than BERT when fully trained. I speculate that ELECTRA is more parameter-efficient than BERT because it does not have to model the full distribution of possible

Model	**CoLA**	**SST**	**MRPC**	**STS**	**QQP**	**MNLI**	**QNLI**	**RTE**	**SQuAD 1.0**	**SQuAD 2.0**
	MCC	Acc	Acc	Spear	Acc	Acc	Acc	Acc	EM	EM
Dev set results										
BERT	58.4	92.8	86.0	87.8	90.8	84.5	88.6	68.5	80.8	73.7
XLNet	–	93.4	–	–	–	85.8	–	–	–	78.5
ELECTRA	65.8	92.4	87.9	89.1	90.9	86.2	92.4	76.3	84.5	80.5
Electric	61.8	91.9	88.0	89.4	90.6	85.7	92.1	73.4	84.5	80.1
Test set results										
BERT	52.1	93.5	84.8	85.8	89.2	84.6	90.5	66.4	–	–
ELECTRA	59.7	93.4	86.7	87.7	89.1	85.8	92.7	73.1	–	–
Electric	61.5	93.2	85.4	86.9	89.2	85.2	91.8	67.3	–	–

Table 5.8: GLUE scores of Electric on downstream tasks. To provide direct comparisons, I only show base-sized models pre-trained on WikiBooks and fine-tuned with standard single-task training.

tokens at each position, but I believe more analysis is needed to completely explain ELECTRA's parameter efficiency.

5.4.4 Fine-Tuning Electric

In Table 5.8 I show results for fine-tuning Electric rather than ELECTRA on downstream tasks (both SQuAD and GLUE). Electric scores better than BERT, showing the energy-based formulation improves cloze model pre-training. However, Electric scores slightly lower than ELECTRA. One possible explanation is that Electric's noise distribution is worse because a two-tower cloze model is less expressive than a masked LM. I tested this hypothesis by training ELECTRA with the same two-tower noise model as Electric. Performance did indeed go down, but it only explained about half the gap. The surprising drop in performance suggests that learning the difference between the data and generations from a low-capacity model leads to better representations than only learning the data distribution, but I believe further research is needed to fully understand the discrepancy.

5.4.5 Pseudo-Log-Likelihood Scoring

An advantage of Electric over BERT is that it can efficiently produce pseudo-log-likelihood (PLL) scores for text (Wang and Cho, 2019). PLLs for Electric are

$$\text{PLL}(\boldsymbol{x}) = \sum_{t=1}^{n} \log(p_\theta(x_t|\boldsymbol{x}_{\backslash t})) \approx \sum_{t=1}^{n} \log(\hat{p}_\theta(x_t|\boldsymbol{x}_{\backslash t})) = \sum_{t=1}^{n} -E(\boldsymbol{x})_t$$

PLLs can be used to re-rank the outputs of an machine translation or speech recognition system. While historically log-likelihoods from language models have been used for such re-ranking, Shin et al. (2019) demonstrate that PLLs from masked language models perform better. However, computing PLLs from a masked language model requires n passes of the Transformer: once with each token masked out. Salazar et al. (2020) suggest distilling BERT into a model that uses no masking to avoid this cost, but this model considerably under-performed regular LMs in their experiments. Electric can produce PLLs for all input tokens in a single pass like a LM while being bidirectional like a masked LM.

It is also possible, although more cumbersome, to get pseudo-log-likelihood scores from ELECTRA by combining the discriminator and generator probabilities. Recall that Electric's loss is minimized when $\hat{p}_\theta = p_{\text{data}}$ so for an optimal discriminator

$$D(\boldsymbol{x})_t = \frac{n \cdot p_{\text{data}}(x_t|\boldsymbol{x}_{\backslash t})}{n \cdot p_{\text{data}}(x_t|\boldsymbol{x}_{\backslash t}) + k \cdot q(x_t|\boldsymbol{x}_{\backslash t})} \implies$$
$$p_{\text{data}}(x_t|\boldsymbol{x}_{\backslash t}) = \frac{k \cdot q(x_t|\boldsymbol{x}_{\backslash t}) D(\boldsymbol{x})_t}{n \cdot (1 - D(\boldsymbol{x})_t)}$$

Although ELECTRA's discriminator has a different form than Electric's, it is learning precisely the same task, so this expression will hold for it as well. Consequently, an estimate for ELECTRA modeling $p(x_t|\boldsymbol{x}_{\backslash t})$ is $\frac{k \cdot q(x_t|\boldsymbol{x}_{\backslash t}) D(\boldsymbol{x})_t}{n \cdot (1 - D(\boldsymbol{x})_t)}$; as the discriminator gets better this expression will get closer to p_{data}. However, ELECTRA cannot produce these scores efficiently because the generator producing q is a masked language model, which cannot score all tokens in parallel. As an alternative, I train ELECTRA-TT, a version of ELECTRA with the same two-tower generator as Electric and use it to produce PLLs.

