Generative Feature Learning

Zhilin Yang

July 24

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Ruslan Salakhutdinov (Co-Chair), Carnegie Mellon University
William W. Cohen (Co-Chair), Carnegie Mellon University
Graham Neubig, Carnegie Mellon University
Jason Weston, Facebook AI Research

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Copyright © 2019 Zhilin Yang
Abstract

It is crucial to use unlabeled data to improve learning because unlabeled data is more accessible and more abundant than labeled data. There are two major learning paradigms towards this goal—the unsupervised pretraining method pretrains a language model on unlabeled data and performs finetuning on downstream tasks, while semi-supervised learning jointly optimizes loss functions on labeled and unlabeled data. In this thesis, we study the problem of generative feature learning, which aims to use generative modeling to leverage unlabeled data for boosting target task performance.

We propose novel methods to address critical challenges in generative feature learning. In the context of unsupervised learning, Transformer-XL is a novel architecture that enables modeling longer-range dependency. Built upon it, XLNet employs a permutation language modeling objective to bridge the gap between language modeling and unsupervised pretraining, which opens the possibility of applying language modeling progress to improves downstream tasks. In the setting of semi-supervised learning, we theoretically answer two open questions of GAN-based semi-supervised learning, which provides more fundamental understanding about why and how GANs work for semi-supervised learning. In addition, we pioneered the research of generative modeling based semi-supervised learning for tasks with complex data structures such as classification on graphs and question answering.

Empirically, our generative feature learning methods have played a leading role in the development of a variety of research topics. The proposed methods obtained the state-of-the-art results on more than 30 benchmarks, including natural language inference, question answering, text classification, language modeling, semi-supervised learning, etc., demonstrating significant practical value.
# Contents

1 Introduction

1.1 Motivation: Learning from Unlabeled Data .................................................. 1
1.2 Background ........................................................................................................ 2
1.3 Challenges ........................................................................................................... 3
1.4 Contributions ..................................................................................................... 4

2 Unsupervised Pretraining with Generative Modeling ........................................... 7

2.1 Attentive Language Models Beyond a Fixed-Length Context ............................. 7
  2.1.1 Motivation ...................................................................................................... 7
  2.1.2 Background .................................................................................................... 8
  2.1.3 Approach ........................................................................................................ 9
  2.1.4 Experiments .................................................................................................. 13

2.2 XLNet: Generalized Autoregressive Pretraining ................................................. 29
  2.2.1 Motivation ...................................................................................................... 29
  2.2.2 Approach ........................................................................................................ 30
  2.2.3 Experiments .................................................................................................. 39

3 Semi-Supervised Learning with Generative Modeling .......................................... 45

3.1 Semi-Supervised Learning with GANs ............................................................... 45
  3.1.1 Motivation ...................................................................................................... 45
  3.1.2 Related Work ................................................................................................. 46
  3.1.3 Theoretical Analysis ...................................................................................... 47
  3.1.4 Case Study on Synthetic Data ........................................................................ 52
  3.1.5 Approach ........................................................................................................ 54
  3.1.6 Experiments .................................................................................................. 57

3.2 Semi-Supervised QA by Generating Questions .................................................. 59
  3.2.1 Motivation ...................................................................................................... 59
  3.2.2 Related Work ................................................................................................. 60


# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Illustration of the vanilla model with a segment length 4.</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Illustration of the Transformer-XL model with a segment length 4.</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Visualizing unnormalized relative perplexity gains with ( r = 0.1 ).</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Perplexity vs context length.</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Illustration of the permutation language modeling objective for predicting ( x_3 ) given the same input sequence ( x ) but with different factorization orders.</td>
<td>31</td>
</tr>
<tr>
<td>2.6</td>
<td>(a): Content stream attention, which is the same as the standard self-attention.</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>(b): Query stream attention, which does not have access information about the content ( x_z ).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(c): Overview of the permutation language modeling training with two-stream attention.</td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>A detailed illustration of the <strong>content stream</strong> of the proposed objective with both the joint view and split views based on a length-4 sequence under the factorization order ([3, 2, 4, 1]). Note that if we ignore the query representation, the computation in this figure is simply the standard self-attention, though with a particular attention mask.</td>
<td>36</td>
</tr>
<tr>
<td>2.8</td>
<td>A detailed illustration of the <strong>query stream</strong> of the proposed objective with both the joint view and split views based on a length-4 sequence under the factorization order ([3, 2, 4, 1]). The dash arrows indicate that the query stream cannot access the token (content) at the same position, but only the location information.</td>
<td>37</td>
</tr>
<tr>
<td>3.1</td>
<td>Percentage of the test samples that satisfy the assumption under our best model.</td>
<td>52</td>
</tr>
<tr>
<td>3.2</td>
<td>Labeled and unlabeled data are denoted by cross and point respectively, and different colors indicate classes.</td>
<td>53</td>
</tr>
<tr>
<td>3.3</td>
<td>Left: Classification decision boundary, where the white line indicates true-fake boundary; Right: True-Fake decision boundary.</td>
<td>53</td>
</tr>
<tr>
<td>3.4</td>
<td>Feature space at convergence</td>
<td>53</td>
</tr>
<tr>
<td>3.5</td>
<td>Left: Blue points are generated data, and the black shadow indicates unlabeled data.</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>Middle and right can be interpreted as above.</td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>Comparing images generated by FM and our model. FM generates collapsed samples, while our model generates diverse “bad” samples.</td>
<td>57</td>
</tr>
</tbody>
</table>
3.7 Model architecture and training. Red boxes denote the modules being updated. “d_true” and “d_gen” are two domain tags. \( D \) is the discriminative model and \( G \) is the generative model. The objectives for the three cases are all to minimize the cross entropy loss of the answer chunks.

3.8 Comparison of discriminator training loss \(-J(U_G, d_{gen}, D)\) on generated QA pairs. The lower the better. MLE refers to questions generated by maximum likelihood training, and RL refers to questions generated by reinforcement learning.

3.9 An example of sampling from context distribution \( p(i, c, \gamma) \) when \( \gamma = 1 \) and \( d = 2 \). In circles, \(+1\) denotes positive instances, \(-1\) denotes negative instances, and \(?\) denotes unlabeled instances. If \( random < r_2 \), we first sample a random walk \( 2 \rightarrow 1 \rightarrow 4 \rightarrow 6 \), and then sample two nodes in the random walk within distance \( d \). If \( random \geq r_2 \), we sample two instances with the same labels.

3.10 Network architecture: transductive v.s. inductive. Each dotted arrow represents a feed-forward network with an arbitrary number of layers (we use only one layer in our experiments). Solid arrows denote direct connections.

3.11 t-SNE Visualization of embedding spaces on the Cora dataset. Each color denotes a class.
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison with state-of-the-art results on WikiText-103. * indicates contempo-</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>rary work.</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>Comparison with state-of-the-art results on enwik8.</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>Comparison with state-of-the-art results on text8.</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>Comparison with state-of-the-art results on One Billion Word. * indicates contem-</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>porary work.</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>Comparison with state-of-the-art results on Penn Treebank. † indicates using</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>two-step finetuning.</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>Ablation study on WikiText-103. For the first two blocks, we use a slightly</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>smaller model (128M parameters). † indicates that the corresponding row is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>reduced to the same setting as the Transformer network in [1], except that two</td>
<td></td>
</tr>
<tr>
<td></td>
<td>auxiliary losses are not implemented in our experiments. “PPL init” refers to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>using the same length as training. “PPL best” indicates the perplexity obtained</td>
<td></td>
</tr>
<tr>
<td></td>
<td>by using the optimal length. “Attn Len” is the shortest possible attention</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length during evaluation to achieve the corresponding result (PPL best).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increasing the attention length during evaluation improves performance only</td>
<td></td>
</tr>
<tr>
<td></td>
<td>when our positional encoding is used. The “Transformer-XL (151M)” setting uses</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a standard parameter budget as previous work [93], where we observe a similar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>effect when increasing the attention length during evaluation.</td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>Ablation study on One Billion Word, a dataset without long-term dependency.</td>
<td>18</td>
</tr>
<tr>
<td>2.8</td>
<td>Ablation study on WikiText-103 with the same GPU memory constraints.</td>
<td>18</td>
</tr>
<tr>
<td>2.9</td>
<td>Relative effective context length (RECL) comparison. See text for the definition</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>of RECL and r. The first three models and the last four models are compared</td>
<td></td>
</tr>
<tr>
<td></td>
<td>as two model groups when we calculate RECL (RECL is computed on a model group</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rather than a single model). Each group has the same parameter budget.</td>
<td></td>
</tr>
<tr>
<td>2.10</td>
<td>Slowdown in terms of running time during evaluation. Evaluation is based on</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>per-token time on one GPU.</td>
<td></td>
</tr>
<tr>
<td>2.14</td>
<td>Comparison with state-of-the-art results on the test set of RACE, a reading</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>comprehension task. * indicates using ensembles. “Middle” and “High” in RACE are</td>
<td></td>
</tr>
<tr>
<td></td>
<td>two subsets representing middle and high school difficulty levels. All BERT and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XLNet results are obtained with a 24-layer architecture with similar model sizes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(aka BERT-Large). Our single model outperforms the best ensemble by 7.6 points</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in accuracy.</td>
<td></td>
</tr>
</tbody>
</table>
2.15 A single model XLNet outperforms human and the best ensemble by 7.6 EM and 2.5 EM on SQuAD1.1. * means ensembles, † marks our runs with the official code.

2.16 Comparison with state-of-the-art error rates on the test sets of several text classification datasets. All BERT and XLNet results are obtained with a 24-layer architecture with similar model sizes (aka BERT-Large).

2.17 Results on GLUE. * indicates using ensembles, and † denotes single-task results in a multi-task row. All results are based on a 24-layer architecture with similar model sizes (aka BERT-Large). See the upper-most rows for direct comparison with BERT and the lower-most rows for comparison with state-of-the-art results on the public leaderboard.

2.18 Comparison with state-of-the-art results on the test set of ClueWeb09-B, a document ranking task. † indicates our implementations.

2.19 Ablation study. The results of BERT on RACE are taken from [173]. We run BERT on the other datasets using the official implementation and the same hyperparameter search space as XLNet. K is a hyperparameter to control the optimization difficulty (see Section 2.2.2). All models are pretrained on the same data.

3.1 Comparison with state-of-the-art methods on three benchmark datasets. Only methods without data augmentation are included. * indicates using the same (small) discriminator architecture, † indicates using a larger discriminator architecture, and ‡ means self-ensembling.

3.2 Ablation study. FM is feature matching. LD is the low-density enforcement term in Eq. (3.4). VI and PT are two entropy maximization methods described in Section 3.1.5. Ent means the conditional entropy term in Eq. (3.6). Max log-p is the maximum log probability of generated samples, evaluated by a PixelCNN++ model. 10-quant shows the 10-quantile of true image log probability. Error means the number of misclassified examples on MNIST, and error rate (%) on others.

3.3 Sampled generated questions given the paragraphs and the answers. P means paragraphs, A means answers, GQ means groundtruth questions, and Q means questions generated by our models. MLE refers to maximum likelihood training, and RL refers to reinforcement learning so as to maximize \( J(U_G, d_{true}, D) \). We truncate the paragraphs to only show tokens around the answer spans with a window size of 20.

3.4 Performance with various labeling rates, unlabeled data sizes \(|U|\), and methods. “Dev” denotes the development set, and “test” denotes the test set. F1 and EM are two metrics.

3.5 Dataset statistics.

3.6 Accuracy on text classification. Upper rows are inductive methods and lower rows are transductive methods.

3.7 Recall@k on DIEL distantly-supervised entity extraction. Upper rows are inductive methods and lower rows are transductive methods. Results marked with * are taken from the original DIEL paper [12] with the same data splits.

3.8 Accuracy on NELL entity classification with labeling rates of 0.1, 0.01, and 0.001. Upper rows are inductive methods and lower rows are transductive methods.
4.1 Summary and comparison of how different methods proposed in this work instantiate generative feature learning. AR means autoregressive modeling. “Comp GAN” refers to Complement GAN, “Semi QA” means semi-supervised QA, and “Planetoid” is our graph-based semi-supervised learning method.
Chapter 1

Introduction

1.1 Motivation: Learning from Unlabeled Data

For most machine learning applications and tasks, unlabeled data is more accessible and more abundant than labeled data. For example, major question answering datasets [78, 113, 172] nowadays typically have around 100K training examples in the form of question-answer pairs, which are tens of millions of words. In comparison, it is relatively easy to obtain 1,000 times more unlabeled English text data [15, 24, 105, 180]. In addition, there is a large number of other natural language understanding datasets that are significantly smaller, with only a few thousand training examples or less [148]. Furthermore, given a specific task, while it is possible to collect more labeled data, the procedure of labeled data collection is expensive and time-consuming. Therefore, it is unlikely to create a labeled dataset that matches the scale of unlabeled data. As a result, apart from the standard paradigm of learning on labeled data with supervised training signals, it is crucial to devise methods that use unlabeled data to aid the learning procedure.

In this thesis, we are interested in the problem of learning from unlabeled data, which can be generically formulated as follows. Given a set of labeled data and a (much larger) set of unlabeled data, we aim to design machine learning algorithms and models to fully leverage the unlabeled data and improve over systems that only have access to the labeled data.

There are two directions of research in the literature that fit into the above problem definition, namely unsupervised pretraining and semi-supervised learning. Although both directions have the same generic goal and motivation, they mainly differ in two aspects.

- Paradigm of knowledge transfer. The knowledge transfer procedure of unsupervised pretraining usually consists of two phases—pretraining and finetuning. Specifically, a model is first pretrained on unlabeled data and then finetuned on labeled data. On the contrary, in the setting of semi-supervised learning, a model is often jointly trained on both labeled and unlabeled data in one single training phase. As a consequence, unsupervised pretraining

---

1In the most general definition of semi-supervised learning, unsupervised pretraining is a special case of semi-supervised learning. However, in this thesis, we use the term “semi-supervised learning” in the narrow sense.

2Throughout the thesis, we use “unsupervised pretraining” to refer to either the sole pretraining phase or both phases combined, interchangeably depending on the context.
targets multiple tasks while semi-supervised learning mainly focuses on one task.

• Data scale and distribution. The scale of unlabeled data in the setting of unsupervised pretraining is usually much larger than that of semi-supervised learning. For example, in the field of natural language understanding, unsupervised pretraining typically uses more than billions of tokens during the pretraining phase. In comparison, in a normal setting of semi-supervised learning, the scale of unlabeled data can be 10 to 1000 times smaller. The difference in data scale also results in a difference in data distribution. Typical unsupervised pretraining covers a wider range of unlabeled data distribution such as the entire Web, while semi-supervised learning uses unlabeled data that is relatively more domain-specific with a distribution closer to labeled data.

1.2 Background

Among other approaches, generative modeling has been an important method for leveraging unlabeled data, in the settings of both unsupervised pretraining and semi-supervised learning. In the most generic form, given some data $x$, a generative model aims to estimate the log likelihood $\log p(x)$. The log likelihood might be defined explicitly by a model that maps data to probabilities, or implicitly modeled as a transformation from a prior distribution to data as in GANs [43]. The benefits of using generative modeling objectives are the wide applicability. In principle, it is possible to estimate the log likelihood of data in any format, including text, images, and complex structures such as graphs.

For unsupervised pretraining, generative modeling has played an important role. The original idea [26] was to optimize a language modeling (i.e., a form of generative modeling on text) loss on unsupervised text corpora, and the learned model parameters along with the architecture are used as initialization for fine-tuning on target supervised tasks. The learning paradigm was proposed in 2015 and obtained increasing attention since 2017 with the success of a series of models, including ELMo [109], CoVe [89], GPT [111], and BERT [32]. These methods differ in terms of training data sizes, model sizes, amounts of computation, model architectures, and objective functions. From the perspectives of models and algorithms, there are three major advances.

1. Architecture. The field of unsupervised pretraining has witnessed a transition from recurrent neural networks (RNNs) [95], especially LSTMs [55], to Transformers [145]. Due to the architectural design that involves direct connections between any two input units, Transformers have been shown to obtain better performance in machine translation [145] compared to RNNs. The most substantial improvement of Transformers over RNNs is in language modeling. Recent work [1, 29] shows Transformers and their variants substantially improve the ability of modeling long-range dependency and are able to scale without much optimization difficulty. These advantages of Transformers pave the road for the success of applications in pretraining.

2. Fine-tuning. While some of the earlier methods [89, 109] use the pretrained models as feature extractors, GPT [111] reinvented the idea of fine-tuning on downstream tasks, which was originally proposed by [26]. The fine-tuning approach increases the percentage of
pretrained parameters among all trainable parameters. Critically, when the input has multiple sequences, such as a question and a context for the task of question answering, GPT uses special symbols to join multiple sequences into one, so that finetuning can be performed on the concatenated sequence. As a result, this allows modeling the dependency between multiple input sequences with multiple pretrained layers. For some tasks such as question answering, this design proved important [11].

3. Bidirectional context. Earlier approaches are based on the standard language modeling objective. As a result, these approaches suffer from not being able to model bidirectional contexts with a multi-layer, deep pretrained network. BERT [32] addressed this issue by using a denoising autoencoding [146] based method, which yields improvement over previous methods.

Generative modeling has also been applied to semi-supervised learning. Variational autoencoders [69] were adapted to the semi-supervised learning setting in [70] by treating the classification label as an additional latent variable in a directed generative model. [87] adds auxiliary variables to the deep VAE structure to make variational distribution more expressive. Later, generative adversarial networks (GANs) started to be applied to semi-supervised learning as well. One of the earliest examples was CatGAN [131], which substitutes the binary discriminator in a standard GAN with a multi-class classifier, and trains both the generator and the discriminator using a information theoretical criteria on unlabeled data. This multi-class formulation has been a successful framework for semi-supervised learning. Under this framework, the feature matching GAN [120] further improves performance by considering a feature matching objective for the generator.

1.3 Challenges

Although generative modeling based methods have been successful in the setting of learning from unlabeled data, there remain a few critical challenges.

• In the setting of unsupervised pretraining, although BERT allows modeling bidirectional contexts, it introduces a new problem—BERT is not able to model the dependency between the target tokens for prediction due to the assumption that all target tokens are independent. On the contrary, standard language modeling is based on the product rule and thus does not suffer from this problem, but it is only able to model unidirectional contexts. It is not clear how to combine the advantages of different approaches while avoiding their disadvantages.

• Language modeling has been a long-standing, rapidly-developing area of research. However, not being able to model bidirectional contexts poses a question regarding whether there are meaningful applications of language modeling besides text generation.

• Conventional language modeling approaches rely on RNNs, which are not the best models for capturing long-range dependency due to the problem of gradient explosion and vanishing [106]. Transformers demonstrate a large potential in this aspect because of the direct connections between all input units. However, previous Transformer language models employ a fixed-length context, which prohibits capturing any longer-range dependency.
beyond the predefined context length. In addition, the fixed-length segments are created by selecting a consecutive chunk of symbols without respecting the sentence or any other semantic boundary. Hence, the models lacks necessary contextual information to well predict the first few symbols, leading to inefficient optimization and inferior performance.

- In the setting of semi-supervised learning, although GAN-based approaches have shown superior performance, little has been understood about why it works. Prominently, there are two questions. First, why does the performance of classification seem contradictory to the quality of generated samples in empirical studies? Second, how does the classifier benefit from joint training with a generator?

- Despite the success of generative modeling based semi-supervised learning, most efforts focus on the standard setting of classification. However, these methods are not directly applicable to tasks with more complex data structures, such as classification on graphs and question answering.

### 1.4 Contributions

This thesis aims to tackle the above challenges using generative modeling. We first use a generic language to describe the idea of our learning framework called generative feature learning and then introduce technical advances under this framework.

The framework of generative feature learning can be described as follows. Given unlabeled data \( D_u = \{ x \} \) and labeled data \( D_l = \{ (x, y) \} \), we optimize the following two losses:

\[
\mathcal{L}_g = \mathbb{E}_{x \sim D_u} l_g(x, G_{\theta, \phi})
\]

\[
\mathcal{L}_t = \mathbb{E}_{x, y \sim D_l} l_t(y, T_{\theta, \psi}(x))
\]

where \( l_g \) and \( l_t \) are loss functions defined on each data point. \( G_{\theta, \phi} \) is a generative model with two parameters \( \theta \) and \( \phi \), and \( T_{\theta, \psi} \) is a target task model with parameters \( \theta \) and \( \psi \), where \( \theta \) is shared between the two models. The target task model maps some input \( x \) to output \( \hat{y} \), which could be predictions of classification labels, question answers, etc.

The above formulation is kept generic to allow flexible design choices:

- The generative model \( G \) can be an implicit model or an explicit model. An implicit model transforms a noise vector \( z \) sampled from some prior distribution into a data point \( x = G(z) \). An explicit model predicts the log probability \( \log p_G(x) = G(x) \) of some data point \( x \).

- The loss function \( l_g \) can be instantiated in different ways. For explicit generative models, \( l_g \) is usually defined as the negative log likelihood of \( x \), written as \( -G(x) \) or \( -\log p_G(x) \). For implicit generative models, the definition of \( l_g \) is more flexible and more closely integrated with the target loss function \( l_t \).

- The target loss function \( l_t \) could be instantiated based on specific target tasks, such as a standard classification cross entropy loss.

- The optimization paradigm could vary. It is possible to employ a multi-phase training procedure where each phase optimizes either \( \mathcal{L}_g \) or \( \mathcal{L}_t \). In other cases, the two losses might be jointly optimized.
In Chapter 2, we will introduce generative feature learning methods in the context of unsupervised pretraining:

- The neural architecture Transformer-XL advances the state of the art in language modeling. Transformer-XL enables capturing longer-range dependency and also resolves the context fragmentation problem. This makes it possible to truly unleash the potential of Transformers and improves language modeling performance for both long and short sequences.

- XLNet is an unsupervised pretraining framework that bridges the gap between language modeling and unsupervised pretraining. XLNet employs a permutation language modeling objective that provides more effective training signals. Moreover, with XLNet, it is now possible to translate language modeling progress into improvement on target tasks via unsupervised pretraining. For example, we show that Transformer-XL, the latest advance in language modeling, improves unsupervised pretraining performance. This also opens the possibility of applying language modeling progress to improve downstream tasks.

In Chapter 3, generative semi-supervised learning methods will be discussed:

- For GAN-based semi-supervised learning, we prove that under a widely-used discriminator objective in GAN-based semi-supervised learning, a preferred generator should instead generate complement data. We provide a theoretical framework to illustrate two points: (1) how a discriminator benefits from joint training with a GAN; (2) it is contradictory to have a good generative model and a good classifier. This framework provides more fundamental understanding about why and how GANs work for semi-supervised learning and improves performance empirically.

- Going beyond the standard semi-supervised learning formulation, we pioneered the research of generative feature learning for tasks with complex data structures. For semi-supervised question answering, we train a generative model to generate questions based on unlabeled text, and combine model-generated questions with human-generated questions for training question answering models. Domain adaptation techniques based on reinforcement learning are integrated into the training procedure to alleviate data distribution discrepancy. For semi-supervised classification on graphs, we define the unsupervised loss function as generating random walk paths on the graphs.

Overall, the approaches proposed in the thesis have played a leading role in the development of a variety of research topics. Specifically, we hold or used to gold the state-of-the-art results on more than 30 benchmarks, including natural language inference, question answering, text classification, language modeling, semi-supervised learning, etc.
Chapter 2

Unsupervised Pretraining with Generative Modeling

This chapter introduces our generative feature learning methods in the setting of unsupervised pretraining. We attempt to answer the following questions:

- Is it possible to truly unleash the potential of Transformers at capturing long-range dependency?
- Is it possible to combine the advantages of autoregressive and autoencoding approaches while avoid their disadvantages?
- Is it possible to bridge the gap between language modeling and pretraining, making it possible to leverage language modeling progress to improve target tasks?

Our approach decouples the problem of unsupervised pretraining into two subproblems: (1) improving language modeling; (2) building a learning paradigm with which progress in language modeling can be translated into benefits in unsupervised pretraining. Following this idea, in Section 2.1 we will first present our neural architecture Transformer-XL that substantially improves language modeling by capturing longer-range dependency and resolving the context fragmentation problem. In Section 2.2 a learning paradigm called XLNet will be introduced to bridge the gap between language modeling and unsupervised pretraining.

2.1 Attentive Language Models Beyond a Fixed-Length Context

2.1.1 Motivation

Language modeling is among the important problems that require modeling long-term dependency, with successful applications such as unsupervised pretraining [26, 32, 109, 111]. However, it has been a challenge to equip neural networks with the capability to model long-term dependency in sequential data. Recurrent neural networks (RNNs), in particular Long Short-Term Memory
(LSTM) networks [55], have been a standard solution to language modeling and obtained strong results on multiple benchmarks. Despite the wide adaption, RNNs are difficult to optimize due to gradient vanishing and explosion [56], and the introduction of gating in LSTMs and the gradient clipping technique [47] might not be sufficient to fully address this issue. Empirically, previous work has found that LSTM language models use 200 context words on average [68], indicating room for further improvement.

On the other hand, the direct connections between long-distance word pairs baked in attention mechanisms might ease optimization and enable the learning of long-term dependency [5, 145]. Recently, Al-Rfou et al. [1] designed a set of auxiliary losses to train deep Transformer networks for character-level language modeling, which outperform LSTMs by a large margin. Despite the success, the LM training in Al-Rfou et al. [1] is performed on separated fixed-length segments of a few hundred characters, without any information flow across segments. As a consequence of the fixed context length, the model cannot capture any longer-term dependency beyond the predefined context length. In addition, the fixed-length segments are created by selecting a consecutive chunk of symbols without respecting the sentence or any other semantic boundary. Hence, the model lacks necessary contextual information needed to well predict the first few symbols, leading to inefficient optimization and inferior performance. We refer to this problem as context fragmentation.

To address the aforementioned limitations of fixed-length contexts, we propose a new architecture called Transformer-XL (meaning extra long). We introduce the notion of recurrence into our deep self-attention network. In particular, instead of computing the hidden states from scratch for each new segment, we reuse the hidden states obtained in previous segments. The reused hidden states serve as memory for the current segment, which builds up a recurrent connection between the segments. As a result, modeling very long-term dependency becomes possible because information can be propagated through the recurrent connections. Meanwhile, passing information from the previous segment can also resolve the problem of context fragmentation. More importantly, we show the necessity of using relative positional encodings rather than absolute ones, in order to enable state reuse without causing temporal confusion. Hence, as an additional technical contribution, we introduce a simple but more effective relative positional encoding formulation that generalizes to attention lengths longer than the one observed during training.

2.1.2 Background

In the last few years, the field of language modeling has witnessed many significant advances, including but not limited to devising novel architectures to better encode the context [1, 10, 91, 95], improving regularization and optimization algorithms [38], speeding up the Softmax computation [45], and enriching the output distribution family [169].

To capture the long-range context in language modeling, a line of work directly feeds a representation of the wider context into the network as an additional input. Existing works range from ones where context representations are manually defined [61, 94, 151] to others that rely on document-level topics learned from data [34, 152].
Figure 2.1: Illustration of the vanilla model with a segment length 4.

More broadly, in generic sequence modeling, how to capture long-term dependency has been a long-standing research problem. From this perspective, since the ubiquitous adaption of LSTM, many efforts have been spent on relieving the vanishing gradient problem, including better initialization [80], additional loss signal [138], augmented memory structure [67] and others that modify the internal architecture of RNNs to ease the optimization [83, 159]. Different from them, our work is based on the Transformer architecture and shows that language modeling as a real-world task benefits from the ability to learn longer-term dependency.

2.1.3 Approach

Given a corpus of tokens $x = (x_1, \ldots, x_T)$, the task of language modeling is to estimate the joint probability $P(x)$, which is often auto-regressively factorized as $P(x) = \prod_t P(x_t | x_{<t})$. With the factorization, the problem reduces to estimating each conditional factor. In this work, we stick to the standard neural approach to modeling the conditional probability. Specifically, a trainable neural network is used to encode the context $x_{<t}$ into a fixed size hidden state, which is multiplied with the word embeddings to obtain the logits. The logits are then fed into the Softmax function, yielding a categorical probability distribution over the next token.

Vanilla Transformer Language Models

In order to apply Transformer or self-attention to language modeling, the central problem is how to train a Transformer to effectively encode an arbitrarily long context into a fixed size representation. Given infinite memory and computation, a simple solution would be to process the entire context sequence using an unconditional Transformer decoder, similar to a feed-forward neural network. However, this is usually infeasible with the limited resource in practice.

One feasible but crude approximation is to split the entire corpus into shorter segments of manageable sizes, and only train the model within each segment, ignoring all contextual information from previous segments. This is the idea adopted by Al-Rfou et al. [11]. We call it the vanilla model and visualize it in Fig. 2.1a. Under this training paradigm, information never flows across segments in either the forward or backward pass. There are two critical limitations of using a fixed-length context. First, the largest possible dependency length is upper bounded by the segment length, which is a few hundred on character-level language modeling [11]. Therefore, although the self-attention mechanism is less affected by the vanishing gradient problem compared
to RNNs, the vanilla model is not able to fully exploit this optimization advantage. Second, though it is possible to use padding to respect the sentence or other semantic boundaries, in practice it has been standard practice to simply chunk long text into fixed-length segments due to improved efficiency \[1, 32, 109\]. However, simply chunking a sequence into fixed-length segments will lead to the context fragmentation problem as discussed in Section \[2.1.1\].

During evaluation, at each step, the vanilla model also consumes a segment of the same length as in training, but only makes one prediction at the last position. Then, at the next step, the segment is shifted to the right by only one position, and the new segment has to be processed all from scratch. As shown in Fig. 2.1b, this procedure ensures that each prediction utilizes the longest possible context exposed during training, and also relieves context fragmentation issue encountered in training. However, this evaluation procedure is extremely expensive. We will show that our proposed architecture is able to substantially improve the evaluation speed.

**Segment-Level Recurrence with State Reuse**

![Diagram of the Transformer-XL model with a segment length 4.](image)

(a) Training phase.

(b) Evaluation phase.

**Figure 2.2:** Illustration of the Transformer-XL model with a segment length 4.

To address the limitations of using a fixed-length context, we propose to introduce a recurrence mechanism to the Transformer architecture. During training, the hidden state sequence computed for the previous segment is fixed and cached to be reused as an extended context when the model processes the next new segment, as shown in Fig. 2.2a. Although the gradient still remains within a segment, this additional input allows the network to exploit information in the history, leading to an ability of modeling longer-term dependency and avoiding context fragmentation. Formally, let the two consecutive segments of length \(L\) be \(s_\tau = [x_\tau, 1, \cdots, x_\tau, L]\) and \(s_{\tau+1} = [x_{\tau+1}, 1, \cdots, x_{\tau+1}, L]\) respectively. Denoting the \(n\)-th layer hidden state sequence produced for the \(\tau\)-th segment \(s_\tau\) by \(h^n_\tau \in \mathbb{R}^{L \times d}\), where \(d\) is the hidden dimension. Then, the \(n\)-th layer hidden state for segment \(s_{\tau+1}\) is produced (schematically) as follows,

\[
\tilde{h}^{n-1}_{\tau+1} = [\text{SG}(h^{n-1}_\tau) \circ h^{n-1}_{\tau+1}],
\]

\[
q^n_{\tau+1}, k^n_{\tau+1}, v^n_{\tau+1} = h^{n-1}_{\tau+1} W^q, \tilde{h}^{n-1}_{\tau+1} W^k, \tilde{h}^{n-1}_{\tau+1} W^v,
\]

\[
h^n_{\tau+1} = \text{Transformer-Layer} \left( q^n_{\tau+1}, k^n_{\tau+1}, v^n_{\tau+1} \right).
\]

where the function \(\text{SG}(\cdot)\) stands for stop-gradient, the notation \([h_u \circ h_v]\) indicates the concatenation of two hidden sequences along the length dimension, and \(W\) denotes model parameters. Compared to the standard Transformer, the critical difference lies in that the key \(k^n_{\tau+1}\) and value \(v^n_{\tau+1}\) are conditioned on the extended context \(\tilde{h}^{n-1}_{\tau+1}\) and hence \(h^{n-1}_{\tau+1}\) cached from the previous segment. We emphasize this particular design by the green paths in Fig. 2.2a.
With this recurrence mechanism applied to every two consecutive segments of a corpus, it essentially creates a segment-level recurrence in the hidden states. As a result, the effective context being utilized can go way beyond just two segments. However, notice that the recurrent dependency between $h^n_{\tau+1}$ and $h^{n-1}_\tau$ shifts one layer downwards per-segment, which differs from the same-layer recurrence in conventional RNN-LMs. Consequently, the largest possible dependency length grows linearly w.r.t. the number of layers as well as the segment length, i.e., $O(N \times L)$, as visualized by the shaded area in Fig. 2.2b. This is analogous to truncated BPTT [95], a technique developed for training RNN-LMs. However, different from truncated BPTT, our method caches a sequence of hidden states instead of the last one, and should be applied together with the relative positional encoding technique described in Section 2.1.3.

Besides achieving extra long context and resolving fragmentation, another benefit that comes with the recurrence scheme is significantly faster evaluation. Specifically, during evaluation, the representations from the previous segments can be reused instead of being computed from scratch as in the case of the vanilla model. In our experiments on enwiki8, Transformer-XL is up to 1,800+ times faster than the vanilla model during evaluation (see Section 3.3.4).

Finally, notice that the recurrence scheme does not need to be restricted to only the previous segment. In theory, we can cache as many previous segments as the GPU memory allows, and reuse all of them as the extra context when processing the current segment. Thus, we can cache a predefined length-$M$ old hidden states spanning (possibly) multiple segments, and refer to them as the memory $m^n \in \mathbb{R}^{M \times d}$, due to a clear connection to the memory augmented neural networks [48, 156]. In our experiments, we set $M$ equal to the segment length during training, and increase it by multiple times during evaluation.

Relative Positional Encodings

While we found the idea presented in the previous subsection very appealing, there is a crucial technical challenge we haven’t solved in order to reuse the hidden states. That is, how can we keep the positional information coherent when we reuse the states? Recall that, in the standard Transformer, the information of sequence order is provided by a set of positional encodings, denoted as $U \in \mathbb{R}^{L_{\text{max}} \times d}$, where the $i$-th row $U_i$ corresponds to the $i$-th absolute position within a segment and $L_{\text{max}}$ prescribes the maximum possible length to be modeled. Then, the actual input to the Transformer is the element-wise addition of the word embeddings and the positional encodings. If we simply adapt this positional encoding to our recurrence mechanism, the hidden state sequence would be computed schematically by

$$h_{\tau+1} = f(h_{\tau}, E_{s_{\tau+1}} + U_{1:L})$$
$$h_{\tau} = f(h_{\tau-1}, E_{s_{\tau}} + U_{1:L}),$$

where $E_{s_{\tau}} \in \mathbb{R}^{L \times d}$ is the word embedding sequence of $s_{\tau}$, and $f$ represents a transformation function. Notice that, both $E_{s_{\tau}}$ and $E_{s_{\tau+1}}$ are associated with the same positional encoding $U_{1:L}$. As a result, the model has no information to distinguish the positional difference between $x_{\tau,j}$ and $x_{\tau+1,j}$ for any $j = 1, \ldots, L$, resulting in a sheer performance loss.

In order to avoid this failure mode, the fundamental idea is to only encode the relative positional information in the hidden states. Conceptually, the positional encoding gives the model
a temporal clue or “bias” about how information should be gathered, i.e., where to attend. For the same purpose, instead of incorporating bias statically into the initial embedding, one can inject the same information into the attention score of each layer. More importantly, it is more intuitive and generalizable to define the temporal bias in a relative manner. For instance, when a query vector \( q_{τ,i} \) attends on the key vectors \( k_{τ,≤i} \), it does not need to know the absolute position of each key vector to identify the temporal order of the segment. Instead, it suffices to know the relative distance between each key vector \( k_{τ,j} \) and itself \( q_{τ,i} \), i.e., \( i − j \). Practically, one can create a set of relative positional encodings \( R \in \mathbb{R}^{L_{\text{max}} \times d} \), where the \( i \)-th row \( R_i \) indicates a relative distance of \( i \) between two positions. By injecting the relative distance dynamically into the attention score, the query vector can easily distinguish the representations of \( x_{τ,j} \) and \( x_{τ+1,j} \) from their different distances, making the state reuse mechanism feasible. Meanwhile, we won’t lose any temporal information, as the absolute position can be recovered recursively from relative distances.

Previously, the idea of relative positional encodings has been explored in the context of machine translation [124] and music generation [58]. Here, we offer a different derivation, arriving at a new form of relative positional encodings, which not only has a one-to-one correspondence to its absolute counterpart but also enjoys much better generalization empirically (see Section 3.3.4). Firstly, in the standard Transformer [145], the attention score between query \( q_i \) and key vector \( k_j \) within the same segment can be decomposed as

\[
A_{i,j}^{\text{abs}} = E_{x_i}^T W_q^T W_k E_{x_j} + E_{x_i}^T W_q^T W_k U_j + U_i^T W_q^T W_k E_{x_j} + U_i^T W_q^T W_k U_j.
\]

Following the idea of only relying on relative positional information, we propose to reparameterize the four terms as follows

\[
A_{i,j}^{\text{rel}} = E_{x_i}^T W_q^T W_{k,E} E_{x_j} + E_{x_i}^T W_q^T W_{k,R} R_{i−j} + U_i^T W_q^T W_{k,E} E_{x_j} + U_i^T W_q^T W_{k,R} R_{i−j}.
\]

• The first change we make is to replace all appearances of the absolute positional embedding \( U_j \) for computing key vectors in term \((b)\) and \((d)\) with its relative counterpart \( R_{i−j} \). This essentially reflects the prior that only the relative distance matters for where to attend. Note that \( R \) is a sinusoid encoding matrix [145] without learnable parameters.