I use the PLLs from Electric and ELECTRA-TT for re-ranking the 100-best hypotheses of a 5-layer BLSTMP model from ESPnet (Watanabe et al., 2018) on the 960-hour

LibriSpeech corpus (Panayotov et al., 2015) following the same experimental setup and using the same n-best lists as Salazar et al. (2020). Given speech features $\boldsymbol{s}$ and speech recognition model f the re-ranked output is

$$\underset{\boldsymbol{x} \in \text{n-best}(f, \boldsymbol{s})}{\arg\max} \; f(\boldsymbol{x}|\boldsymbol{s}) + \lambda \text{PLL}(\boldsymbol{x})$$

where n-best$(f, \boldsymbol{s})$ consists of the top n (we use $n = 100$) predictions from the speech recognition model found with beam search, $f(\boldsymbol{x}|\boldsymbol{s})$ is the score the speech model assigns the candidate output sequence $\boldsymbol{x}$. I select the best λ on the dev set out of $[0.05, 0.1, ..., 0.95, 1.0]$, with different λs selected for the "clean" and "other" portions of the data.

I compare Electric and ELECTRA-TT against GPT-2 (Radford et al., 2019), and BERT (Devlin et al., 2019). I also compare against TwoTower, a two-tower cloze model similar to Electric's noise distribution, but of the same size as Electric. This systems is also bidirectional while only requiring a single Transformer pass like Electric, but only "shallowly" bidirectional, as it just concatenates forward and backward hidden states.

Results are shown in Table 5.9. Electric scores better than GPT-2 when trained on comparable data. While scoring worse than BERT, Electric is much faster to run. It also slightly outperforms ELECTRA-TT, which is consistent with the finding from Labeau and Allauzen (2018) that NCE outperforms negative sampling for training language models. Furthermore, Electric is simpler and faster than ELETRA-TT in that it does not require running the generator to produce PLL scores. TwoTower scores lower than Electric, presumably because it is not a "deeply" bidirectional model and instead just concatenates forward and backward hidden states.

5.5 Negative Results

I briefly describe a few ideas that did not look promising in my initial experiments:

- I initially attempted to make BERT more efficient by strategically masking out tokens (e.g., masking my rarer tokens more frequently, or training a model to guess which tokens BERT would struggle to predict if they were masked out). This resulted in fairly minor speedups over regular BERT.

Rescoring Model	**Pre-Training Data**	**Dev**		**Test**		**Transformer Passes**
		clean	other	clean	other	
None	–	7.17	19.79	7.26	20.37	0
BERT	WikiBooks	5.17	16.44	5.41	17.41	n
Electric	Wikibooks	5.47	16.56	5.65	17.42	1
GPT-2	OpenWebText	5.39	16.81	5.64	17.61	1
TwoTower	OpenWebText	5.12	16.37	5.32	17.25	1
ELECTRA-TT	OpenWebText	5.05	16.27	5.22	17.01	1
Electric	OpenWebText	4.97	16.23	5.18	16.93	1

Table 5.9: Word error rates on LibriSpeech after rescoring with base-sized models. None, GPT-2, and BERT results are from Salazar et al. (2020). Runtime is measured in passes through the Transformer and data indicates the pre-training dataset. "Clean" and "other" are easier and harder splits of the data. *I use a public re-implementation of OpenWebText.

- Given that ELECTRA seemed to benefit (up to a certain point) from having a weaker generator (see Section 5.3), I explored raising the temperature of the generator's output softmax or disallowing the generator from sampling the correct token. Neither of these improved results.

- I tried adding a sentence-level contrastive objective. For this task, I kept 20% of input sentences unchanged rather than noising them with the generator. I then added a prediction head to the model that predicted if the entire input was corrupted or not. Surprisingly, this slightly decreased scores on downstream tasks.

5.6 Previous Work

Previous work on self-supervised pre-training is detailed thoroughly in background section 2.5. Here I describe previous work in a few other areas relevant to ELECTRA.

Generative Adversarial Networks. GANs (Goodfellow et al., 2014) are effective at generating high-quality synthetic data. Radford et al. (2016) propose using the discriminator of a GAN in downstream tasks, which is similar to my method. GANs have been applied to text data (Yu et al., 2017; Zhang et al., 2017), although state-of-the-art approaches still lag behind standard maximum-likelihood training (Caccia et al., 2020; Tevet et al., 2018). Although it does not use adversarial learning, the generator is particularly reminiscent of MaskGAN (Fedus et al., 2018), which trains the generator to fill in tokens deleted from the input.

Contrastive Learning. Broadly, contrastive learning methods distinguish observed data points from fictitious negative samples. They have been applied to many modalities including text (Smith and Eisner, 2005), images (Chopra et al., 2005), and video (Wang and Gupta, 2015; Sermanet et al., 2017) data. Common approaches learn embedding spaces where related data points are similar (Saunshi et al., 2019) or models that rank real data points over negative samples (Collobert et al., 2011; Bordes et al., 2013). ELECTRA is particularly related to Noise-Contrastive Estimation (NCE) (Gutmann and Hyvärinen, 2010), which also trains a binary classifier to distinguish real and fake data points.

Word2Vec (Mikolov et al., 2013a), one of the earliest pre-training methods for NLP, uses contrastive learning. In fact, ELECTRA can be viewed as a massively scaled-up version of Continuous Bag-of-Words (CBOW) with Negative Sampling. CBOW also predicts an input token given surrounding context and negative sampling rephrases the learning task as a binary classification task on whether the input token comes from the data or proposal distribution. However, CBOW uses a bag-of-vectors encoder rather than a Transformer and a simple proposal distribution derived from unigram token frequencies instead of a learned generator.

Energy-Based Models. Energy-based models have been widely explored in machine learning (Dayan et al., 1995; LeCun et al., 2007). While many training methods involve sampling from the EBM using gradient-based MCMC (Du and Mordatch, 2019) or Gibbs sampling (Hinton, 2002), I considered these approaches too slow for pre-training because they require multiple passes through the model per sample. I instead use noise-contrastive estimation (Gutmann and Hyvärinen, 2010), which has widely been used in NLP for learning word vectors (Mnih and Kavukcuoglu, 2013) and text generation models (Jean et al., 2014; Józefowicz et al., 2016). While EBMs have previously been applied to left-to-right (Wang et al., 2015) or globally normalized (Rosenfeld et al., 2001; Deng et al., 2020) text generation, they have not previously been applied to cloze models or for pre-training NLP models. Several papers have pointed out the connection between EBMs and GANs (Zhao et al., 2017; Finn et al., 2016), which is similar to the Electric/ELECTRA connection.

Efficient Pre-Training. Models such as DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), and MobileBERT (Sun et al., 2020) show that BERT can effectively be distilled down into a smaller model. Another way of improving inference efficiency is through quantization, such as in Q8-BERT (Zafrir et al., 2019) or Q-BERT (Shen et al., 2020). In contrast, I focus more on pre-training speed rather than inference speed, so I train ELECTRA-Small from scratch. Several models have already been built on top of ELECTRA to further improve efficiency such as ConvBERT (Jiang et al., 2020b) and Funnel Transformer (Dai et al., 2020), which aim to use more efficient neural architectures than vanilla Transformers. MC-BERT (Xu et al., 2020), CodeBERT (Feng et al., 2020b), and KgPLM (He et al., 2020) explore extending ELECTRA's training algorithm or apply it to other domains. Many ELECTRA models have been trained in other languages as well, with over 250 models in Hugging Face's collection of pre-trained transformers[6] containing "ELECTRA" in their name.

5.7 Conclusion

I have proposed replaced token detection, a powerful self-supervised task for language representation learning. The key idea is training a text encoder to distinguish input tokens from high-quality negative samples produced by a small generator network. Compared to masked language modeling, my pre-training objective is more compute-efficient and results in better performance on downstream tasks. It works well even when using relatively small amounts of compute, which I hope will make developing and applying pre-trained text encoders more accessible to researchers and practitioners with less access to computing resources. Nevertheless, it also keeps the same excellent scalability that makes masked language models so successful. Overall, I believe ELECTRA is the closest I have come to realising the goals set out by this dissertation: it can be transferred to many tasks, scales well to large amounts of data, and learns efficiently.

[6]`https://huggingface.co/models?search=electra`

Chapter 6

Conclusion

The field of natural language processing has changed dramatically over the past few years, and transfer learning has been central to this change. Historically, much of NLP research has operated by developing a system that learns a specific task from human-labeled data. In contrast, it is now possible to download pre-trained models that can be fine-tuned to excellent performance at many tasks with minimal task-specific engineering. However, this rapid progress from transfer learning did not come for free: underlying this success is the use of large models requiring extensive compute resources to train. To alleviate these costs, this dissertation's goal has been to develop transfer learning methods that are efficient (making effective use of compute resources) while remaining scalable (yielding improvements as data and model sizes grow). I have realized this goal across three core areas of transfer learning: multi-task, semi-supervised, and self-supervised learning.

In Chapter 3, I develop a general NLP system with many capabilities through multi-task learning. Specifically, I present born-again multi-tasking (BAM), which uses knowledge distillation to achieve effective multi-task learning with consistent gains over single-task systems, even when training on many diverse tasks. BAM lets multi-task models scale to many tasks with less risk of task interference degrading results. In Chapter 4, I develop Cross-View Training (CVT), a variant of self-training particularly suited to text data. Similar to co-training, CVT enforces consistent predictions across different views of the input. However, it takes advantage of neural networks by using shared representation to transfer knowledge across input views. In contrast to the many recent pre-training methods in NLP, CVT learns representations targeted towards particular tasks: rather than learning broadly about language in general, the model more narrowly focuses on task-relevant information. This property makes CVT highly efficient, where it outperforms a comparably-sized ELMo model while training 10x faster.