• Secondly, we introduce a trainable parameter \( u \in \mathbb{R}^d \) to replace the query \( U_i^T W_q^T \) in term \((c)\). In this case, since the query vector is the same for all query positions, it suggests that the attentive bias towards different words should remain the same regardless of the query position. With a similar reasoning, a trainable parameter \( v \in \mathbb{R}^d \) is added to substitute \( U_i^T W_q^T \) in term \((d)\).

• Finally, we deliberately separate the two weight matrices \( W_{k,E} \) and \( W_{k,R} \) for producing the content-based key vectors and location-based key vectors respectively.
Under the new parameterization, each term has an intuitive meaning: term \((a)\) represents content-based addressing, term \((b)\) captures a content-dependent positional bias, term \((c)\) governs a global content bias, and \((d)\) encodes a global positional bias.

In comparison, the formulation in Shaw et al. [124] only has terms \((a)\) and \((b)\), dropping the two bias terms \((c)\) and \((d)\). Moreover, Shaw et al. [124] merge the multiplication \(W_k R\) into a single trainable matrix \(\hat{R}\), which abandons the inductive bias built into the original sinusoid positional encoding [145]. In contrast, our relative positional embedding \(R\) adapts the sinusoid formulation. As a benefit of the inductive bias, a model trained on a memory of some certain length can automatically generalize to a memory several times longer during evaluation.

Equipping the recurrence mechanism with our proposed relative positional embedding, we finally arrive at the Transformer-XL architecture. For completeness, we summarize the computational procedure for a \(N\)-layer Transformer-XL with a single attention head here. For \(n = 1, \ldots, N:\)

\[
\begin{align*}
\tilde{h}_{\tau, n-1} &= \left[\text{SG}(m_{\tau, n-1}) \circ h_{\tau, n-1}\right] \\
q_n^\tau, k_n^\tau, v_n^\tau &= h_{\tau, n-1}^\top W_q^\top, \tilde{h}_{\tau, n-1}^\top W_E^\top, \tilde{h}_{\tau, n-1}^\top W_v^\top \\
A_n^\tau_{i,j} &= q_n^\tau_i k_n^\tau_j + q_n^\tau_i W_k^\top W_{R_{i-j}} \\
&\quad + u^\top k_{\tau,j} + v^\top W_{R_{i-j}} \\
a_n^\tau &= \text{Masked-Softmax}(A_n^\tau)v_n^\tau \\
o_n^\tau &= \text{LayerNorm}(\text{Linear}(a_n^\tau) + h_{\tau, n-1}) \\
h_{\tau, n}^\tau &= \text{Positionwise-Feed-Forward}(o_n^\tau)
\end{align*}
\]

with \(h_{\tau, 0}^0 := E_{s, \tau}\) defined as the word embedding sequence. In addition, it is worth mentioning that a naive way to compute \(A\) requires computing \(W_{k,R_{i-j}}\) for all pairs \((i, j)\), whose cost is quadratic w.r.t. the sequence length. However, noticing that the value of \(i - j\) only ranges from zero to the sequence length, we show a simple computation procedure in Section 2.1.4, which reduces the cost to be linear w.r.t. the sequence length.

### 2.1.4 Experiments

#### Main Results

We apply Transformer-XL to a variety of datasets on both word-level and character-level language modeling to have a comparison with state-of-the-art systems, including WikiText-103 [91], enwik8 [88], text8 [88], One Billion Word [17], and Penn Treebank [94].

WikiText-103 is the largest available word-level language modeling benchmark with long-term dependency. It contains 103M training tokens from 28K articles, with an average length of 3.6K tokens per article, which allows testing the ability of long-term dependency modeling. We set the attention length to 384 during training and 1600 during evaluation. We adopted adaptive softmax and input representations [4, 45]. As shown in Table 2.1, Transformer-XL reduces the previous state-of-the-art (SoTA) perplexity from 20.5 to 18.3, which demonstrates the superiority of the Transformer-XL architecture.
Table 2.1: Comparison with state-of-the-art results on WikiText-103. \(^\circ\) indicates contemporary work.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grave et al. [46] - LSTM</td>
<td>-</td>
<td>48.7</td>
</tr>
<tr>
<td>Bai et al. [6] - TCN</td>
<td>-</td>
<td>45.2</td>
</tr>
<tr>
<td>Dauphin et al. [31] - GCNN-8</td>
<td>-</td>
<td>44.9</td>
</tr>
<tr>
<td>Grave et al. [46] - Neural cache</td>
<td>-</td>
<td>40.8</td>
</tr>
<tr>
<td>Dauphin et al. [31] - GCNN-14</td>
<td>-</td>
<td>37.2</td>
</tr>
<tr>
<td>Merity et al. [93] - QRNN</td>
<td>151M</td>
<td>33.0</td>
</tr>
<tr>
<td>Rae et al. [112] - Hebbian + Cache</td>
<td>-</td>
<td>29.9</td>
</tr>
<tr>
<td>Ours - Transformer-XL Standard</td>
<td>151M</td>
<td>24.0</td>
</tr>
<tr>
<td>Ours - Transformer-XL Large</td>
<td>257M</td>
<td>18.3</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison with state-of-the-art results on enwik8.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>bpc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ha et al. [53] - LN HyperNetworks</td>
<td>27M</td>
<td>1.34</td>
</tr>
<tr>
<td>Chung et al. [21] - LN HM-LSTM</td>
<td>35M</td>
<td>1.32</td>
</tr>
<tr>
<td>Zilly et al. [181] - RHN</td>
<td>46M</td>
<td>1.27</td>
</tr>
<tr>
<td>Mujika et al. [100] - FS-LSTM-4</td>
<td>47M</td>
<td>1.25</td>
</tr>
<tr>
<td>Krause et al. [74] - Large mLSTM</td>
<td>46M</td>
<td>1.24</td>
</tr>
<tr>
<td>Knol [72] - cmix v13</td>
<td>-</td>
<td>1.23</td>
</tr>
<tr>
<td>Al-Rfou et al. [1] - 12L Transformer</td>
<td>44M</td>
<td>1.11</td>
</tr>
<tr>
<td>Ours - 12L Transformer-XL</td>
<td>41M</td>
<td>1.06</td>
</tr>
<tr>
<td>Al-Rfou et al. [1] - 64L Transformer</td>
<td>235M</td>
<td>1.06</td>
</tr>
<tr>
<td>Ours - 18L Transformer-XL</td>
<td>88M</td>
<td>1.03</td>
</tr>
<tr>
<td>Ours - 24L Transformer-XL</td>
<td>277M</td>
<td>0.99</td>
</tr>
</tbody>
</table>

The dataset enwik8 contains 100M bytes of unprocessed Wikipedia text. We compare our architecture with the previous results in Table 2.2. Under the model size constraint, the 12-layer Transformer-XL achieves a new SoTA result, outperforming the 12-layer vanilla Transformer from Al-Rfou et al. [1] by 0.05, while both Transformer variants have a large margin over conventional RNN-based models. Notably, our 12-layer architecture achieves the same result as the 64-layer network from Al-Rfou et al. [1], using only 17% of the parameter budget. In order to see whether better performances can be obtained by increasing the model size, we train 18-layer and 24-layer Transformer-XLs with increased model sizes. With the attention length 784 during training and 3,800 during evaluation, we obtained a new SoTA result and our method is the first to break through 1.0 on widely-studied character-level benchmarks. Different from Al-Rfou et al. [1], Transformer-XL does not need any auxiliary losses, and thus all benefits are credited to a better architecture.

Similar to but different from enwik8, text8 contains 100M processed Wikipedia characters.
<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>bpc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooijmans et al. [23] - BN-LSTM</td>
<td>-</td>
<td>1.36</td>
</tr>
<tr>
<td>Chung et al. [21] - LN HM-LSTM</td>
<td>35M</td>
<td>1.29</td>
</tr>
<tr>
<td>Zilly et al. [181] - RHN</td>
<td>45M</td>
<td>1.27</td>
</tr>
<tr>
<td>Krause et al. [74] - Large mLSTM</td>
<td>45M</td>
<td>1.27</td>
</tr>
<tr>
<td>Al-Rfou et al. [1] - 12L Transformer</td>
<td>44M</td>
<td>1.18</td>
</tr>
<tr>
<td>Al-Rfou et al. [1] - 64L Transformer</td>
<td>235M</td>
<td>1.13</td>
</tr>
<tr>
<td>Ours - 24L Transformer-XL</td>
<td>277M</td>
<td>1.08</td>
</tr>
</tbody>
</table>

Table 2.3: Comparison with state-of-the-art results on text8.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shazeer et al. [125] - Sparse Non-Negative</td>
<td>33B</td>
<td>52.9</td>
</tr>
<tr>
<td>Chelba et al. [17] - RNN-1024 + 9 Gram</td>
<td>20B</td>
<td>51.3</td>
</tr>
<tr>
<td>Kuchaiev and Ginsburg [75] - G-LSTM-2</td>
<td>-</td>
<td>36.0</td>
</tr>
<tr>
<td>Dauphin et al. [31] - GCNN-14 bottleneck</td>
<td>-</td>
<td>31.9</td>
</tr>
<tr>
<td>Jozefowicz et al. [65] - LSTM</td>
<td>1.8B</td>
<td>30.6</td>
</tr>
<tr>
<td>Jozefowicz et al. [65] - LSTM + CNN</td>
<td>1.04B</td>
<td>30.0</td>
</tr>
<tr>
<td>Shazeer et al. [126] - Low-Budget MoE</td>
<td>~5B</td>
<td>34.1</td>
</tr>
<tr>
<td>Shazeer et al. [126] - High-Budget MoE</td>
<td>~5B</td>
<td>28.0</td>
</tr>
<tr>
<td>Shazeer et al. [127] - Mesh Tensorflow</td>
<td>4.9B</td>
<td>24.0</td>
</tr>
<tr>
<td>Baevski and Auli [4] - Adaptive Input°</td>
<td>0.46B</td>
<td>24.1</td>
</tr>
<tr>
<td>Ours - Transformer-XL Base</td>
<td>0.46B</td>
<td>23.5</td>
</tr>
<tr>
<td>Ours - Transformer-XL Large</td>
<td>0.8B</td>
<td>21.8</td>
</tr>
</tbody>
</table>

Table 2.4: Comparison with state-of-the-art results on One Billion Word. ° indicates contemporary work.

created by lowering case the text and removing any character other than the 26 letters a through z, and space. Due to the similarity, we simply adapt the best model and the same hyper-parameters on enwik8 to text8 without further tuning. The comparison with previous methods is summarized in Table 2.3. Again, Transformer-XL achieves the new SoTA result with a clear margin.

One Billion Word does not preserve any long-term dependency because sentences have been shuffled. Consequently, this dataset mainly tests the ability of modeling only short-term dependency. The comparison between Transformer-XL and the other methods is shown in Table 2.4. Although Transformer-XL is mainly designed to better capture longer-term dependency, it dramatically improves the single-model SoTA from 23.7 to 21.8. Specifically, Transformer-XL significantly outperforms a contemporary method using vanilla Transformers [4], suggesting the advantage of Transformer-XL is generalizable to modeling short sequences.

We also report the results on word-level Penn Treebank in Table 2.5. Similar to AWD-LSTM [92], we apply variational dropout and weight average to Transformer-XL. With proper
Table 2.5: Comparison with state-of-the-art results on Penn Treebank. † indicates using two-step finetuning.

regularization, Transformer-XL achieves a new SoTA result among models without two-step finetuning. Penn Treebank has only 1M training tokens, which implies that Transformer-XL also generalizes well even on small datasets.

**Ablation Study**

We conduct two sets of ablation studies to examine the effects of two proposed techniques used in Transformer-XL: the recurrence mechanism and the new positional encoding scheme.

The first study is performed on WikiText-103, which requires modeling long-term dependency. The results are reported in Table 2.6. Among the compared encoding schemes, Shaw et al. [124] is relative, while Vaswani et al. [145] and Al-Rfou et al. [1] are absolute. “Full” and “half” losses refer to applying a cross entropy loss to all or the recent half positions in the segment. We found that absolute encodings only work well with half losses because half losses exclude positions with very short attention lengths during training for better generalization. Table 2.6 shows that both the recurrence mechanism and our encoding scheme are necessary to achieve the best performance, as well as generalizing to longer attention sequences during evaluation time. Although the backpropagation length during training is only 128, with the two techniques the attention length can be increased to 640 at test time. In the standard setting with 151M parameters, the perplexity decreases as the attention length increases.

Since the recurrence mechanism costs additional memory, we also compare Transformer-XL with baselines under the same GPU memory constraints. As shown in Table 2.8, despite using a shorter backpropagation length, Transformer-XL remains superior to the baselines.

The second study targets at isolating the effects of resolving the context fragmentation problem from the benefit of capturing longer context length. In order to achieve this goal, we deliberately choose a dataset that does not require long-term dependency, so that any improvement from establishing the recurrence can be attributed to solving the context fragmentation. Specifically, we
Table 2.6: Ablation study on WikiText-103. For the first two blocks, we use a slightly smaller model (128M parameters). † indicates that the corresponding row is reduced to the same setting as the Transformer network in [1], except that two auxiliary losses are not implemented in our experiments. “PPL init” refers to using the same length as training. “PPL best” indicates the perplexity obtained by using the optimal length. “Attn Len” is the shortest possible attention length during evaluation to achieve the corresponding result (PPL best). Increasing the attention length during evaluation improves performance only when our positional encoding is used. The “Transformer-XL (151M)” setting uses a standard parameter budget as previous work [93], where we observe a similar effect when increasing the attention length during evaluation.

**Relative Effective Context Length**

Khandelwal et al. [68] proposed a method to evaluate the Effective Context Length (ECL) of a sequence model. ECL is the longest length to which increasing the context span would lead to a gain more than a threshold. However, ECL ignores the fact that it is harder to get improvement when a model already achieves a lower perplexity using only a shorter context, and thus it is not suitable for fair comparison among multiple models. We instead propose a new metric called Relative Effective Context Length (RECL). RECL is defined on a model group instead of a single model, and the gain of a long context is measure by the relative improvement over the best short context model. As such, the model group shares the same baseline to enable fair comparison. RECL also has a parameter $r$, which means constraining the comparison on top-$r$ hard examples.
Table 2.7: Ablation study on One Billion Word, a dataset without long-term dependency.

<table>
<thead>
<tr>
<th>Backprop Len</th>
<th>Recurrence</th>
<th>Encoding</th>
<th>Loss</th>
<th>pplx best</th>
<th>pplx init</th>
<th>Attn Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>✓</td>
<td>Ours</td>
<td>Full</td>
<td>26.77</td>
<td>27.02</td>
<td>500</td>
</tr>
<tr>
<td>128</td>
<td>✓</td>
<td>Ours</td>
<td>Partial</td>
<td>28.33</td>
<td>28.69</td>
<td>460</td>
</tr>
<tr>
<td>176</td>
<td>✗</td>
<td>Ours</td>
<td>Full</td>
<td>27.98</td>
<td>28.43</td>
<td>400</td>
</tr>
<tr>
<td>172</td>
<td>✗</td>
<td>Ours</td>
<td>Partial</td>
<td>28.83</td>
<td>28.83</td>
<td>120</td>
</tr>
</tbody>
</table>

Table 2.8: Ablation study on WikiText-103 with the same GPU memory constraints.

See Appendix 2.1.4 for more details about RECL. As shown in Table 2.9, Transformer-XL manages to model dependency of 900 words long on average with \( r = 0.1 \). The RECL of Transformer-XL is 80% and 450% longer than recurrent networks and Transformer respectively. Both the recurrence mechanism and our positional encodings contribute to a longer RECL. This further substantiates our argument that Transformer-XL is able to model longer-term dependency.

**Evaluation Speed**

Finally, we compare the evaluation speed of our model with the vanilla Transformer model [1]. As shown in Table 2.10, due to the state reuse scheme, Transformer-XL achieves an up to 1,874 times speedup during evaluation.

**Efficient Computation of the Attention with Relative Positional Embedding**

As we discussed in Section 2.1.3, the naive way of computing the \( W_{k,R}R_{k-i-j} \) for all pairs \((i, j)\) is subject to a quadratic cost. Here, we present a simple method with only a linear cost. Firstly, notice that the relative distance \( i - j \) can only be integer from 0 to \( M + L - 1 \), where \( M \) and \( L \) are the memory length and segment length respectively. Hence, the rows of the matrix

\[
Q := \begin{bmatrix}
R_{M+L-1}^T \\
R_{M+L-2}^T \\
\vdots \\
R_{M}^T \\
R_{0}^T 
\end{bmatrix} \quad W_{k,R}^T = \begin{bmatrix}
[ W_{k,R}R_{M+L-1}]^T \\
[ W_{k,R}R_{M+L-2}]^T \\
\vdots \\
[ W_{k,R}R_{1}]^T \\
[ W_{k,R}R_{0}]^T 
\end{bmatrix} \in \mathbb{R}^{(M+L)\times d}
\]

consist of all possible vector outputs of \( W_{k,R}R_{k-i-j} \) for any \((i, j)\). Note that we have defined \( Q \) in a reversed order, i.e., \( Q_k = W_{k,R}R_{M+L-1-k} \), to make further discussion easier.
July 22, 2019
DRAFT

<table>
<thead>
<tr>
<th>Model</th>
<th>r = 0.1</th>
<th>r = 0.5</th>
<th>r = 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-XL 151M</td>
<td>900</td>
<td>800</td>
<td>700</td>
</tr>
<tr>
<td>QRNN</td>
<td>500</td>
<td>400</td>
<td>300</td>
</tr>
<tr>
<td>LSTM</td>
<td>400</td>
<td>300</td>
<td>200</td>
</tr>
<tr>
<td>Transformer-XL 128M</td>
<td>700</td>
<td>600</td>
<td>500</td>
</tr>
<tr>
<td>- use Shaw et al. [124] encoding</td>
<td>400</td>
<td>400</td>
<td>300</td>
</tr>
<tr>
<td>- remove recurrence</td>
<td>300</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Transformer</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

Table 2.9: Relative effective context length (RECL) comparison. See text for the definition of RECL and r. The first three models and the last four models are compared as two *model groups* when we calculate RECL (RECL is computed on a model group rather than a single model). Each group has the same parameter budget.