In Chapter 5, I present ELECTRA, a highly efficient method for self-supervised pre-training of Transformer networks. Motivated from an energy-based model view of the cloze pre-training task, I develop a new pre-training task called replaced token detection. This task trains a transformer network to distinguish data tokens from fakes sampled from a generator network. Replaced token detection enables a bidirectional model like BERT to learn from all input tokens like a language model, getting the best of both worlds. ELECTRA works well at both ends of the compute spectrum, yielding strong results when the model is trained on a single GPU and learning around 4x faster than existing approaches when trained at large scale. BAM, CVT, and ELECTRA are effective at a diverse range of NLP tasks, ranging from syntactic ones such as dependency parsing to challenging natural language understanding tasks such as natural language inference and reading comprehension, demonstrating the generality of the methods.

I believe many of the trends exhibited in my dissertation closely mirror emerging trends in the whole field. One trend is a progression away from large labeled datasets. In this thesis, I present first a system learning only from labeled datasets (BAM), then one learning on a mix of labeled and unlabeled data (CVT), and lastly, one that learns purely from unlabeled text (ELECTRA). While models like BERT and ELECTRA are still fine-tuned on large labeled corpora, the field is moving away from even this, with recent work demonstrating the remarkable few-shot performance of language models (Brown et al., 2020). There is growing evidence that models learning purely from self-supervised tasks can pick up linguistic features previously added to models through learning from human annotations (Linzen et al., 2016; Hewitt and Manning, 2019). Ultimately, self-supervised methods like ELECTRA are surpassing the other transfer learning approaches because they scale the best and result in a more general model. That being said, other kinds of transfer learning still can have added value, with recent work shows that multi-task learning (Aghajanyan et al., 2021) and semi-supervised learning (Du et al., 2020) improve pre-trained models when applied at a large enough scale.

Another trend is the progression away from specific NLP tasks and towards more general NLP systems. While designing task-specific architectures and training methods was previously the primary way of improving NLP performance, highly general architectures (e.g., the Transformer) combined with large-scale self-supervised learning have enabled a single model to do many tasks. In this thesis, I have avoided focusing on task-specific enhancements because they ultimately stop being worth the effort when simply training a bit longer can produce larger gains. This trend of generality in NLP systems is continuing, with large language models displaying a remarkable ability to perform many different NLP tasks zero-shot (Radford et al., 2019).

Lastly, there is a trend towards ever-larger machine learning models. This trend will certainly continue, especially because it is generally more compute-efficient to train large models for few steps than it is to train smaller models for many steps (Kaplan et al., 2020). The great scaling properties of transfer learning make the future of NLP exciting because we will continue to see improved results as hardware advances allow for the training of bigger models. However, there are many open questions on *how* models should get bigger, which I think is an impactful area for future work. The most straightforward and widely used approach is to simply make the networks wider and deeper. However, with the importance of efficiency in mind, I believe it is essential to develop ways of scaling up models in more sophisticated, less compute-intensive ways. One area already seeing widespread research is improving model efficiency on long sequences through mechanisms like sparse attention (Dai et al., 2019; Child et al., 2019; Kitaev et al., 2020; Beltagy et al., 2020). Another direction is investigating how to increase parameters but not compute through sparse activation of weights. For example, sparsely gated mixture of experts (Shazeer et al., 2017; Fedus et al., 2021) dynamically routes examples to specialized parts of the network. Another research area with great potential is memory augmented neural networks (Khandelwal et al., 2020). These techniques add an external memory that the network can quickly access through a mechanism like maximum inner product search, allowing models to scale more efficiently by increasing the memory size. I find these directions particularly intriguing because they narrow a large discrepancy between neural networks and the brain: the brain is sparsely activated while most neural networks are not.

I also think there is interesting follow-up work for ELECTRA in particular. One direction is trying to bring the same efficiency gains to sequence-to-sequence pre-training methods such as BART (Lewis et al., 2020) and T5 (Raffel et al., 2020), which have the advantages of being usable for generation tasks and more naturally applied to few-shot learning. Another interesting topic is multilingual learning. Unlike generation-based pre-training, a multilingual ELECTRA model would not need to maintain information about the language of the text it sees because it does not have to generate tokens in that language, perhaps leading to more cross-lingual representations.

Looking towards a future filled with even larger transfer learning models, I hope my dissertation will inspire more research focusing on computationally efficient transfer learning. Efficient training will not only make models better, but also expand the number of researchers able to work in this area, leading to faster progress and broader research goals. Ultimately, I believe the future of NLP is bright because efficient and scalable transfer learning will continue to produce systems with remarkable and useful capabilities.

Bibliography

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2017. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *ICLR*.

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning. *ArXiv*, abs/2101.11038.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*.

Mikel Artetxe and Holger Schwenk. 2019. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *TACL*, 7:597–610.

Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In *NeurIPS*.

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

Philip Bachman, Ouais Alsharif, and Doina Precup. 2014. Learning with pseudo-ensembles. In *NeurIPS*.

Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. 2019. Cloze-driven pretraining of self-attention networks. In *EMNLP*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *ArXiv*, abs/2004.05150.

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. 2019. MixMatch: A holistic approach to semi-supervised learning. In *NeurIPS*.

Joachim Bingel and Anders Søgaard. 2017. Identifying beneficial task relations for multi-task learning in deep neural networks. In *EACL*.

Terra Blevins, Omer Levy, and Luke S. Zettlemoyer. 2018. Deep RNNs encode soft hierarchical syntax. In *ACL*.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *COLT*. ACM.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *TACL*, 5:135–146.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NeurIPS*.

Avishek Joey Bose, Huan Ling, and Yanshuai Cao. 2018. Adversarial contrastive estimation. In *ACL*.

Samuel R Bowman, Ellie Pavlick, Edouard Grave, Benjamin Van Durme, Alex Wang, Jan Hula, Patrick Xia, Raghavendra Pappagari, R Thomas McCoy, Roma Patel, et al. 2018. Looking for ELMo's friends: Sentence-level pretraining beyond language modeling. *ArXiv*, abs/1812.10860.

Peter F Brown, Vincent J Della Pietra, Peter V Desouza, Jennifer C Lai, and Robert L Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18:467–479.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.

Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. 2020. Language GANs falling short. In *ICLR*.

Jamie Callan, Mark Hoy, Changkuk Yoo, and Le Zhao. 2009. Clueweb09 data set. https://lemurproject.org/clueweb09.php/.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *ECCV*.

Rich Caruana. 1993. Multitask learning: A knowledge-based source of inductive bias. In *ICML*.

Rich Caruana. 1997. Multitask learning. *Machine Learning*, 28(1):41–75.

Daniel M. Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *SemEval@ACL*.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, Roldano Cattoni, and Marcello Federico. 2015. The IWSLT 2015 evaluation campaign. In *International Workshop on Spoken Language Translation*.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH*.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *ArXiv*, abs/1904.10509.

Jason PC Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional LSTM-CNNs. *TACL*, 4:357–370.

Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. *CVPR*.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. 2015. Attention-based models for speech recognition. In *NeurIPS*.

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

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D. Manning, and Quoc V. Le. 2019b. BAM! Born-again multi-task networks for natural language understanding. In *ACL*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020a. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020b. Pre-training transformers as energy-based cloze models. In *EMNLP*.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc V. Le. 2018. Semi-supervised sequence modeling with cross-view training. In *EMNLP*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural language processing (almost) from scratch. *JMLR*, 12:2493–2537.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *ACL*.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *NeurIPS*.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V Le. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *ArXiv*, abs/2006.03236.

Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Ruslan Salakhutdinov. 2017. Good semi-supervised learning that requires a bad GAN. In *NeurIPS*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL*.

Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. 1995. The Helmholtz machine. *Neural Computation*, 7:889–904.

Yuntian Deng, Anton Bakhtin, Myle Ott, and Arthur Szlam. 2020. Residual energy-based models for text generation. In *ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *IWP@IJCNLP*.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. In *NeurIPS*.

Linhao Dong, Shuang Xu, and Bo Xu. 2018. Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5888.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *ICLR*.

Jingfei Du, Edouard Grave, Beliz Gunel, Vishrav Chaudhary, Onur Celebi, Michael Auli, Ves Stoyanov, and Alexis Conneau. 2020. Self-training improves pre-training for natural language understanding. *ArXiv*, abs/2010.02194.

Yilun Du and Igor Mordatch. 2019. Implicit generation and generalization in energy-based models. In *NeurIPS*.

Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. 2015. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *ACL*.

Greg Durrett and Dan Klein. 2014. A joint model for entity analysis: Coreference, typing, and linking. *TACL*, 2:477–490.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. In *ACL*.

Manaal Faruqui and Chris Dyer. 2014. Improving vector space word representations using multilingual correlation. In *EACL*.

William Fedus, Ian J. Goodfellow, and Andrew M. Dai. 2018. MaskGAN: Better text generation via filling in the _______. In *ICLR*.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *ArXiv*, abs/2101.03961.

Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. 2020a. Language-agnostic BERT sentence embedding. *ArXiv*, abs/2007.01852.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020b. CodeBERT: A pre-trained model for programming and natural languages. In *EMNLP*.

Jenny Rose Finkel and Christopher D. Manning. 2009. Joint parsing and named entity recognition. In *HLT-NAACL*.

Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. 2016. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. In *NeurIPS 2016 Workshop on Adversarial Training*.

Tommaso Furlanello, Zachary C Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born again neural networks. In *ICML*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B. Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *ACL-PASCAL@ACL*.

Yoav Goldberg. 2019. Assessing BERT's syntactic abilities. *ArXiv*, abs/1901.05287.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep learning. *MIT Press*.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *NeurIPS*.