<table>
<thead>
<tr>
<th>Attn Len</th>
<th>How much Al-Rfou et al. [1] is slower</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,800</td>
<td>1,874x</td>
</tr>
<tr>
<td>2,800</td>
<td>1,409x</td>
</tr>
<tr>
<td>1,800</td>
<td>773x</td>
</tr>
<tr>
<td>800</td>
<td>363x</td>
</tr>
</tbody>
</table>

Table 2.10: Slowdown in terms of running time during evaluation. Evaluation is based on per-token time on one GPU.

Next, we collect the term \((b)\) for all possible \(i, j\) into the following \(L \times (M + L)\) matrix,

\[
B = \begin{bmatrix}
q_0^T W_{k,R} R_M & \cdots & q_0^T W_{k,R} R_0 & 0 & \cdots & 0 \\
q_1^T W_{k,R} R_{M+1} & \cdots & q_1^T W_{k,R} R_1 & q_1^T W_{k,R} R_0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
q_{L-1}^T W_{k,R} R_{M+L-1} & \cdots & q_{L-1}^T W_{k,R} R_1 & q_{L-1}^T W_{k,R} R_0 & \cdots & q_{L-1}^T W_{k,R} R_0 \\
q_0^T Q_{L-1} & \cdots & q_0^T Q_{M+L-1} & 0 & \cdots & 0 \\
q_1^T Q_{L-2} & \cdots & q_1^T Q_{M+L-2} & q_1^T Q_{M+L-1} & \cdots & 0 \\
\vdots & \ddots & \vdots & \ddots & \ddots & \ddots \\
q_{L-1}^T Q_0 & \cdots & q_{L-1}^T Q_M & q_{L-1}^T Q_{M+1} & \cdots & q_{L-1}^T Q_{M+L-1}
\end{bmatrix}
\]

Then, we further define

\[
\tilde{B} = qQ^T = \begin{bmatrix}
q_0^T Q_0 & \cdots & q_0^T Q_M & q_0^T Q_{M+1} & \cdots & q_0^T Q_{M+L-1} \\
q_1^T Q_0 & \cdots & q_1^T Q_M & q_1^T Q_{M+1} & \cdots & q_1^T Q_{M+L-1} \\
\vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\
q_{L-1}^T Q_0 & \cdots & q_{L-1}^T Q_M & q_{L-1}^T Q_{M+1} & \cdots & q_{L-1}^T Q_{M+L-1}
\end{bmatrix}
\]

Now, it is easy to see an immediate relationship between \(B\) and \(\tilde{B}\), where the \(i\)-th row of \(B\) is
simply a left-shifted version of \(i\)-th row of \(\widetilde{B}\). Hence, the computation of \(B\) only requires a matrix multiplication \(qQ^\top\) to compute \(\widetilde{B}\) and then a set of left-shifts.

Similarly, we can collect all term \((d)\) for all possible \(i,j\) into another \(L \times (M + L)\) matrix \(D\),

\[
D = \begin{bmatrix}
v^\top Q_{L-1} & \cdots & v^\top Q_{M+L-1} & 0 & \cdots & 0 \\
v^\top Q_{L-2} & \cdots & v^\top Q_{M+L-2} & v^\top Q_{M+L-1} & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
v^\top Q_0 & \cdots & v^\top Q_M & v^\top Q_{M+1} & \cdots & v^\top Q_{M+L-1}
\end{bmatrix}.
\]

Then, we can follow the same procedure to define

\[
\widetilde{d} = [Qv]^\top = [v^\top Q_0 \cdots v^\top Q_M v^\top Q_{M+1} \cdots v^\top Q_{M+L-1}].
\]

Again, each row of \(D\) is simply a left-shift version of \(\widetilde{d}\). Hence, the main computation cost comes from the matrix-vector multiplication \(\widetilde{d} = [Qv]^\top\), which is not expensive any more.

**Details About RECL**

In this section, we describe the details of the metric RECL. Let \(\mathcal{M} = \{m_1, m_2, \cdots, m_N\}\) be a model group consisting of \(N\) models. Let \(l_i(c, t)\) denote the loss of model \(m_i\) on the \(t\)-th token in the corpus with a context length \(c\). Concretely, the loss can be written as

\[
l_i(c, t) = -\log P_{m_i}(x_t|x_{t-1}, \cdots, x_{t-c})
\]

where \(P_{m_i}\) is the probability distribution given by model \(m_i\), and \(x_t\) is the \(t\)-th token in the corpus. Given a short context length \(c\) and a long context length \(c'\) such that \(c' \geq c\), we can further define a baseline for each position \(t\),

\[
b(c, t) = \min_{i=1}^N l_i(c, t)
\]

The relative loss of \(m_i\) w.r.t. the model group \(\mathcal{M}\) is written as

\[
f_i(c, c') = \frac{1}{|T|} \sum_{t \in T} \min (b(c, t), l_i(c', t))
\]

![Figure 2.3: Visualizing unnormalized relative perplexity gains with \(r = 0.1\).](image)
The above equation uses the minimum loss of all models on the short length \( c \) as a baseline, and only losses smaller than the baseline will be effectively counted towards the relative loss. This enables fair comparison between multiple models because all models with a long context length \( c' \) need to improve over the same baseline. Sometimes we only care about those positions where the baseline performs poorly (which means short-term dependency with context length \( c \) is not sufficient), so given a ratio parameter \( r \), we define the set \( T \) is the above equation as

\[
T = \text{top-} r \text{ positions } t \text{ with largest } b(c, t)
\]

The relative gain is subsequently defined as the relative perplexity reduction:

\[
g_i(c, c') = \frac{\exp f_i(c, c) - \exp f_i(c, c')}{\exp f_i(c, c)}
\]

Given a step size \( \Delta \), we then use an algorithm to find the RECL by thresholding the relative gain:

1. Set initial short context length \( c \), and long context length \( c' = c + \Delta \)
2. Compute \( g_i(c, c') \). If \( g_i(c, c') < 0.01 \), return \( \text{RECL} = c \). If \( g_i(c, c') \geq 0.01 \), set \( c = c', c' = c + \Delta \) and go to step 1.

In Figure 2.3 we visualize the unnormalized relative perplexity gains \( \exp f_i(c, c) - \exp f_i(c, c') \) with various pairs of \((c, c')\) when \( r = 0.1 \). It is clear that Transformer-XL has a longer RECL compared to RNNs and other baselines because the relative gains are substantially larger.

For reference, we plot the perplexities with varying context lengths in Figure 2.4. The y-axis denotes the “normal” perplexity (not calibrated by baselines).

**Generated Text**

In this section, we present some generated text from our best model trained on the Wikitext-103 dataset. Trained only on WikiText-103 which is medium-sized, Transformer-XL is already able to generate relatively coherent articles with thousands of tokens without manual cherry picking,
despite minor flaws. We seed our Transformer-XL with a context of at most 512 consecutive tokens randomly sampled from the test set of Wikitext-103. Then, we run Transformer-XL to generate a \textit{pre-defined} number of tokens (500 or 1,000 in our case). For each generation step, we first find the top-40 probabilities of the next-step distribution and sample from top-40 tokens based on the re-normalized distribution. To help reading, we detokenize the context, the generated text and the reference text. Three generated examples are shown in Tables 2.11, 2.12, and 2.13. Note that we do not perform any cherry picking and present the first three examples we generate in the paper. In the text, “\text{= text =}”, “\text{= = text = =}” and “\text{= = = text = = =}” denote the Wikipedia page tile, section title and subsection title, respectively, due to the original data preprocessing procedure of Wikitext-103 [91].

As we can see, though only trained on 100M tokens, Transformer-XL is a strong model at generating long text articles, particularly in the following aspects:

- Transformer-XL is able to structurally maintain the sectional arrangement of Wikipedia.
- Transformer-XL manages to semantically stay on the same topic throughout the course of generation.
- Long-range references are common in the generated text.
- Transformer-XL often generates novel content that is not present in the training data.

For more detailed explanation of the interesting observations in each example, please refer to the corresponding caption.

Despite the overall excellence of the generation quality, the model can only perceive the seed context and hallucinate what to generate based on the limited knowledge (100M tokens only) it is trained on. As a result, the generated text sometimes looks clearly relevant but not close enough or to the point compared to what human writer would do. That said, we believe this issue is mostly a problem of limited training data size and could be alleviated by using a larger training set.
Kershaw started the 2010 season by posting a 3.07 ERA in April, but did so by walking 22 batters in 29 innings. On May 4, he had his worst start of his career against the Milwaukee Brewers at Dodger Stadium, throwing just 57 pitches in 11/3 innings, while retiring only four of the 13 batters he faced — including the pitcher. He was booed loudly upon being pulled from the game. Kershaw said after the game, "I didn’t give our team any kind of chance. It’s just not a good feeling to let your teammates down, let everybody down. It stings, it hurts. I’ve got to figure things out." Kershaw rebounded his next start by pitching an 8 inning two-hitter and out-dueling the then undefeated Ubaldo Jiménez. He credited his control of the slider being the major turning point for him. Later in the season, he was suspended for five games after hitting Aaron Rowand of the Giants with a pitch in a game on July 20. The incident occurred after both teams were given a warning following Giants ace Tim Lincecum hitting Matt Kemp earlier in the game. He threw his first career complete game shutout on September 14, 2010 also against San Francisco and finished the season with a record of 13 – 10 and a 2.91 ERA in 32 starts, pitching 2041/3 innings and recording 212 strikeouts.

Reference:

After finishing the 2010 season strong, the Dodgers named Kershaw as the Opening Day Starter for the 2011 season. On May 29, he pitched the second complete-game shutout of his career, striking out 10 while winning a two-hitter against the Florida Marlins, 8–0; he also had two singles and an RBI, scoring twice in the game. He produced his third career shutout on June 20, a two-hit, 11-strikeout effort against the Detroit Tigers. Kershaw became the first Dodgers starter to strike out the side in the 9th inning since Sandy Koufax’s perfect game. In his next start, on June 26, Kershaw pitched another complete game (against the Los Angeles Angels of Anaheim). He became the first Dodger starter to have back-to-back complete game victories since Jeff Weaver in the 2005 season and the first Dodger to have double-digit strikeouts in consecutive starts since Chan-Ho Park in the 2000 season. He was awarded the National League Player of the Week award for the week of June 20 – 26 as a result of those two starts. Midway through June, Kershaw had amassed 32 career victories.

XL Generation:

The Dodgers signed Kershaw to a one-year deal on July 14, 2011, with a player option for 2012, in a deal worth just under $300,000. The Dodgers entered the 2011 season as the major league leaders, and Kershaw had the best record in the National League in saves (39), games started (78) and walks (120); he also had the lowest win/loss ratio in the league (1.87) and he had the worst on-base percentage (.349). He had the best ERA (2.00), most wins (16), or most strikeouts (29) in his career (25), while leading the league in losses per nine innings pitched (4.01). Following the 2011 season, the Dodgers traded Kershaw to the Kansas City Royals on July 29 for pitcher Javier Torres.

During spring training, Kershaw played very well. He was selected to spring training as a relief pitcher for the Royals for the 2012 season. After an injury to closer Javier Jázquez, he was activated on April 29 to replace Matt Holliday in the Royals’ starting rotation. In his only start with the Royals, on August 6, 2012, Kershaw struck out five batters in seven innings pitched to help the Royals to their first victory in franchise history. On September 27, 2012, it appeared Kershaw was going to pitch a complete game shutout against the Detroit Tigers, but did not manage to do so since the Tigers won 3–1. At the conclusion of the season, Kershaw was named Major League Baseball’s Most Valuable Player, was chosen to the All-Star Game at Busch Stadium and was named to the All-Star Game as the starting pitcher at shortstop. The Royals announced on February 4, 2013 that Kershaw would spend 2013 as starting pitcher, though he was expected to miss the season as a result of a shoulder injury.

On May 17, 2013, Kershaw sustained another back injury and did not start in August and October 2013. He appeared in 22 starts, all starts, finishing with a strikeout-to-walk ratio of 1.50 and a 2.91 ERA. He also had the third most strikeouts in the league: 10. On May 20, 2013, he

Reference:

a 3.15 ERA and 593 career strikeouts in 568.2 innings. According to the Elias Sports Bureau, Kershaw was the first 23-year-old pitcher to have that many victories, an ERA that low and an average of more than one strikeout per inning since ERA became an official statistic in 1910. Kershaw was selected to the National League team for the 2011 Major League Baseball All-Star Game, his first All-Star selection. In the month of July, Kershaw was 4 – 1 with a 2.02 ERA and NL-leading 45 strikeouts, earning him the National League Pitcher of the Month Award. On August 23, he struck out Matt Holliday of the St. Louis Cardinals for his 200th strikeout of the season and became the 10th Dodger pitcher to record back-to-back 200 strikeout seasons and the first since Chan-Ho Park did it in the 2001 season. Kershaw finished the 2011 season by leading the NL with 21 wins, 248 strikeouts and a 2.28 ERA, winning the NL pitching Triple Crown, the first Triple Crown winner since Jake Peavy of the 2007 San Diego Padres and the first Dodger since Sandy Koufax won it in the 1966 season. Justin Verlander of the Detroit Tigers won the American League Triple Crown the same season, marking the first major-league season since 1924 to feature Triple Crown-winning pitchers in both leagues. Kershaw’s 21 wins were the most by a Dodger pitcher since Orel Hershiser won 23 during the 1988 season. His ERA was the lowest by a Dodger since Hershiser’s 2.03 in the 1985 season, his strikeouts were the most by a Dodger since Koufax’s 317 in 1966 and his 233 1/3 innings pitched were the most since Chan Ho Park pitched 234 in 2001. Since 1965 when Koufax did it, Peavy and Kershaw are only two pitchers in the National League have led the league in wins, strikeouts, ERA, and WHIP (walks plus hits per inning pitched). Kershaw also became just the second <unk> to have a 240-plus strikeouts in a season before the age of 24, joining Vida Blue. After the season, Kershaw was awarded the Warren Spahn Award as the best left-handed pitcher in 2011, the Players Choice Award for Most Outstanding National League pitcher, the Gold Glove Award as the top fielding pitcher in the NL and the Sporting News (TSN) National League Pitcher of the Year. He was additionally selected as the starting pitcher for the TSN NL All-Star Team. On November 17, he was honored with the National League Cy Young Award, making him the youngest Cy Young winner since Dwight Gooden.

Table 2.11: Example 1 – 500 tokens generated by XL using a snippet from the Wikitext-103 test set as initial context. The sample is randomly generated without any cherry picking.

There are many interesting observations from this example:

- Firstly, Kershaw never went to Royals in real life. Despite that, Transformer-XL stays on the fully imagined topic and keeps hallucinating the experience of Kershaw in Royals across the generated text.

- Secondly, notice that XL correctly tracks the chronological order from 2011 to 2012 and to the finally 2013 season in the section titles.

- In addition, notice that Transformer-XL accurately uses the the phrase “another back injury” in the 2013 season paragraph, since it has talked about one earlier injure in the 2012 season. This shows again Transformer-XL’s ability of capturing long-term dependency.
Context:

Species range across the Neotropics from Mexico in the north to Bolivia, Paraguay, and southern Brazil in the south. According to a species are conspicuous elements of the landscape, while smaller species are found in both in the forest understory and in savannas. Disturbance has been implicated in the formation of vegetation dominated by large species. In seasonally dry Amazonian forests the density of large adult A. palms was correlated with canopy openness; the species also dominates savannas formed by repeated forest fires in Trinidad and Tobago. A. speciosa forms pure stands in many parts of Brazil where natural forest vegetation has been cleared. Similarly, stands of A. in Bahia, Brazil (which are cultivated for fibre) are managed using fire — the seedlings survive cutting and burning, and are able to dominate burnt forest patches. The fruit are dispersed by animals; fruit which are not dispersed frequently suffer seed predation by <unk> beetles. Certain species of <unk> have been mentioned as examples of "anachronistic" species which are adapted for dispersal by now-extinct Pleistocene megafauna. On Island, A., in the Brazilian Amazon, A. is fruit were consumed by tapires, collared peccaries, deer and primates. Rodents, including agoutis, fed upon the fruit and, as the fruit availability declined, they fed on the seeds. Other dispersers of <unk> fruit include Crested <unk> which consume the fruit and disperse the seeds of A. in the Brazilian Pantanal.

Uses: species have a long history of human utilisation. <unk> seeds have been found in archaeological sites in Colombia dating back to 9000 BP. A variety of species remain important sources of edible oil, thatch, edible seeds and fibre. The leaves of <unk> and A. are used extensively for thatching. Several species are oil palms, with A. speciosa among the most important economically. Products extracted from A. speciosa were reported to support over 300,000 households in the Brazilian state of Maranhão in 2005, and in 1985 it was estimated to support over 450,000 households throughout the Brazil. <unk> fibres, extracted from the leaf bases of A. are, commercially important, and generated about US $ 20 million in annual income to Brazilian farmers in 1996.

Reference:

20 million in annual income to Brazilian farmers in 1996.

The Heart of Ezra Greer is a 1917 American silent drama film produced by the Thanhouser Company and directed by Emile Nouzé. The film focuses on Ezra Greer, a successful middle-aged man who searches for his college age daughter, Mary. The wayward Mary was romanced and abandoned by Jack, later bearing his child. Once Ezra becomes broke he finds employment as the valet for Jack. After Jack's engagement to a cabaret girl, Mary becomes upset and leaves her child at Jack's home. Contrary to Jack's wishes, Ezra keeps the child and Jack ultimately reveals that the child is his own. Ezra convince Jack to make things right and Ezra convinces the cabaret girl to leave Jack. After a carriage accident in which the baby is injured, Ezra and Jack rush to the hospital and find Mary as a nurse caring over the child. The film ends with the marriage of Jack and Mary.

The film was released by Pathé on October 7, 1917. The film was the final release from Thanhouser and was deemed to be an average film by most reviewers. Criticism for the film hinged on far-fetched coincidences to drive the plot. The film is presumed lost.

The film follows Ezra Greer, a middle-aged man who has worked hard since his youth. He cares deeply for his motherless daughter, Mary, but was unable to attend the annual commencement at her coeducational college. He awaits for her to return from college, but Mary leaves with her romantic interest, Jack. On promise of marriage and wealth, Mary is romanced and given birth to a fatherless child. Without word from his daughter, Ezra resigns from his job and attempts to seek her out and finds a poor motherless child, Marie. With Ezra's money exhausted he seeks employment and finds it as the valet of Jack. One day, Mary seeks an announcement of Jack's engagement to a cabaret girl known as "The Baby Vamp". Bitter over the prospect of her child's future, she leaves the child at Jack's home during his absence with a note. Jack orders Ezra to take the baby to an orphanage, but Marie begs Ezra to keep him. After continually seeing the child, Jack is overcome with remorse and explains to Ezra and seeks his advice. Not knowing he was making the case for his own daughter, Ezra convinces Jack to seek out Mary and forget the Baby Vamp. The Baby

Table 2.12: Example 2 – 500 tokens generated by XL using a snippet from the Wikitext-103 test set as initial context. The sample is randomly generated without any cherry picking.
This example exhibit some additional interesting properties of Transformer-XL:

- After finishing the last paragraph of the seed context, both the reference and generated text start a new topic (i.e., Wikipedia page), as marked by the single “\title\” line. This suggests the model has the ability of identifying the end of a topic / page, and randomly starting with a new topic.

- Even more interestingly, a newly-started page is on a book called “The Tale of Mrs. Tittlemouse”. Transformer-XL manages to copy the same book title and some related information from the training set, but hallucinates novel content of the book. This demonstrates a degree of generalization instead of memorization. Please refer to the original book content at the Wikipedia page.
The Battle of Dürenstein (also known as the Battle of <unk>, Battle of <unk> and Battle of <unk>), German: <unk> bei <unk>), on 11 November 1805 was an engagement in the Napoleonic Wars during the War of the Third Coalition. Dürenstein (modern <unk>) is located in the <unk> Valley, on the River Danube, 73 kilometers (45 mi) upstream from Vienna, Austria. The river makes a crescent-shaped curve between <unk> and nearby Krems an der Donau and the battle was fought in the flood plain between the river and the mountains. At Dürenstein a combined force of Russian and Austrian troops trapped a French division commanded by Théodore Maxime Gazan. The French division was part of the newly created VIII Corps, the so-called Corps Mortier, under command of Édouard Mortier. In pursuing the Austrian retreat from Bavaria, Mortier had over-extended his three divisions along the north bank of the Danube. Mikhail <unk> Kutuzov, commander of the Coalition force, enticed Mortier to send Gazan’s division into a trap and French troops were caught in a valley between two Russian columns. They were rescued by the timely arrival of a second division, under command of Pierre Dupont de l’Étang. The battle extended well into the night. Both sides claimed victory. The French lost more than a third of their participants, and Gazan’s division experienced over 40 percent losses. The Austrians and Russians also had heavy <unk> to 16 <unk>; perhaps the most significant was the death in action of Johann Heinrich von Schmitt, one of Austria’s most capable chiefs of staff. The battle was fought three weeks after the Austrian capitulation at Ulm and three weeks before the Russo-Austrian defeat at the Battle of Austerlitz. After Austerlitz Austria withdrew from the war. The French demanded a high indemnity and Francis II abdicated as Holy Roman Emperor, releasing the German states from their allegiance to the Holy Roman Empire.

In a series of conflicts from 1803-15 known as the Napoleonic Wars, various European powers formed five coalitions against the First French Empire. Like the wars sparked by the French Revolution (1789), these further revolutionized the formation, organization and training of European armies and led to an unprecedented militarization, mainly due to mass conscription. Under the leadership of Napoleon, French power rose quickly as the Grande Armée conquered most of Europe, and collapsed rapidly.

Reference:
after the disastrous invasion of Russia in 1812. Napoleon’s empire ultimately suffered complete military defeat in the 1813 – 14 campaigns, resulting in the restoration of the Bourbon monarchy in France. Although Napoleon made a spectacular retreat in 1815, known as the Hundred Days, his defeat at the Battle of Waterloo, the pursuit of his army and himself, his abdication and banishment to the Island of Saint Helena concluded the Napoleonic Wars.

In July 1805, the French 1st Army entered southern Italy. The army, under the command of Marshal Marmont, were reinforced by a few battalions of infantry under Claude General Auguste de Marmont at the town of Philippensburg and another battalion at Belluno. On 17 September 1805, the army marched from Belluno towards Krems. By 29 September, they had reached Belluno and conducted its advance against a small Austrian force. By 31 September, the whole force had been reinforced by a brigade from the Army of Tyrol under the command of Pierre Augereau. The Austrians were now under the command of Marshal Jean Victor Marie Moreau, a member of the Directory. Moreau had taken command of the Austrian invasion force in the spring of 1805. His command included the VI Corps commanded by Jean Baptiste Drouet de Ney and the VI Corps commanded by Generals Jean Victor Marie Moreau and Joseph Souham. Ney’s corps consisted of the III Corps and VI Corps, which consisted of the III Corps and VI Corps, located in the Austrian Netherlands, was commanded by Friedrich Joseph, Count Baillet de Latour. Moreau’s army consisted of six divisions and several associated brigades.

XL Generation:

The Battle of Dürenstein =

The Battle of Dürenstein (also known as the Battle of <unk>, Battle of <unk> and Battle of <unk>), German: <unk> bei <unk>), on 11 November 1805 was an engagement in the Napoleonic Wars during the War of the Third Coalition. Dürenstein (modern <unk>) is located in the <unk> Valley, on the River Danube, 73 kilometers (45 mi) upstream from Vienna, Austria. The river makes a crescent-shaped curve between <unk> and nearby Krems an der Donau and the battle was fought in the flood plain between the river and the mountains. At Dürenstein a combined force of Russian and Austrian troops trapped a French division commanded by Théodore Maxime Gazan. The French division was part of the newly created VIII Corps, the so-called Corps Mortier, under command of Édouard Mortier. In pursuing the Austrian retreat from Bavaria, Mortier had over-extended his three divisions along the north bank of the Danube. Mikhail <unk> Kutuzov, commander of the Coalition force, enticed Mortier to send Gazan’s division into a trap and French troops were caught in a valley between two Russian columns. They were rescued by the timely arrival of a second division, under command of Pierre Dupont de l’Étang. The battle extended well into the night. Both sides claimed victory. The French lost more than a third of their participants, and Gazan’s division experienced over 40 percent losses. The Austrians and Russians also had heavy <unk> to 16 <unk>; perhaps the most significant was the death in action of Johann Heinrich von Schmitt, one of Austria’s most capable chiefs of staff. The battle was fought three weeks after the Austrian capitulation at Ulm and three weeks before the Russo-Austrian defeat at the Battle of Austerlitz. After Austerlitz Austria withdrew from the war. The French demanded a high indemnity and Francis II abdicated as Holy Roman Emperor, releasing the German states from their allegiance to the Holy Roman Empire.

= = Background = =

In July 1805, the French 1st Army entered southern Italy. The army, under the command of Marshal Marmont, were reinforced by a few battalions of infantry under Claude General Auguste de Marmont at the town of Philippensburg and another battalion at Belluno. On 17 September 1805, the army marched from Belluno towards Krems. By 29 September, they had reached Belluno and conducted its advance against a small Austrian force. By 31 September, the whole force had been reinforced by a brigade from the Army of Tyrol under the command of Pierre Augereau. The Austrians were now under the command of Marshal Jean Victor Marie Moreau, a member of the Directory. Moreau had taken command of the Austrian invasion force in the spring of 1805. His command included the VI Corps commanded by Jean Baptiste Drouet de Ney and the VI Corps commanded by Generals Jean Victor Marie Moreau and Joseph Souham. Ney’s corps consisted of the III Corps and VI Corps, which consisted of the III Corps and VI Corps, located in the Austrian Netherlands, was commanded by Friedrich Joseph, Count Baillet de Latour. Moreau’s army consisted of six divisions and several associated brigades.

Reference:

after the disastrous invasion of Russia in 1812. Napoleon’s empire ultimately suffered complete military defeat in the 1813 – 14 campaigns, resulting in the restoration of the Bourbon monarchy in France. Although Napoleon made a spectacular retreat in 1815, known as the Hundred Days, his defeat at the Battle of Waterloo, the pursuit of his army and himself, his abdication and banishment to the Island of Saint Helena concluded the Napoleonic Wars.

= = Background = =

In July 1805, the French 1st Army entered southern Italy. The army, under the command of Marshal Marmont, were reinforced by a few battalions of infantry under Claude General Auguste de Marmont at the town of Philippensburg and another battalion at Belluno. On 17 September 1805, the army marched from Belluno towards Krems. By 29 September, they had reached Belluno and conducted its advance against a small Austrian force. By 31 September, the whole force had been reinforced by a brigade from the Army of Tyrol under the command of Pierre Augereau. The Austrians were now under the command of Marshal Jean Victor Marie Moreau, a member of the Directory. Moreau had taken command of the Austrian invasion force in the spring of 1805. His command included the VI Corps commanded by Jean Baptiste Drouet de Ney and the VI Corps commanded by Generals Jean Victor Marie Moreau and Joseph Souham. Ney’s corps consisted of the III Corps and VI Corps, which consisted of the III Corps and VI Corps, located in the Austrian Netherlands, was commanded by Friedrich Joseph, Count Baillet de Latour. Moreau’s army consisted of six divisions and several associated brigades.

Reference:

after the disastrous invasion of Russia in 1812. Napoleon’s empire ultimately suffered complete military defeat in the 1813 – 14 campaigns, resulting in the restoration of the Bourbon monarchy in France. Although Napoleon made a spectacular retreat in 1815, known as the Hundred Days, his defeat at the Battle of Waterloo, the pursuit of his army and himself, his abdication and banishment to the Island of Saint Helena concluded the Napoleonic Wars.

= = Background = =

In July 1805, the French 1st Army entered southern Italy. The army, under the command of Marshal Marmont, were reinforced by a few battalions of infantry under Claude General Auguste de Marmont at the town of Philippensburg and another battalion at Belluno. On 17 September 1805, the army marched from Belluno towards Krems. By 29 September, they had reached Belluno and conducted its advance against a small Austrian force. By 31 September, the whole force had been reinforced by a brigade from the Army of Tyrol under the command of Pierre Augereau. The Austrians were now under the command of Marshal Jean Victor Marie Moreau, a member of the Directory. Moreau had taken command of the Austrian invasion force in the spring of 1805. His command included the VI Corps commanded by Jean Baptiste Drouet de Ney and the VI Corps commanded by Generals Jean Victor Marie Moreau and Joseph Souham. Ney’s corps consisted of the III Corps and VI Corps, which consisted of the III Corps and VI Corps, located in the Austrian Netherlands, was commanded by Friedrich Joseph, Count Baillet de Latour. Moreau’s army consisted of six divisions and several associated brigades.

Reference:
Notably, from the second section on, the generated text correctly follows a fine-grained chronological order on the level of month and day to narrate events in 1805, except a mistake (1804 instead of 1805) near the end of the paragraph. To ease reading which we have highlighted all the date related phrases by magenta in the generation.


- Although this example is significantly longer, we can see that Transformer-XL is still able to stay on the same topic and makes up non-existing stories about the Napoleon wars.
- Notably, from the second section on, the generated text correctly follows a fine-grained chronological order on the level of month and day to narrate events in 1805, except a mistake (1804 instead of 1805) near the end of the paragraph. To ease reading which we have highlighted all the date related phrases by magenta in the generation.
2.2 XLNet: Generalized Autoregressive Pretraining

2.2.1 Motivation

Unsupervised representation learning has been highly successful in the domain of natural language processing [26, 32, 89, 109, 111]. Typically, these methods first pretrain neural networks on large-scale unlabeled text corpora, and then finetune the models or representations on downstream tasks. Under this shared high-level idea, different unsupervised pretraining objectives have been explored in literature. Among them, autoregressive (AR) language modeling and autoencoding (AE) have been the two most successful pretraining objectives.

AR language modeling seeks to estimate the probability distribution of a text corpus with an autoregressive model [26, 109, 111]. Specifically, given a text sequence \( x = (x_1, \cdots, x_T) \), AR language modeling factorizes the likelihood into a forward product \( p(x) = \prod_{t=1}^{T} p(x_t \mid x_{<t}) \) or a backward one \( p(x) = \prod_{t=T}^{1} p(x_t \mid x_{>t}) \). A parametric model (e.g. a neural network) is trained to model each conditional distribution. Since an AR language model is only trained to encode a uni-directional context (either forward or backward), it is not effective at modeling deep bidirectional contexts. On the contrary, downstream language understanding tasks often require bidirectional context information. This results in a gap between AR language modeling and effective pretraining.

In comparison, AE based pretraining does not perform explicit density estimation but instead aims to reconstruct the original data from corrupted input. A notable example is BERT [32], which has been the state-of-the-art pretraining approach. Given the input token sequence, a certain portion of tokens are replaced by a special symbol \( [\text{MASK}] \), and the model is trained to recover the original tokens from the corrupted version. Since density estimation is not part of the objective, BERT is allowed to utilize bidirectional contexts for reconstruction. As an immediate benefit, this closes the aforementioned bidirectional information gap in AR language modeling, leading to improved performance. However, the artificial symbols like \( [\text{MASK}] \) used by BERT during pretraining are absent from real data at finetuning time, resulting in a pretrain-finetune discrepancy. Moreover, since the predicted tokens are masked in the input, BERT is not able to model the joint probability using the product rule as in AR language modeling. In other words, BERT assumes the predicted tokens are independent of each other given the unmasked tokens, which is oversimplified as high-order, long-range dependency is prevalent in natural language [29].

Faced with the pros and cons of existing language pretraining objectives, in this work, we propose XLNet, a generalized autoregressive method that leverages the best of both AR language modeling and AE while avoiding their limitations.

Related Work The idea of permutation-based AR modeling has been explored in [40, 143], but there are several key differences. Previous models are orderless, while XLNet is essentially order-aware with positional encodings. This is important for language understanding because an orderless model is degenerated to bag-of-words, lacking basic expressivity. The above difference results from the fundamental difference in motivation—previous models aim to improve density estimation by baking an “orderless” inductive bias into the model while XLNet is motivated by enabling AR language models to learn bidirectional contexts.
2.2.2 Approach

Background

In this section, we first review and compare the conventional AR language modeling and BERT for language pretraining. Given a text sequence \( x = [x_1, \cdots, x_T] \), AR language modeling performs pretraining by maximizing the likelihood under the forward autoregressive factorization:

\[
\max_{\theta} \log p_{\theta}(x) = \sum_{t=1}^{T} \log p_{\theta}(x_t \mid x_{<t}) = \sum_{t=1}^{T} \log \frac{\exp \left( h_{\theta}(x_{1:t-1})^T e(x_t) \right)}{\sum_{x'} \exp \left( h_{\theta}(x_{1:t-1})^T e(x') \right)},
\]  

(2.1)

where \( h_{\theta}(x_{1:t-1}) \) is a context representation produced by neural models, such as RNNs or Transformers, and \( e(x) \) denotes the embedding of \( x \). In comparison, BERT is based on denoising auto-encoding. Specifically, for a text sequence \( x \), BERT first constructs a corrupted version \( \hat{x} \) by randomly setting a portion (e.g. 15%) of tokens in \( x \) to a special symbol \([\text{MASK}]\). Let the masked tokens be \( \bar{x} \). The training objective is to reconstruct \( \bar{x} \) from \( \hat{x} \):

\[
\max_{\theta} \log p_{\theta}(\bar{x} \mid \hat{x}) \approx \sum_{t=1}^{T} m_t \log p_{\theta}(x_t \mid \hat{x}) = \sum_{t=1}^{T} m_t \log \frac{\exp \left( H_{\theta}(\bar{x})^T e(x_t) \right)}{\sum_{x'} \exp \left( H_{\theta}(\bar{x})^T e(x') \right)},
\]  

(2.2)

where \( m_t = 1 \) indicates \( x_t \) is masked, and \( H_{\theta} \) is a Transformer that maps a length-\( T \) text sequence \( x \) into a sequence of hidden vectors \( H_{\theta}(x) = [H_{\theta}(x)_1, H_{\theta}(x)_2, \cdots, H_{\theta}(x)_T] \). The pros and cons of the two pretraining objectives are compared in the following aspects:

- **Independence Assumption**: As emphasized by the \( \approx \) sign in Eq. (2.2), BERT factorizes the joint conditional probability \( p(\bar{x} \mid \hat{x}) \) based on an independence assumption that all masked tokens \( \bar{x} \) are separately reconstructed. In comparison, the AR language modeling objective equation (2.1) factorizes \( p_{\theta}(x) \) using the product rule that holds universally without such an independence assumption.

- **Input noise**: The input to BERT contains artificial symbols like \([\text{MASK}]\) that never occur in downstream tasks, which creates a pretrain-finetune discrepancy. Replacing \([\text{MASK}]\) with original tokens as in [32] does not solve the problem because original tokens can be only used with a small probability — otherwise Eq. (2.2) will be trivial to optimize. In comparison, AR language modeling does not rely on any input corruption and does not suffer from this issue.

- **Context dependency**: The AR representation \( h_{\theta}(x_{1:t-1}) \) is only conditioned on the tokens up to position \( t \) (i.e. tokens to the left), while the BERT representation \( H_{\theta}(x) \) has access to the contextual information on both sides. As a result, the BERT objective allows the model to be pretrained to better capture bidirectional context.

Objective: Permutation Language Modeling

According to the comparison above, AR language modeling and BERT possess their unique advantages over the other. A natural question to ask is whether there exists a pretraining objective that brings the advantages of both while avoiding their weaknesses.
Figure 2.5: Illustration of the permutation language modeling objective for predicting $x_3$ given the same input sequence $x$ but with different factorization orders.

Borrowing ideas from orderless NADE [143], we propose the permutation language modeling objective that not only retains the benefits of AR models but also allows models to capture bidirectional contexts. Specifically, for a sequence $x$ of length $T$, there are $T!$ different orders to perform a valid autoregressive factorization. Intuitively, if model parameters are shared across all factorization orders, in expectation, the model will learn to gather information from all positions on both sides.

To formalize the idea, let $Z_T$ be the set of all possible permutations of the length-$T$ index sequence $[1, 2, \ldots, T]$. We use $z_t$ and $z_{<t}$ to denote the $t$-th element and the first $t - 1$ elements of a permutation $z \in Z_T$. Then, our proposed permutation language modeling objective can be expressed as follows:

$$\max_{\theta} \mathbb{E}_{z \sim Z_T} \left[ \sum_{t=1}^{T} \log p_{\theta}(x_{z_t} \mid x_{z_{<t}}) \right].$$  \hspace{1cm} (2.3)

Essentially, for a text sequence $x$, we sample a factorization order $z$ at a time and decompose the likelihood $p_{\theta}(x)$ according to factorization order. Since the same model parameter $\theta$ is shared
Remark on Permutation The proposed objective only permutes the factorization order, not the sequence order. In other words, we keep the original sequence order, use the positional encodings corresponding to the original sequence, and rely on a proper attention mask in Transformers to achieve permutation of the factorization order. Note that this choice is necessary, since the model will only encounter text sequences with the natural order during finetuning.

To provide an overall picture, we show an example of predicting the token $x_3$ given the same input sequence $x$ but under different factorization orders in Figure 2.5.