Stephan Gouws, Yoshua Bengio, and Greg Corrado. 2015. BilBOWA: Fast bilingual distributed representations without word alignments. In *ICML*.

Yves Grandvalet and Yoshua Bengio. 2005. Semi-supervised learning by entropy minimization. In *NeurIPS*.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5):602–610.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. 2018. Colorless green recurrent networks dream hierarchically. In *NAACL-HLT*.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *NAACL-HLT*.

Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A joint many-task model: Growing a neural network for multiple NLP tasks. In *EMNLP*.

Bin He, Xin Jiang, JingHui Xiao, and Qun Liu. 2020. KgPLM: Knowledge-guided language model pre-training via generative and discriminative learning. *ArXiv*, abs/2012.03551.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *CVPR*.

John Hewitt and Christopher D Manning. 2019. A structural probe for finding syntax in word representations. In *NAACL-HLT*.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning distributed representations of sentences from unlabelled data. In *NAACL-HLT*.

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

Geoffrey E. Hinton. 2002. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800.

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

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

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn treebank. *Computational Linguistics*, 33(3):355–396.

Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, 6:65–70.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *ICLR*.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. OntoNotes: the 90% solution. In *NAACL-HLT*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *ACL*.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First Quora dataset release: Question pairs. https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. In *ACL*.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *ACL*.

Zhengbao Jiang, Wei Xu, Jun Araki, and Graham Neubig. 2020a. Generalizing natural language analysis through span-relation representations. In *ACL*.

Zihang Jiang, Weihao Yu, Daquan Zhou, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. 2020b. ConvBERT: Improving BERT with span-based dynamic convolution. In *NeurIPS*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *EMNLP*.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2019. SpanBERT: Improving pre-training by representing and predicting spans. *ArXiv*, abs/1907.10529.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *ArXiv*, abs/1602.02410.

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

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019a. CTRL: A conditional transformer language model for controllable generation. *ArXiv*, abs/1909.05858.

Nitish Shirish Keskar, Bryan McCann, Caiming Xiong, and Richard Socher. 2019b. Unifying question answering and text classification via span extraction. *ArXiv*, abs/1904.09286.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. 2018. Sharp nearby, fuzzy far away: How neural language models use context. In *ACL*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *ICLR*.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *EMNLP*.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114 13:3521–3526.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *NeurIPS*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *ICLR*.

Alex Krizhnevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one mst parser. In *EMNLP*.

Matthieu Labeau and Alexandre Allauzen. 2018. Learning with noise-contrastive estimation: Easing training by learning to scale. In *COLING*.

Samuli Laine and Timo Aila. 2017. Temporal ensembling for semi-supervised learning. In *ICLR*.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. In *NeurIPS*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *ICML*.

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. 2007. A tutorial on energy-based learning. In *Predicting Structured Data*, pages 191–246. MIT Press, Cambridge, MA.

Hector J. Levesque. 2011. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*.

Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. 2019. VisualBERT: A simple and performant baseline for vision and language. *ArXiv*, abs/1908.03557.

Zhizhong Li and Derek Hoiem. 2016. Learning without forgetting. In *ECCV*.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *TACL*, 4:521–535.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019a. Linguistic knowledge and transferability of contextual representations. In *NAACL-HLT*.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Adversarial multi-task learning for text classification. In *ACL*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019b. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. *ArXiv*, abs/1904.09482.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019c. Multi-task deep neural networks for natural language understanding. In *ACL*.

Yanjun Liu, Fandong Meng, Jinchao Zhang, Jinan Xu, Yufeng Chen, and Jie Zhou. 2019d. GCDT: A global context enhanced deep transition architecture for sequence labeling. In *ACL*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019e. RoBERTa: A robustly optimized BERT pretraining approach. *ArXiv*, abs/1907.11692.

Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. ViLBERT: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *NeurIPS*.

Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. 2017. Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*.

Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2016. Multi-task sequence to sequence learning. In *ICLR*.

Minh-Thang Luong and Christopher D. Manning. 2015. Stanford neural machine translation systems for spoken language domains. In *IWSLT*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015a. Bilingual word representations with monolingual quality in mind. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*.

Thang Luong, Hieu Quang Pham, and Christopher D. Manning. 2015b. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Aleksandr Romanovich Luriia. 1976. *Cognitive development: Its cultural and social foundations*. Harvard university press.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNN-CRF. In *ACL*.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *ACL*.

Zhuang Ma and Michael Collins. 2018. Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency. In *EMNLP*.

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

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *NeurIPS*.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *ArXiv*, abs/1806.08730.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *ACL*.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In *ICLR Workshop Papers*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *NeurIPS*.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-stitch networks for multi-task learning. In *CVPR*.

Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using LSTMs on sequences and tree structures. In *ACL*.

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2017a. Adversarial training methods for semi-supervised text classification. In *ICLR*.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. 2017b. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *ArXiv*, abs/1704.03976.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. 2016. Distributional smoothing with virtual adversarial training. In *ICLR*.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *NeurIPS*.