### Architecture: Two-Stream Self-Attention for Target-Aware Representations

![Diagram](image)

**Figure 2.6:** (a): Content stream attention, which is the same as the standard self-attention. (b): Query stream attention, which does not have access information about the content $x_{zt}$. (c): Overview of the permutation language modeling training with two-stream attention.

While the permutation language modeling objective has desired properties, naive implementation with standard Transformer parameterization may not work. To see the problem, assume we parameterize the next-token distribution $p_\theta(X_{zt} \mid x_{z<zt})$ using the standard Softmax formulation, i.e.,

$$p_\theta(X_{zt} = x \mid x_{z<zt}) = \frac{\exp(e(x) \cdot h_\theta(x_{z<zt}))}{\sum_{x'} \exp(e(x') \cdot h_\theta(x_{z<zt}))},$$

where $h_\theta(x_{z<zt})$ denotes the hidden representation of $x_{z<zt}$ produced by the shared Transformer network after proper masking. Now notice that the representation $h_\theta(x_{z<zt})$ does not depend on which position it will predict, i.e., the value of $z_t$. Consequently, the same distribution is predicted regardless of the target position, which is not able to learn useful representations. Specifically, let’s consider two different permutations $z^{(1)}$ and $z^{(2)}$.
and $z^{(2)}$ satisfying the following relationship
\[ z^{(1)}_{<t} = z^{(2)}_{<t} = z_{<t} \quad \text{but} \quad z_i^{(1)} = i \neq j = z_i^{(2)}. \]

Then, substituting the two permutations respectively into the naive parameterization, we have
\[
p_\theta(X_i = x \mid x_{<i}) = p_\theta(X_j = x \mid x_{<j}) = \frac{\exp (e(x)^\top h(x_{<t}))}{\sum_{x'} \exp (e(x')^\top h(x_{<t}))},
\]

Effectively, two different target positions $i$ and $j$ share exactly the same model prediction. However, the ground-truth distribution of two positions should certainly be different.

To avoid this problem, we propose to re-parameterize the next-token distribution to be target position aware:
\[
p_\theta(X_{zt} = x \mid x_{<t}) = \frac{\exp (e(x)^\top g_\theta(x_{<t}, z_t))}{\sum_{x'} \exp (e(x')^\top g_\theta(x_{<t}, z_t))},
\]

where $g_\theta(x_{<t}, z_t)$ denotes a new type of representations which additionally take the target position $z_t$ as input.

**Two-Stream Self-Attention** While the idea of target-aware representations removes the ambiguity in target prediction, how to formulate $g_\theta(x_{<t}, z_t)$ remains a non-trivial problem. Among other possibilities, we propose to “stand” at the target position $z_t$ and rely on the position $z_t$ to gather information from the context $x_{<t}$ through attention. For this parameterization to work, there are two requirements that are contradictory in a standard Transformer architecture:

1. To predict the token $x_{zt}$, $g_\theta(x_{<t}, z_t)$ should only use the position $z_t$ and not the content $x_{zt}$, otherwise the objective becomes trivial;
2. To predict the other tokens $x_{zt}$ with $j > t$, $g_\theta(x_{<t}, z_t)$ should also encode the content $x_{zt}$ to provide full contextual information. To resolve such a contradiction, we propose to use two sets of hidden representations instead of one:

- The content representation $h_\theta(x_{<t})$, or abbreviated as $h_{zt}$, which serves a similar role to the standard hidden states in Transformer. This representation encodes both the context and $x_{zt}$ itself.
- The query representation $g_\theta(x_{<t}, z_t)$, or abbreviated as $g_{zt}$, which only has access to the contextual information $x_{<t}$ and the position $z_t$, but not the content $x_{zt}$, as discussed above.

Computationally, the first layer query stream is initialized with a trainable vector, i.e. $g_i^{(0)} = w$, while the content stream is set to the corresponding word embedding, i.e. $h_i^{(0)} = e(x_i)$. For each self-attention layer $m = 1, \ldots, M$, the two streams of representations are schematically updated with a shared set of parameters as follows (illustrated in Figures 2.6 (a) and (b)):

\[ g_{zt}^{(m)} \leftarrow \text{Attention}(Q = g_{zt}^{(m-1)}, KV = h_{zt}^{(m-1)}; \theta), \quad (\text{query stream: use } z_t \text{ but cannot see } x_{zt}) \]
\[ h_{zt}^{(m)} \leftarrow \text{Attention}(Q = h_{zt}^{(m-1)}, KV = h_{zt}^{(m-1)}; \theta), \quad (\text{content stream: use both } z_t \text{ and } x_{zt}) \]

where Q, K, V denote the query, key, and value in an attention operation [145]. The update rule of the content representations is exactly the same as the standard self-attention, so during finetuning, 1

1To avoid clutter, we omit the implementation details including multi-head attention, residual connection, layer normalization and position-wise feed-forward as used in Transformer(-XL).
we can simply drop the query stream and use the content stream as a normal Transformer(-XL).

Finally, we can use the last-layer query representation $g_{z_{t}}^{(M)}$ to compute Eq. (2.4).

**Partial Prediction** While the permutation language modeling objective equation 2.3 has several benefits, it is a much more challenging optimization problem due to the permutation and causes slow convergence in preliminary experiments. To reduce the optimization difficulty, we choose to only predict the last tokens in a factorization order. Formally, we split $z$ into a non-target subsequence $z_{\leq c}$ and a target subsequence $z_{>c}$, where $c$ is the cutting point. The objective is to maximize the log-likelihood of the target subsequence conditioned on the non-target subsequence, i.e.,

$$
\max_{\theta} \mathbb{E}_{z \sim Z_{T}} \left[ \log p_{\theta}(x_{z_{>c}} | x_{z_{\leq c}}) \right] = \mathbb{E}_{z \sim Z_{T}} \left[ \sum_{t=c+1}^{|z|} \log p_{\theta}(x_{z_{t}} | x_{z_{<t}}) \right].
$$

Note that $z_{>c}$ is chosen as the target because it possesses the longest context in the sequence given the current factorization order $z$. A hyperparameter $K$ is used such that about $1/K$ tokens are selected for predictions; i.e., $|z|/(|z|-c) \approx K$. For unselected tokens, their query representations need not be computed, which saves speed and memory.

**Incorporating Ideas from Transformer-XL**

Since our objective function fits in the AR framework, we incorporate the state-of-the-art AR language model, Transformer-XL [29], into our pretraining framework, and name our method after it. We integrate two important techniques in Transformer-XL, namely the relative positional encoding scheme and the segment recurrence mechanism. We apply relative positional encodings based on the original sequence as discussed earlier, which is straightforward. Now we discuss how to integrate the recurrence mechanism into the proposed permutation setting and enable the model to reuse hidden states from previous segments. Without loss of generality, suppose we have two segments taken from a long sequence $s$; i.e., $\tilde{x} = s_{1:T}$ and $x = s_{T+1:2T}$. Let $\tilde{z}$ and $z$ be permutations of $[1 \cdots T]$ and $[T+1 \cdots 2T]$ respectively. Then, based on the permutation $\tilde{z}$, we process the first segment, and then cache the obtained content representations $\tilde{h}_{z_{t}}^{(m)}$ for each layer $m$. Then, for the next segment $x$, the attention update with memory can be written as

$$
h_{z_{t}}^{(m)} \leftarrow \text{Attention}(Q = h_{z_{t}}^{(m-1)}, KV = [\tilde{h}_{z_{t}}^{(m-1)}, h_{z_{\leq t}}^{(m-1)}]; \theta)
$$

where $[\cdot, \cdot]$ denotes concatenation along the sequence dimension. Notice that positional encodings only depend on the actual positions in the original sequence. Thus, the above attention update is independent of $\tilde{z}$ once the representations $\tilde{h}_{z_{t}}^{(m)}$ are obtained. This allows caching and reusing the memory without knowing the factorization order of the previous segment. In expectation, the model learns to utilize the memory over all factorization orders of the last segment. The query stream can be computed in the same way. Finally, Figure 2.6(c) presents an overview of the proposed permutation language modeling with two-stream attention.
Visualizing Memory and Permutation

In this section, we provide a detailed visualization of the proposed permutation language modeling objective, including the mechanism of reusing memory (aka the recurrence mechanism), how we use attention masks to permute the factorization order, and the difference of the two attention streams. As shown in Figure 2.7 and 2.8, given the current position $z_t$, the attention mask is decided by the permutation (or factorization order) $z$ such that only tokens the occur before $z_t$ in the permutation can be attended; i.e., positions $z_i$ with $i < t$. Moreover, comparing Figure 2.7 and 2.8, we can see how the query stream and the content stream work differently with a specific permutation through attention masks. The main difference is that the query stream cannot do self-attention and does not have access to the token at the position, while the content stream performs normal self-attention.

Modeling Multiple Segments

Many downstream tasks have multiple input segments, e.g., a question and a context paragraph in question answering. We now discuss how we pretrain XLNet to model multiple segments in the autoregressive framework. During the pretraining phase, following BERT, we randomly sample two segments (either from the same context or not) and treat the concatenation of two segments as one sequence to perform permutation language modeling. We only reuse the memory that belongs to the same context. Specifically, the input to our model is similar to BERT: [A, SEP, B, SEP, CLS], where “SEP” and “CLS” are two special symbols and “A” and “B” are the two segments. Although we follow the two-segment data format, XLNet-Large does not use the objective of next sentence prediction [32] as it does not show consistent improvement in our ablation study (see Section 2.2.3).

Relative Segment Encodings

Architecturally, different from BERT that adds an absolute segment embedding to the word embedding at each position, we extend the idea of relative encodings from Transformer-XL to also encode the segments. Given a pair of positions $i$ and $j$ in the sequence, if $i$ and $j$ are from the same segment, we use a segment encoding $s_{ij} = s_+$ or otherwise $s_{ij} = s_-$, where $s_+$ and $s_-$ are learnable model parameters for each attention head. In other words, we only consider whether the two positions are within the same segment, as opposed to considering which specific segments they are from. This is consistent with the core idea of relative encodings; i.e., only modeling the relationships between positions. When $i$ attends to $j$, the segment encoding $s_{ij}$ is used to compute an attention weight $a_{ij} = (q_i + b)^ts_{ij}$, where $q_i$ is the query vector as in a standard attention operation and $b$ is a learnable head-specific bias vector. Finally, the value $a_{ij}$ is added to the normal attention weight. There are two benefits of using relative segment encodings. First, the inductive bias of relative encodings improves generalization [29]. Second, it opens the possibility of finetuning on tasks that have more than two input segments, which is not possible using absolute segment encodings.
Figure 2.7: A detailed illustration of the content stream of the proposed objective with both the joint view and split views based on a length-4 sequence under the factorization order $[3 \rightarrow 2 \rightarrow 4 \rightarrow 1]$. Note that if we ignore the query representation, the computation in this figure is simply the standard self-attention, though with a particular attention mask.
Figure 2.8: A detailed illustration of the **query stream** of the proposed objective with both the joint view and split views based on a length-4 sequence under the factorization order [3 \rightarrow 2 \rightarrow 4 \rightarrow 1]. The dash arrows indicate that the query stream cannot access the token (content) at the same position, but only the location information.
Discussion and Analysis

Comparison with BERT
Comparing Eq. (2.2) and (2.5), we observe that both BERT and XLNet perform partial prediction, i.e., only predicting a subset of tokens in the sequence. This is a necessary choice for BERT because if all tokens are masked, it is impossible to make any meaningful predictions. In addition, for both BERT and XLNet, partial prediction plays a role of reducing optimization difficulty by only predicting tokens with sufficient context. However, the independence assumption discussed in Section 2.2.2 disables BERT to model dependency between targets.

To better understand the difference, let’s consider a concrete example [New, York, is, a, city]. Suppose both BERT and XLNet select the two tokens [New, York] as the prediction targets and maximize

\[ \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city}) \]

Also suppose that XLNet samples the factorization order [is, a, city, New, York]. In this case, BERT and XLNet respectively reduce to the following objectives:

\[ J_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city}) \]
\[ J_{\text{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New}, \text{is a city}) \]

Notice that XLNet is able to capture the dependency between the pair (New, York), which is omitted by BERT. Although in this example, BERT learns some dependency pairs such as (New, city) and (York, city), it is obvious that XLNet always learns more dependency pairs given the same target and contains “denser” effective training signals.

To prove a general point beyond one example, we now turn to more formal expressions. Inspired by previous work \cite{169}, given a sequence \( x = [x_1, \ldots, x_T] \), we define a set of target-context pairs of interest, \( \mathcal{I} = \{(x, \mathcal{U})\} \), where \( \mathcal{U} \) is a set of tokens in \( x \) that form a context of \( x \). Intuitively, we want the model to learn the dependency of \( x \) on \( \mathcal{U} \) through a pretraining loss term \( \log p(x \mid \mathcal{U}) \). For example, given the above sentence, the pairs of interest \( \mathcal{I} \) could be instantiated as:

\[ \mathcal{I} = \left\{ (x = \text{York}, \mathcal{U} = \{\text{New}\}), (x = \text{York}, \mathcal{U} = \{\text{city}\}), (x = \text{York}, \mathcal{U} = \{\text{New, city}\}), \cdots \right\} \]

Note that \( \mathcal{I} \) is merely a virtual notion without unique ground truth, and our analysis will hold regardless of how \( \mathcal{I} \) is instantiated.

Given a set of target tokens \( \mathcal{T} \) and a set of non-target tokens \( \mathcal{N} = x \setminus \mathcal{T} \), BERT and XLNet both maximize \( \log p(\mathcal{T} \mid \mathcal{N}) \) but with different formulations:

\[ J_{\text{BERT}} = \sum_{x \in \mathcal{T}} \log p(x \mid \mathcal{N}); \quad J_{\text{XLNet}} = \sum_{x \in \mathcal{T}} \log p(x \mid \mathcal{N} \cup \mathcal{T}_{<x}) \]

where \( \mathcal{T}_{<x} \) denote tokens in \( \mathcal{T} \) that have a factorization order prior to \( x \). Both objectives consist of multiple loss terms in the form of \( \log p(x \mid \mathcal{V}_x) \). Intuitively, if there exists a target-context pair \( (x, \mathcal{U}) \in \mathcal{I} \) such that \( \mathcal{U} \subseteq \mathcal{V}_x \), then the loss term \( \log p(x \mid \mathcal{V}_x) \) provides a training signal to the dependency between \( x \) and \( \mathcal{U} \). For convenience, we say a target-context pair \( (x, \mathcal{U}) \in \mathcal{I} \) is covered by a model (objective) if \( \mathcal{U} \subseteq \mathcal{V}_x \).

Given the definition, let’s consider two cases:
• If $\mathcal{U} \subseteq \mathcal{N}$, the dependency $(x, \mathcal{U})$ is covered by both BERT and XLNet.
• If $\mathcal{U} \subseteq \mathcal{N} \cup \mathcal{T}_{<x}$ and $\mathcal{U} \cap \mathcal{T}_{<x} \neq \emptyset$, the dependency can only be covered by XLNet but not BERT. As a result, XLNet is able to cover more dependencies than BERT. In other words, the XLNet objective contains more effective training signals, which empirically leads to better performance in Section 3.3.4.

**Comparison with Language Modeling** Borrowing the above examples and notations, a standard AR language model like GPT \cite{NEURIPS2019_9015} is only able to cover the dependency $(x = \text{York}, \mathcal{U} = \{\text{New}\})$ but not $(x = \text{New}, \mathcal{U} = \{\text{York}\})$. XLNet, on the other hand, is able to cover both in expectation over all factorization orders. Such a limitation of AR language modeling can be critical in real-world applications. For example, consider a span extraction question answering task with the context “Thom Yorke is the singer of Radiohead” and the question “Who is the singer of Radiohead”. The representations of “Thom Yorke” are not dependent on “Radiohead” with AR language modeling and thus they will not be chosen as the answer by the standard approach that employs softmax over all token representations. More formally, consider a context-target pair $(x, \mathcal{U})$:

- If $\mathcal{U} \cap \mathcal{T}_{<x} \neq \emptyset$, where $\mathcal{T}_{<x}$ denotes the tokens prior to $x$ in the original sequence, AR language modeling is not able to cover the dependency.
- In comparison, XLNet is able to cover all dependencies in expectation.

Approaches like ELMo \cite{Peters:18} concatenate forward and backward language models in a shallow manner, which is not sufficient for modeling deep interactions between the two directions.

**Bridging the Gap Between Language Modeling and Pretraining** With a deep root in density estimation\footnote{The problem of language modeling is essentially density estimation for text data.}, language modeling has been a rapidly-developing research area \cite{Joulin:16,li:2019,NEURIPS2019_9015}. However, there has been a gap between language modeling and pretraining due to the lack of the capability of bidirectional context modeling, as analyzed above. It has even been challenged by some machine learning practitioners whether language modeling is a meaningful pursuit if it does not directly improve downstream tasks\footnote{https://openreview.net/forum?id=HJePno0cYm}. XLNet generalizes language modeling and bridges such a gap. As a result, it further “justifies” language modeling research. Moreover, it becomes possible to leverage the rapid progress of language modeling research for pretraining. As an example, we integrate Transformer-XL into XLNet to demonstrate the usefulness of the latest language modeling progress.

### 2.2.3 Experiments

**Pretraining and Implementation**

Following BERT \cite{Devlin:19}, we use the BooksCorpus \cite{Zhu:18} and English Wikipedia as part of our pretraining data, which have 13GB plain text combined. In addition, we include Giga5 (16GB text) \cite{Zhu:18}, ClueWeb 2012-B (extended from \cite{Zhu:19}), and Common Crawl \cite{45585} for pretraining. We use heuristics to aggressively filter out short or low-quality articles for ClueWeb 2012-B and Common Crawl, which results in 19GB and 78GB text respectively. After tokenization
Table 2.14: Comparison with state-of-the-art results on the test set of RACE, a reading comprehension task. ∗ indicates using ensembles. “Middle” and “High” in RACE are two subsets representing middle and high school difficulty levels. All BERT and XLNet results are obtained with a 24-layer architecture with similar model sizes (aka BERT-Large). Our single model outperforms the best ensemble by 7.6 points in accuracy.

with SentencePiece [76], we obtain 2.78B, 1.09B, 4.75B, 4.30B, and 19.97B subword pieces for Wikipedia, BooksCorpus, Giga5, ClueWeb, and Common Crawl respectively, which are 32.89B in total.

Our largest model XLNet-Large has the same architecture hyperparameters as BERT-Large, which results in a similar model size. The sequence length and memory length are set to 512 and 384 respectively. We train XLNet-Large on 512 TPU v3 chips for 500K steps with an Adam optimizer, linear learning rate decay and a batch size of 2048, which takes about 2.5 days. It was observed that the model still underfits the data at the end of training but continuing training did not help downstream tasks, which indicates that given the optimization algorithm, the model does not have enough capacity to fully leverage the data scale. However, in this work, we refrain from training a larger model as its practical usage for finetuning might be limited. Further, we train an XLNet-Base, analogous to BERT-Base, on BooksCorpus and Wikipedia only, for ablation study and fair comparison with BERT. Related results are presented in Section 2.2.3.