Lili Mou, Ran Jia, Yan Xu, Ge Li, Lu Zhang, and Zhi Jin. 2016. Distilling word embeddings: An encoding approach. In *CIKM*.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.

Nikita Nangia and Samuel R. Bowman. 2019. Human vs. muppet: A conservative estimate of human performance on the GLUE benchmark. In *ACL*.

Guillaume Obozinski, Ben Taskar, and Michael Jordan. 2006. Multi-task feature selection. *Statistics Department, UC Berkeley, Technical Report*, 2(2.2):2.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. LibriSpeech: An ASR corpus based on public domain audio books. *ICASSP*.

Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. 2016. Actor-mimic: Deep multitask and transfer reinforcement learning. In *ICLR*.

Emilio Parisotto, Francis Song, Jack W. Rae, Razvan Pascanu, Çaglar Gülçehre, Siddhant M. Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew Botvinick, Nicolas Heess, and Raia Hadsell. 2020. Stabilizing transformers for reinforcement learning. In *ICML*.

Sungrae Park, Jun-Keon Park, Su-Jin Shin, and Il-Chul Moon. 2018. Adversarial dropout for supervised and semi-supervised learning. In *AAAI*.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. English Gigaword, fifth edition. Technical report, Linguistic Data Consortium, Philadelphia.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep multitask learning for semantic dependency parsing. In *ACL*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. 2017. Regularizing neural networks by penalizing confident output distributions. In *ICLR*.

David N Perkins, Gavriel Salomon, et al. 1992. Transfer of learning. *International encyclopedia of education*, 2:6452–6457.

Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *ACL*.

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

Jason Phang, Thibault Févry, and Samuel R Bowman. 2018. Sentence encoders on STILTs: Supplementary training on intermediate labeled-data tasks. *ArXiv*, abs/1811.01088.

Barbara Plank and Héctor Martínez Alonso. 2017. When is multitask learning effective? Semantic sequence prediction under varying data conditions. In *EACL*.

Boris T Polyak. 1964. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *https://blog.openai.com/language-unsupervised*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical Report.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21:140:1–140:67.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *NAACL-HLT*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy S. Liang. 2016. SQuAD: 100, 000+ questions for machine comprehension of text. In *EMNLP*.

Marek Rei. 2017. Semi-supervised multitask learning for sequence labeling. In *ACL*.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *EMNLP*.

Ronald Rosenfeld, Stanley F. Chen, and Xiaojin Zhu. 2001. Whole-sentence exponential language models: A vehicle for linguistic-statistical integration. *Computer Speech & Language*, 15:55–73.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *ArXiv*, abs/1706.05098.

Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. 2019. Latent multi-task architecture learning. In *AAAI*.

Sebastian Ruder and Barbara Plank. 2018. Strong baselines for neural semi-supervised learning under domain shift. In *ACL*.

Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. 2016. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *NeurIPS*.

Julian Salazar, Davis Liang, Toan Q. Nguyen, and Katrin Kirchhoff. 2020. Masked language model scoring. In *ACL*.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training GANs. In *NeurIPS*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Nikunj Saunshi, Orestis Plevrakis, Sanjeev Arora, Mikhail Khodak, and Hrishikesh Khandeparkar. 2019. A theoretical analysis of contrastive unsupervised representation learning. In *ICML*.

H Scudder. 1965. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3):363–371.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Improving neural machine translation models with monolingual data. In *ACL*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *ACL*.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. *ArXiv*, abs/1611.01603.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine. 2017. Time-contrastive networks: Self-supervised learning from video. *ICRA*.

Shikhar Sharma, Ryan Kiros, and Ruslan Salakhutdinov. 2016. Action recognition using visual attention. In *ICLR Workshop*.

Noam Shazeer, A. Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian based ultra low precision quantization of BERT. In *AAAI*.

Xing Shi, Inkit Padhi, and Kevin Knight. 2016. Does string-based neural MT learn source syntax? In *EMNLP*.

Joonbo Shin, Yoonhyung Lee, and Kyomin Jung. 2019. Effective sentence scoring method using BERT for speech recognition. In *Asian Conference on Machine Learning*, pages 1081–1093.

Vikas Sindhwani and Mikhail Belkin. 2005. A co-regularization approach to semi-supervised learning with multiple views. In *ICML Workshop on Learning with Multiple Views*.

David A Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *EMNLP*.

Noah A. Smith and Jason Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *ACL*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *ACL*.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. MASS: Masked sequence to sequence pre-training for language generation. In *ICML*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *ACL*.

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate sequence labeling with iterated dilated convolutions. In *EMNLP*.

Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J Pal. 2018. Learning general purpose distributed sentence representations via large scale multi-task learning. In *ICLR*.

Chen Sun, Fabien Baradel, Kevin Murphy, and Cordelia Schmid. 2019a. Learning video representations using contrastive bidirectional transformer. *ArXiv*, abs/1906.05743.

Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. 2019b. VideoBERT: A joint model for video and language representation learning. *ICCV*.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. 2019c. ERNIE: Enhanced representation through knowledge integration. *ArXiv*, abs/1904.09223.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *ACL*.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *ICML*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NeurIPS*.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *CVPR*.

Hao Tan and Mohit Bansal. 2019. LXMERT: Learning cross-modality encoder representations from transformers. In *EMNLP*.

Xu Tan, Yi Ren, Di He, Tao Qin, and Tie-Yan Liu. 2019. Multilingual neural machine translation with knowledge distillation. In *ICLR*.

Antti Tarvainen and Harri Valpola. 2017a. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *NeurIPS*.

Antti Tarvainen and Harri Valpola. 2017b. Weight-averaged consistency targets improve semi-supervised deep learning results. In *Workshop on Learning with Limited Labeled Data, NeurIPS*.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. 2017. Distral: Robust multitask reinforcement learning. In *NeurIPS*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *ACL*.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. 2018. What do you learn from context? Probing for sentence structure in contextualized word representations. In *ICLR*.

Guy Tevet, Gavriel Habib, Vered Shwartz, and Jonathan Berant. 2018. Evaluating text GANs as language models. In *NAACL-HLT*.

Erik F Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *CoNLL*.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *NAACL-HLT*.

Joseph P. Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *ACL*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.

Vikas Verma, Alex Lamb, C. Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. 2019. Manifold mixup: Better representations by interpolating hidden states. In *ICML*.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *ICML*.

Alex Wang and Kyunghyun Cho. 2019. BERT has a mouth, and it must speak: BERT as a markov random field language model. *ArXiv*, abs/1902.04094.

Alex Wang, Amapreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Bin Wang and Zhijian Ou. 2018. Learning neural trans-dimensional random field language models with noise-contrastive estimation. In *ICASSP*.

Bin Wang, Zhijian Ou, and Zhiqiang Tan. 2015. Trans-dimensional random fields for language modeling. In *ACL*.

Wei Wang, Bin Bi, Ming Yan, Chen Wu, Zuyi Bao, Liwei Peng, and Luo Si. 2020. StructBERT: Incorporating language structures into pre-training for deep language understanding. In *ICLR*.

Xiaolong Wang and Abhinav Gupta. 2015. Unsupervised learning of visual representations using videos. In *ICCV*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2018. Neural network acceptability judgments. *ArXiv*, abs/1805.12471.

Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, Adithya Renduchintala, and Tsubasa Ochiai. 2018. ESPnet: End-to-end speech processing toolkit. In *INTERSPEECH*.

Xiang Wei, Zixia Liu, Liqiang Wang, and Boqing Gong. 2018. Improving the improved training of Wasserstein GANs. In *ICLR*.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Towards universal paraphrastic sentence embeddings. In *ICLR*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-–4):229–256.

Sam Joshua Wiseman, Alexander Matthew Rush, Stuart Merrill Shieber, and Jason Weston. 2015. Learning anaphoricity and antecedent ranking features for coreference resolution. In *ACL*.

Huijia Wu, Jiajun Zhang, and Chengqing Zong. 2017. Shortcut sequence tagging. *ArXiv*, abs/1701.00576.

Caiming Xiong, Victor Zhong, and Richard Socher. 2017. Dynamic coattention networks for question answering. In *ICLR*.

Chang Xu, Dacheng Tao, and Chao Xu. 2013. A survey on multi-view learning. *ArXiv*, abs/1304.5634.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan R. Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*.

Zhenhui Xu, Linyuan Gong, Guolin Ke, Di He, Shuxin Zheng, Liwei Wang, Jiang Bian, and Tie-Yan Liu. 2020. MC-BERT: Efficient language pre-training via a meta controller. *ArXiv*, abs/2006.05744.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2020. mT5: A massively multilingual pre-trained text-to-text transformer. *ArXiv*, abs/2010.11934.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *IWPT*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

David Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *ACL*.

Lantao Yu, Weinan Zhang, Jun Wang, and Yingrui Yu. 2017. SeqGAN: Sequence generative adversarial nets with policy gradient. In *AAAI*.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit BERT. In *Workshop on Energy Efficient Machine Learning and Cognitive Computing, NeurIPS*.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2018. mixup: Beyond empirical risk minimization. In *ICLR*.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *ICML*.

Kelly W. Zhang and Samuel R. Bowman. 2018. Language modeling teaches you more syntax than translation does: Lessons learned through auxiliary task analysis. In *BlackboxNLP@EMNLP*.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. 2017. Adversarial feature matching for text generation. In *ICML*.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *ACL*.

Junbo Zhao, Michael Mathieu, and Yann LeCun. 2017. Energy-based generative adversarial network. In *ICLR*.

Junru Zhou and Zhao Hai. 2019. Head-driven phrase structure grammar parsing on penn treebank. In *ACL*.

Zhi-Hua Zhou and Ming Li. 2005. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on knowledge and Data Engineering*.

Yukun Zhu, 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 *ICCV*.