Since the recurrence mechanism is introduced, we use a bidirectional data input pipeline where each of the forward and backward directions takes half of the batch size. For training XLNet-Large, we set the partial prediction constant $K$ as 6 (see Section 2.2.2). Our finetuning procedure follows BERT [32] except otherwise specified. We employ an idea of span-based prediction, where we first sample a length $L \in [1, \cdots, 5]$, and then randomly select a consecutive span of $L$ tokens as prediction targets within a context of $(KL)$ tokens.

**RACE Dataset**

The RACE dataset [77] contains near 100K questions taken from the English exams for middle and high school Chinese students in the age range between 12 to 18, with the answers generated by human experts. This is one of the most difficult reading comprehension datasets that involve challenging reasoning questions. Moreover, the average length of the passages in RACE are longer than 300, which is significantly longer than other popular reading comprehension datasets such as SQuAD [114]. As a result, this dataset serves as a challenging benchmark for long text understanding. We use a sequence length of 640 during finetuning. As shown in Table 2.14, a
### Table 2.15:

A single model XLNet outperforms human and the best ensemble by 7.6 EM and 2.5 EM on SQuAD1.1. * means ensembles, † marks our runs with the official code.

<table>
<thead>
<tr>
<th>SQuAD1.1</th>
<th>EM</th>
<th>F1</th>
<th>SQuAD2.0</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>84.1</td>
<td>90.9</td>
<td>BERT†</td>
<td>78.98</td>
<td>81.77</td>
</tr>
<tr>
<td>XLNet</td>
<td><strong>88.95</strong></td>
<td><strong>94.52</strong></td>
<td>XLNet</td>
<td><strong>86.12</strong></td>
<td><strong>88.79</strong></td>
</tr>
</tbody>
</table>

**Dev set results without data augmentation**

**Test set results on leaderboard, with data augmentation (as of June 19, 2019)**

| Human    | 82.30 | 91.22| BERT+N-Gram+Self-Training† | 85.15 | 87.72 |
| ATB      | 86.94 | 92.64| SG-Net                   | 85.23 | 87.93 |
| BERT*    | 87.43 | 93.16| BERT+DAE+AoA             | 85.88 | 88.62 |
| XLNet    | **89.90** | **95.08** | XLNet               | **86.35** | **89.13** |

The single model XLNet outperforms the best ensemble by 7.6 points in accuracy. It is also clear that XLNet substantially outperforms other pretrained models such as BERT and GPT. Since RACE contains relatively long passages, we believe one of the reasons why XLNet obtains substantial gains on this dataset is that the integration of the Transformer-XL architecture improves the capability of modeling long text, besides the AR objective. More analysis on the sequence length is presented in Section 2.2.3.

### SQuAD Dataset

SQuAD is a large-scale reading comprehension dataset with two tasks. SQuAD1.1 [113] contains questions that always have a corresponding answer in the given passages, while SQuAD2.0 [114] introduces unanswerable questions. To finetune an XLNet on SQuAD2.0, we jointly apply a logistic regression loss for answerability prediction similar to classification tasks and a standard span extraction loss for question answering [32]. Since v1.1 and v2.0 share the same answerable questions in the training set, we simply remove the answerability prediction part from the model finetuned on v2.0 for evaluation on v1.1. As the top leaderboard entries all employ some form of data augmentation, we jointly train an XLNet on SQuAD2.0 and NewsQA [139] for our leaderboard submission. As shown in Table 2.15, XLNet obtains the state-of-the-art single model results on the leaderboard, outperforming a series of BERT-based methods. Notably, on v1.1, an XLNet single model outperforms human and the best ensemble by 7.6 and 2.5 points in EM. Finally, for direct comparison with BERT to eliminate the effects of additional tricks in leaderboard submissions, we compare XLNet against BERT on the dev set. XLNet substantially outperforms BERT by 3.6 and 7.0 points in F1 for v1.1 and v2.0.

### Text Classification

Following previous work on text classification [98, 174], we evaluate XLNet on the following benchmarks: IMDB, Yelp-2, Yelp-5, DBpedia, AG, Amazon-2, and Amazon-5. According to Table 2.16, XLNet achieves new state-of-the-art results on all the considered datasets, reducing
Table 2.16: Comparison with state-of-the-art error rates on the test sets of several text classification datasets. All BERT and XLNet results are obtained with a 24-layer architecture with similar model sizes (aka BERT-Large).

<table>
<thead>
<tr>
<th>Model</th>
<th>IMDB</th>
<th>Yelp-2</th>
<th>Yelp-5</th>
<th>DBpedia</th>
<th>AG</th>
<th>Amazon-2</th>
<th>Amazon-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN [64]</td>
<td>-</td>
<td>2.90</td>
<td>32.39</td>
<td>0.84</td>
<td>6.57</td>
<td>3.79</td>
<td>36.24</td>
</tr>
<tr>
<td>DPCNN [64]</td>
<td>-</td>
<td>2.64</td>
<td>30.58</td>
<td>0.88</td>
<td>6.87</td>
<td>3.32</td>
<td>34.81</td>
</tr>
<tr>
<td>Mixed VAT [98, 119]</td>
<td>4.32</td>
<td>-</td>
<td>-</td>
<td>0.70</td>
<td>4.95</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ULMFiT [57]</td>
<td>4.6</td>
<td>2.16</td>
<td>29.98</td>
<td>0.80</td>
<td>5.01</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BERT [161]</td>
<td>4.51</td>
<td>1.89</td>
<td>29.32</td>
<td>0.64</td>
<td>-</td>
<td>2.63</td>
<td>34.17</td>
</tr>
</tbody>
</table>

Table 2.17: Results on GLUE. * indicates using ensembles, and † denotes single-task results in a multi-task row. All results are based on a 24-layer architecture with similar model sizes (aka BERT-Large). See the upper-most rows for direct comparison with BERT and the lower-most rows for comparison with state-of-the-art results on the public leaderboard.

The GLUE dataset [149] is a collection of 9 natural language understanding tasks. The test set labels are removed from the publicly released version, and all the practitioners must submit their predictions on the evaluation server to obtain test set results. In Table 2.17, we present results of multiple settings, including single-task and multi-task, as well as single models and ensembles. In the multi-task setting, we jointly train an XLNet on the four largest datasets—MNLI, SST-2, QNLI, and QQP—and finetune the network on the other datasets. Only single-task training is employed for the four large datasets. For QNLI, we employed a pairwise relevance ranking scheme as in [85] for our test set submission. However, for fair comparison with BERT, our result on the QNLI dev set is based on a standard classification paradigm. For WNLI, we use the
Table 2.18: Comparison with state-of-the-art results on the test set of ClueWeb09-B, a document ranking task. † indicates our implementations.

<table>
<thead>
<tr>
<th>Model</th>
<th>NDCG@20</th>
<th>ERR@20</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRMM 52</td>
<td>24.3</td>
<td>13.8</td>
</tr>
<tr>
<td>KNRM 27</td>
<td>26.9</td>
<td>14.9</td>
</tr>
<tr>
<td>Conv 27</td>
<td>28.7</td>
<td>18.1</td>
</tr>
<tr>
<td>BERT†</td>
<td>30.53</td>
<td>18.67</td>
</tr>
<tr>
<td>XLNet</td>
<td>31.10</td>
<td>20.28</td>
</tr>
</tbody>
</table>

loss described in [73]. A multi-task ensemble XLNet achieves the state-of-the-art results on 7 out of 9 tasks on the public leaderboard. On the most widely-benchmarked task MNLI, XLNet improves the “matched” and “mismatched” settings by 2.0 and 1.8 points respectively. Note that the leaderboard competitors employ improved techniques over BERT such as distillation, modified multi-task losses, or meta learning, but still underperform XLNet which does not employ additional tricks besides using a standard multi-task learning method. Since the leaderboard is not intended for ablation study or hyperparameter tuning, we only evaluated our best multi-task models on the test set. To obtain a direct comparison with BERT, we run a single-task XLNet on the dev set. As shown in the upper-most rows of Table 2.17, XLNet consistently outperforms BERT, with an improvement of 13.4 points, 3.2 points, 3.0 points, 2.4 points, 1.8 points on RTE, MNLI, CoLA, SST-2, and STS-B respectively.

ClueWeb09-B Dataset

Following the setting in previous work [27], we use the ClueWeb09-B dataset to evaluate the performance on document ranking. The queries were created by the TREC 2009-2012 Web Tracks based on 50M documents and the task is to rerank the top 100 documents retrieved using a standard retrieval method. Since document ranking, or ad-hoc retrieval, mainly concerns the low-level representations instead of high-level semantics, this dataset serves as a testbed for evaluating the quality of word embeddings. We use a pretrained XLNet to extract word embeddings for the documents and queries without finetuning, and employ a kernel pooling network [164] to rank the documents. According to Table 2.18, XLNet substantially outperforms the other methods, including a BERT model that uses the same training procedure as ours. This illustrates that XLNet learns better low-level word embeddings than BERT. Note that for fair comparison we exclude the results (19.55 in ERR@20, slightly worse than ours) in [163] as it uses additional entity-related data.

Ablation Study

We perform an ablation study to understand the importance of each design choice based on four datasets with diverse characteristics. Specifically, there are three main aspects we hope to study:

- The effectiveness of the permutation language modeling objective, especially compared to the
Table 2.19: Ablation study. The results of BERT on RACE are taken from [173]. We run BERT on the other datasets using the official implementation and the same hyperparameter search space as XLNet. $K$ is a hyperparameter to control the optimization difficulty (see Section 2.2.2). All models are pretrained on the same data.

denoising auto-encoding objective used by BERT.

- The importance of using Transformer-XL as the backbone neural architecture and employing segment-level recurrence (i.e. using memory).
- The necessity of some implementation details including span-based prediction, the bidirectional input pipeline, and next-sentence prediction.

With these purposes in mind, in Table 3.2, we compare 6 XLNet-Base variants with different implementation details (rows 3 - 8), the original BERT-Base model (row 1), and an additional Transformer-XL baseline trained with the denoising auto-encoding (DAE) objective used in BERT but with the bidirectional input pipeline (row 2). For fair comparison, all models are based on a 12-layer architecture with the same model hyper-parameters as BERT-Base and are trained on only Wikipedia and the BooksCorpus. All results reported are the median of 5 runs.

Examining rows 1 - 4 of Table 3.2, we see the two full XLNet-Base models trained with different values of $K$ significantly outperform both BERT and the DAE trained Transformer-XL across tasks, showing the superiority of the permutation language modeling objective. Meanwhile, it is also interesting to see that the DAE trained Transformer-XL achieves better performance than BERT on tasks with long text such as RACE and SQuAD, suggesting the excellence of Transformer-XL in language modeling also benefits pretraining. Next, if we remove the memory caching mechanism (row 5), the performance clearly drops, especially for RACE which involves the longest context among the 4 tasks. In addition, rows 6 - 7 show that both span-based prediction and the bidirectional input pipeline play important roles in XLNet. Finally, we unexpectedly find the the next-sentence prediction objective proposed in the original BERT does not necessarily lead to an improvement in our setting. Instead, it tends to harm the performance except for the RACE dataset. Hence, when we train XLNet-Large, we exclude the next-sentence prediction objective.
Chapter 3

Semi-Supervised Learning with Generative Modeling

This chapter focuses on using generative modeling to improve downstream task performance in a semi-supervised learning setting. We attempt to answer the following questions:

- Can we understand GAN-based semi-supervised learning in a more fundamental way? How does the classifier benefit from joint training with the generator?
- Can we apply generative modeling based approaches to semi-supervised tasks with more complex data structures, such as classification on graphs and question answering?

We consider three scenarios: 1) semi-supervised classification by generating low-density adversarial samples, 2) semi-supervised question answering by generating natural language questions given context, and 3) semi-supervised learning on graphs by modeling the generation of random walk.

3.1 Semi-Supervised Learning with GANs

3.1.1 Motivation

Deep neural networks are usually trained on a large amount of labeled data, and it has been a challenge to apply deep models to datasets with limited labels. Semi-supervised learning (SSL) aims to leverage the large amount of unlabeled data to boost the model performance, particularly focusing on the setting where the amount of available labeled data is limited. Traditional graph-based methods [8, 179] were extended to deep neural networks [71, 155, 166], which involves applying convolutional neural networks [81] and feature learning techniques to graphs so that the underlying manifold structure can be exploited. [116] employs a Ladder network to minimize the layerwise reconstruction loss in addition to the standard classification loss. Variational auto-encoders have also been used for semi-supervised learning [70, 87] by maximizing the variational lower bound of the unlabeled data log-likelihood.

Recently, generative adversarial networks (GANs) [43] were demonstrated to be able to
generate visually realistic images. GANs set up an adversarial game between a discriminator and a generator. The goal of the discriminator is to tell whether a sample is drawn from true data or generated by the generator, while the generator is optimized to generate samples that are not distinguishable by the discriminator. Feature matching (FM) GANs \cite{120} apply GANs to semi-supervised learning on $K$-class classification. The objective of the generator is to match the first-order feature statistics between the generator distribution and the true distribution. Instead of binary classification, the discriminator employs a $(K + 1)$-class objective, where true samples are classified into the first $K$ classes and generated samples are classified into the $(K + 1)$-th class. This $(K + 1)$-class discriminator objective leads to strong empirical results, and was later widely used to evaluate the effectiveness of generative models \cite{36,142}.

Though empirically feature matching improves semi-supervised classification performance, the following questions still remain open. First, it is not clear why the formulation of the discriminator can improve the performance when combined with a generator. Second, it seems that good semi-supervised learning and a good generator cannot be obtained at the same time. For example, \cite{120} observed that mini-batch discrimination generates better images than feature matching, but feature matching obtains a much better semi-supervised learning performance. The same phenomenon was also observed in \cite{142}, where the model generated better images but failed to improve the performance on semi-supervised learning.

In this work, we take a step towards addressing these questions. First, we show that given the current $(K + 1)$-class discriminator formulation of GAN-based SSL, good semi-supervised learning requires a “bad” generator. Here by bad we mean the generator distribution should not match the true data distribution. Then, we give the definition of a preferred generator, which is to generate complement samples in the feature space. Theoretically, under mild assumptions, we show that a properly optimized discriminator obtains correct decision boundaries in high-density areas in the feature space if the generator is a complement generator.

Based on our theoretical insights, we analyze why feature matching works on 2-dimensional toy datasets. It turns out that our practical observations align well with our theory. However, we also find that the feature matching objective has several drawbacks. Therefore, we develop a novel formulation of the discriminator and generator objectives to address these drawbacks. In our approach, the generator minimizes the KL divergence between the generator distribution and a target distribution that assigns high densities for data points with low densities in the true distribution, which corresponds to the idea of a complement generator. Furthermore, to enforce our assumptions in the theoretical analysis, we add the conditional entropy term to the discriminator objective.

### 3.1.2 Related Work

Besides the adversarial feature matching approach \cite{120}, several previous works have incorporated the idea of adversarial training in semi-supervised learning. Notably, \cite{131} proposes categorical generative adversarial networks (CatGAN), which substitutes the binary discriminator in standard GAN with a multi-class classifier, and trains both the generator and the discriminator using information theoretical criteria on unlabeled data. From the perspective of regularization, \cite{97,99} propose virtual adversarial training (VAT), which effectively smooths the output distribution of
the classifier by seeking virtually adversarial samples. It is worth noting that VAT bears a similar merit to our approach, which is to learn from auxiliary non-realistic samples rather than realistic data samples. Despite the similarity, the principles of VAT and our approach are orthogonal, where VAT aims to enforce a smooth function while we aim to leverage a generator to better detect the low-density boundaries. Different from aforementioned approaches, [171] proposes to train conditional generators with adversarial training to obtain complete sample pairs, which can be directly used as additional training cases. Recently, Triple GAN [82] also employs the idea of conditional generator, but uses adversarial cost to match the two model-defined factorizations of the joint distribution with the one defined by paired data.

Apart from adversarial training, there has been other efforts in semi-supervised learning using deep generative models recently. As an early work, [70] adapts the original Variational Auto-Encoder (VAE) to a semi-supervised learning setting by treating the classification label as an additional latent variable in the directed generative model. [87] adds auxiliary variables to the deep VAE structure to make variational distribution more expressive. With the boosted model expressiveness, auxiliary deep generative models (ADGM) improve the semi-supervised learning performance upon the semi-supervised VAE. Different from the explicit usage of deep generative models, the Ladder networks [116] take advantage of the local (layerwise) denoising auto-encoding criterion, and create a more informative unsupervised signal through lateral connection.

The idea of using samples to define the domain distribution has been explored in [154]. The main difference is that we consider using generative models to generate complement samples instead of using existing data.

3.1.3 Theoretical Analysis

Given a labeled set \( L = \{ (x, y) \} \), let \( \{ 1, 2, \cdots, K \} \) be the label space for classification. Let \( D \) and \( G \) denote the discriminator and generator, and \( P_D \) and \( p_G \) denote the corresponding distributions. Consider the discriminator objective function of GAN-based semi-supervised learning [120]:

\[
\max_D \mathbb{E}_{x,y \sim L} \log P_D(y|x, y \leq K) + \mathbb{E}_{x \sim p} \log P_D(y \leq K|x) + \mathbb{E}_{x \sim p_G} \log P_D(K + 1|x),
\]

where \( p \) is the true data distribution. The probability distribution \( P_D \) is over \( K + 1 \) classes where the first \( K \) classes are true classes and the \((K + 1)\)-th class is the fake class. The objective function consists of three terms. The first term is to maximize the log conditional probability for labeled data, which is the standard cost as in supervised learning setting. The second term is to maximize the log probability of the first \( K \) classes for unlabeled data. The third term is to maximize the log probability of the \((K + 1)\)-th class for generated data. Note that the above objective function bears a similar merit to the original GAN formulation if we treat \( P(K + 1|x) \) to be the probability of fake samples, while the only difference is that we split the probability of true samples into \( K \) sub-classes.

Let \( h(x) \) be a nonlinear vector-valued function, and \( w_k \) be the weight vector for class \( k \). As a standard setting in previous work [36] [120], the discriminator \( D \) is defined as \( P_D(k|x) = \frac{\exp(w_k^T h(x))}{\sum_{k=1}^{K+1} \exp(w_k^T h(x))} \). Since this is a form of over-parameterization, \( w_{K+1} \) is fixed as a zero vector [120]. We next discuss the choices of different possible \( G \)’s.
Perfect Generator

Here, by perfect generator we mean that the generator distribution \( p_G \) exactly matches the true data distribution \( p \), i.e., \( p_G = p \). We now show that when the generator is perfect, it does not improve the generalization over the supervised learning setting.

**Proposition 1.** If \( p_G = p \), and \( D \) has infinite capacity, then for any optimal solution \( D = (w, h) \) of the following supervised objective,

\[
\max_D \mathbb{E}_{x,y \sim \mathcal{L}} \log P_D(y|x, y \leq K), \tag{3.2}
\]

there exists \( D^* = (w^*, h^*) \) such that \( D^* \) maximizes Eq. (3.1) and that for all \( x \), \( P_D(y|x, y \leq K) = P_{D^*}(y|x, y \leq K) \).

**Proof.** Given an optimal solution \( D = (w, h) \) for the supervised objective, due to the infinite capacity of the discriminator, there exists \( D^* = (w^*, h^*) \) such that for all \( x \) and \( k \leq K \),

\[
\exp(w^*_k h^*(x)) = \frac{\exp(w^*_k h(x))}{\sum_{k'} \exp(w^*_k h(x))} \tag{3.3}
\]

For all \( x \),

\[
P_{D^*}(y|x, y \leq K) = \frac{\exp(w^*_k h^*(x))}{\sum_{k'} \exp(w^*_k h^*(x))} = \frac{\exp(w^*_k h(x))}{\sum_{k'} \exp(w^*_k h(x))} = P_D(y|x, y \leq K)
\]

Let \( L_D \) be the supervised objective in Eq. (3.1). Since \( p = p_G \), the objective in Eq. (3.1) can be written as

\[
J_D = L_D + \mathbb{E}_{x \sim p} [\log P_D(K + 1|x) + \log(1 - P_D(K + 1|x))]
\]

Given Eq. (3.3), we have

\[
P_{D^*}(K + 1|x) = \frac{1}{1 + \sum_k \exp w^*_k h^*(x)} = \frac{1}{2}
\]

Therefore, \( D^* \) maximizes the second term of \( J_D \). Because \( D \) maximizes \( L_D \), \( D^* \) also maximizes \( L_D \). It follows that \( D^* \) maximizes \( J_D \). \( \square \)

Proposition 1 states that for any optimal solution \( D \) of the supervised objective, there exists an optimal solution \( D^* \) of the \((K + 1)\)-class objective such that \( D \) and \( D^* \) share the same generalization error. In other words, using the \((K + 1)\)-class objective does not prevent the model from experiencing any arbitrarily high generalization error that it could suffer from under the supervised objective. Moreover, since all the optimal solutions are equivalent w.r.t. the \((K + 1)\)-class objective, it is the optimization algorithm that really decides which specific solution the model will reach, and thus what generalization performance it will achieve. This implies that when the generator is perfect, the \((K + 1)\)-class objective by itself is not able to improve the generalization performance. In fact, in many applications, an almost infinite amount of unlabeled data is available, so learning a perfect generator for purely sampling purposes should not be useful. In this case, our theory suggests that not only the generator does not help, but also unlabeled data is not effectively utilized when the generator is perfect.
Complement Generator

The function $h$ maps data points in the input space to the feature space. We use $f = h(x)$ to denote feature space data points, as opposed to $x$ that denotes input space data points. Let $p_k(f)$ be the density of the data points of class $k$ in the feature space. Given a threshold $\epsilon_k$, let $F_k$ be a subset of the data support where $p_k(f) > \epsilon_k$, i.e., $F_k = \{ f : p_k(f) > \epsilon_k \}$. We assume that given $\{\epsilon_k\}_{k=1}^K$, the $F_k$’s are disjoint with a margin. More formally, for any $f_j \in F_j$, $f_k \in F_k$, and $j \neq k$, we assume that there exists a real number $0 < \alpha < 1$ such that $\alpha f_j + (1-\alpha) f_k \notin F_j \cup F_k$. As long as the probability densities of different classes do not share any mode, i.e., $\forall i \neq j, \text{argmax}_f p_i(f) \cap \text{argmax}_f p_j(f) = \emptyset$, this assumption can always be satisfied by tuning the thresholds $\epsilon_k$’s. With the assumption held, we will show that the model performance would be better if the thresholds could be set to smaller values (ideally zero). We also assume that each $F_k$ contains at least one labeled data point.

Suppose $\cup_{k=1}^K F_k$ is bounded by a convex set $B$. If the support $F_G$ of a generator $G$ in the feature space is a relative complement set in $B$, i.e., $F_G = B - \cup_{k=1}^K F_k$, we call $G$ a complement generator. The reason why we utilize a bounded $B$ to define the complement is presented in Section 3.1.3. Note that the definition of complement generator implies that $G$ is a function of $h$. By treating $G$ as a function of $h$, theoretically $D$ can optimize the original objective function in Eq. (3.1).

Now we present the assumption on the convergence conditions of the discriminator. Let $U$ and $G$ be the sets of unlabeled data and generated data.

**Assumption 1. Convergence conditions.** When $D$ converges on a finite training set $\{L, U, G\}$, $D$ learns a (strongly) correct decision boundary for all training data points. More specifically, (1) for any $(x, y) \in L$, we have $w_y^\top h(x) > w_k^\top h(x)$ for any other class $k \neq y$; (2) for any $x \in G$, we have $0 > \max_{k=1}^K w_k^\top h(x)$; (3) for any $x \in U$, we have $\max_{k=1}^K w_k^\top h(x) > 0$.

In Assumption 1 conditions (1) and (2) assume classification correctness on labeled data and true-fake correctness on generated data respectively, which is directly induced by the objective function. Likewise, it is also reasonable to assume true-fake correctness on unlabeled data, i.e., $\log \sum_k \exp w_k^\top h(x) > 0$ for $x \in U$. However, condition (3) goes beyond this and assumes $\max_k w_k^\top h(x) > 0$. We discuss this issue in detail in Section 3.1.3 and argue that these assumptions are reasonable. Moreover, in Section 3.1.5, our approach addresses this issue explicitly by adding a conditional entropy term to the discriminator objective to enforce condition (3).

**Lemma 1.** Suppose for all $k$, the $L_2$-norms of weights $w_k$ are bounded by $\|w_k\|_2 \leq C$. Suppose that there exists $\epsilon > 0$ such that for any $f_G \in F_G$, there exists $f_G' \in G$ such that $\|f_G - f_G'\|_2 \leq \epsilon$. With the conditions in Assumption 1, for all $k \leq K$, we have $w_k^\top f_G < C\epsilon$.

**Proof.** Let $\Delta f = f_G - f_G'$, then we have $\|\Delta f\|_2 \leq \epsilon$. Because $w_k^\top f_G < 0$ by assumption, it follows

$$w_k^\top f_G = w_k^\top (f_G' + \Delta f) = w_k^\top f_G' + w_k^\top \Delta f < w_k^\top \Delta f \leq C\epsilon$$

**Corollary 1.** When unlimited generated data samples are available, with the conditions in Lemma 1 we have $\lim_{|G| \to \infty} w_k^\top f_G \leq 0$. 

49
Proposition 2. Given the conditions in Corollary 1 for all class \( k \leq K \), for all feature space points \( f_k \in F_k \), we have \( w_k^\top f_k > w_j^\top f_k \) for any \( j \neq k \).

Proof. Without loss of generality, suppose \( j = \arg \max_{j \neq k} w_j^\top f_k \). Now we prove it by contradiction. Suppose \( w_k^\top f_k \leq w_j^\top f_k \). Since \( F_k \)'s are disjoint with a margin, \( B \) is a convex set and \( F_G = B - \cup_k F_k \), there exists \( 0 < \alpha < 1 \) such that \( f_G = \alpha f_k + (1 - \alpha) f_j \) with \( f_G \in F_G \) and \( f_j \) being the feature of a labeled data point in \( F_j \). By Corollary 1, it follows that \( w_j^\top f_G \leq 0 \). Thus, \( w_j^\top f_G = \alpha w_j^\top f_k + (1 - \alpha) w_j^\top f_j \leq 0 \). By Assumption 1, \( w_j^\top f_k > 0 \) and \( w_j^\top f_j > 0 \), leading to contradiction. It follows that \( w_k^\top f_k > w_j^\top f_k \) for any \( j \neq k \).

Proposition 2 guarantees that when \( G \) is a complement generator, under mild assumptions, a near-optimal \( D \) learns correct decision boundaries in each high-density subset \( F_k \) (defined by \( \epsilon_k \)) of the data support in the feature space. Intuitively, the generator generates complement samples so the logits of the true classes are forced to be low in the complement. As a result, the discriminator obtains class boundaries in low-density areas. This builds a connection between our approach with manifold-based methods \([8, 179]\) which also leverage the low-density boundary assumption.

With our theoretical analysis, we can now answer the questions raised in Section 2.1.1. First, the \((K + 1)\)-class formulation is effective because the generated complement samples encourage the discriminator to place the class boundaries in low-density areas (Proposition 2). Second, good semi-supervised learning indeed requires a bad generator because a perfect generator is not able to improve the generalization performance (Proposition 1).

On the Feature Space Bound Assumption

To obtain our theoretical results, we assume that \( \bigcup_{k=1}^{K} F_k \) is bounded by a convex set \( B \). And the definition of complement generator requires that \( F_G = B - \bigcup_{k=1}^{K} F_k \). Now we justify the necessity of the introduction of \( B \).

The bounded \( B \) is introduced to ensure that Assumption 1 is realizable. We first show that for Assumption 1 to hold, \( F_G \) must be a convex set.

We define \( S = \{ f : \max_{k=1}^{K} w_k^\top f < 0 \} \).

Lemma 2. \( S \) is a convex set.

Proof. We prove it by contradiction. Suppose \( S \) is a non-convex set, then there exists \( f_1, f_2 \in S \), and \( 0 < \alpha < 1 \), such that \( f = \alpha f_1 + (1 - \alpha) f_2 \notin S \). For all \( k \), we have \( w_k^\top f_1 < 0 \) and \( w_k^\top f_2 < 0 \), and thus it follows

\[
    w_k^\top f = \alpha w_k^\top f_1 + (1 - \alpha) w_k^\top f_2 < 0
\]

Therefore, \( \max_{k=1}^{K} w_k^\top f < 0 \), and we have \( f \in S \), leading to contradiction. We conclude that \( S \) is a convex set.

If the feature space is unbounded and \( F_G \) is defined as \( \mathbb{R}^d - \bigcup_{k=1}^{K} F_k \), where \( d \) is the feature space dimension, then by Assumption 1 we have \( S = F_G \). Since \( F_G \) is the complement set of
\[ \bigcup_{k=1}^{K} F_k \] and \( F_k \)'s are disjoint, \( F_G \) is a non-convex set, if \( K \geq 2 \). However, by Lemma \[2\], \( F_G \) is convex, leading to contradiction. We therefore define the complement generator using a bound \( B \).

**The Reasonableness of Assumption 1**

Here, we justify the proposed Assumption 1.

**Classification correctness on \( L \)** For (1), it assumes the correctness of classification on labeled data \( L \). This only requires the transformation \( h(x) \) to have high enough capacity, such that the limited amount of labeled data points are linearly separable in the feature space. Under the setting of semi-supervised learning, where \( |L| \) is quite limited, this assumption is usually reasonable.

**True-Fake correctness on \( G \)** For (2), it assumes that on generated data, the classifier can correctly distinguish between true and generated data. This can be seen by noticing that \( w_{K+1}^T h(x) = 0 \), and the assumption thus reduces to \( w_{K+1}^T h(x) > \max_{k=1}^{K} w_k^T h(x) \). For this part to hold, again we essentially require a transformation \( h(x) \) with high enough capacity to distinguish true and fake data, which is a standard assumption made in GAN literature.

**Strong true-fake belief on \( U \)** Finally, part (3) of the assumption is a little bit trickier than the other two.

- Firstly, note that (3) is related to the true-fake correctness, because \( \max_{k=1}^{K} w_k^T h(x) > 0 \) is a sufficient (but not necessary) condition for \( x \) being classified as a true data point. Instead, the actual necessary condition is that \( \log \sum_{k=1}^{K} \exp(w_k^T h(x)) \geq w_{K+1}^T h(x) = 0 \). Thus, it means the condition (3) might be violated.

- However, using the relationship \( \log \sum_{k=1}^{K} \exp(w_k^T h(x)) \leq \log K \max_{k=1}^{K} \exp(w_k^T h(x)) \), to guarantee the necessary condition \( \log \sum_{k=1}^{K} \exp(w_k^T h(x)) \geq 0 \), we must have

\[
\log K \max_{k=1}^{K} \exp(w_k^T h(x)) \geq 0
\]

\[ \implies \max_{k=1}^{K} w_k^T h(x) \geq \log 1/K \]

Hence, if the condition (3) is violated, it means

\[
\log 1/K \leq \max_{k=1}^{K} w_k^T h(x) \leq 0
\]

Note that this is a very small interval for the logit \( w_k^T h(x) \), whose possible range expands the entire real line \((-\infty, \infty)\). Thus, the region where such violation happens should be limited in size, making the assumption reasonable in practice.

- Moreover, even there exists a limited violation region, as long as part (1) and part (2) in Assumption 1 hold, Proposition 2 always hold for regions inside \( U \) where \( \max_{k=1}^{K} w_k^T h(x) > 0 \). This can be viewed as a further Corollary.
Empirically, we find that it is easy for the model to satisfy the correctness assumption on labeled data perfectly. To verify the other two assumptions, we keep track of the percentage of test samples that the two assumptions hold under our best models. More specifically, to verify the true-fake correctness on $G$, we calculate the ratio after each epoch

$$ \frac{\sum_{x \sim \mathcal{T}} \mathbb{I}[\max_{i=1}^{K} w_i^T h(x) > 0]}{|\mathcal{T}|}, $$

where $\mathcal{T}$ denotes the test set and $|\mathcal{T}|$ is number of sample in it. Similarly, for the strong true-fake belief on $\mathcal{U}$, we generate the same number of samples as $|\mathcal{T}|$ and calculate

$$ \frac{\sum_{x \sim p_G} \mathbb{I}[\max_{i} w_i^T h(x) < 0]}{|\mathcal{T}|}. $$

The plot is presented in Fig. 3.1. As we can see, the two ratios are both above 0.9 for both SVHN and CIFAR-10, which suggests our assumptions are reasonable in practice.

### 3.1.4 Case Study on Synthetic Data

In the previous section, we have established the fact a complement generator, instead of a perfect generator, is what makes a good semi-supervised learning algorithm. Now, to get a more intuitive understanding, we conduct a case study based on two 2D synthetic datasets, where we can easily verify our theoretical analysis by visualizing the model behaviors. In addition, by analyzing how feature matching (FM) [120] works in 2D space, we identify some potential problems of it, which motivates our approach to be introduced in the next section. Specifically, two synthetic datasets are four spins and two circles, as shown in Fig. 3.2.

**Soundness of complement generator**  Firstly, to verify that the complement generator is a preferred choice, we construct the complement generator by uniformly sampling from the a
Figure 3.2: Labeled and unlabeled data are denoted by cross and point respectively, and different colors indicate classes.

Figure 3.3: Left: Classification decision boundary, where the white line indicates true-fake boundary; Right: True-Fake decision boundary

Figure 3.4: Feature space at convergence

Figure 3.5: Left: Blue points are generated data, and the black shadow indicates unlabeled data. Middle and right can be interpreted as above.

bounded 2D box that contains all unlabeled data, and removing those on the manifold. Based on the complement generator, the result on four spins is visualized in Fig. 3.3. As expected, both the classification and true-fake decision boundaries are almost perfect. More importantly, the classification decision boundary always lies in the fake data area (left panel), which well matches our theoretical analysis.

Visualization of feature space  Next, to verify our analysis about the feature space, we choose the feature dimension to be 2, apply the FM to the simpler dataset of two circles, and visualize the feature space in Fig. 3.4. As we can see, most of the generated features (blue points) resides in between the features of two classes (green and orange crosses), although there exists some overlap. As a result, the discriminator can almost perfectly distinguish between true and generated samples as indicated by the black decision boundary, satisfying the our required Assumption[1]. Meanwhile, the model obtains a perfect classification boundary (blue line) as our analysis suggests.

Pros and cons of feature matching  Finally, to further understand the strength and weakness of FM, we analyze the solution FM reaches on four spins shown in Fig. 3.5. From the left panel, we can see many of the generated samples actually fall into the data manifold, while the rest scatters around in the nearby surroundings of data manifold. It suggests that by matching the first-order moment by SGD, FM is performing some kind of distribution matching, though in a rather weak manner. Loosely speaking, FM has the effect of generating samples close to the manifold. But due to its weak power in distribution matching, FM will inevitably generate samples outside of
the manifold, especially when the data complexity increases. Consequently, the generator density $p_G$ is usually lower than the true data density $p$ within the manifold and higher outside. Hence, an optimal discriminator $P_D^*(K+1 \mid x) = p(x)/(p(x) + p_G(x))$ could still distinguish between true and generated samples in many cases. However, there are two types of mistakes the discriminator can still make:

1. Higher density mistake inside manifold: Since the FM generator still assigns a significant amount of probability mass inside the support, wherever $p_G > p > 0$, an optimal discriminator will incorrectly predict samples in that region as “fake”. Actually, this problem has already shown up when we examine the feature space (Fig. 3.4).

2. Collapsing with missing coverage outside manifold: As the feature matching objective for the generator only requires matching the first-order statistics, there exists many trivial solutions the generator can end up with. For example, it can simply collapse to mean of unlabeled features, or a few surrounding modes as long as the feature mean matches. Actually, we do see such collapsing phenomenon in high-dimensional experiments when FM is used (see Fig. 3.6a and Fig. 3.6c). As a result, a collapsed generator will fail to cover some gap areas between manifolds. Since the discriminator is only well-defined on the union of the data supports of $p$ and $p_G$, the prediction result in such missing area is under-determined and fully relies on the smoothness of the parametric model. In this case, significant mistakes can also occur.

### 3.1.5 Approach

As discussed in previous sections, feature matching GANs suffer from the following drawbacks: 1) the first-order moment matching objective does not prevent the generator from collapsing (missing coverage); 2) feature matching can generate high-density samples inside manifold; 3) the discriminator objective does not encourage realization of condition (3) in Assumption 1 as discussed in Section 3.1.3. Our approach aims to explicitly address the above drawbacks.

Following prior work [43, 120], we employ a GAN-like implicit generator. We first sample a latent variable $z$ from a uniform distribution $U(0, 1)$ for each dimension, and then apply a deep convolutional network to transform $z$ to a sample $x$.

**Generator Entropy**

Fundamentally, the first drawback concerns the entropy of the distribution of generated features, $\mathcal{H}(p_G(f))$. This connection is rather intuitive, as the collapsing issue is a clear sign of low entropy. Therefore, to avoid collapsing and increase coverage, we consider explicitly increasing the entropy.

Although the idea sounds simple and straightforward, there are two practical challenges. Firstly, as implicit generative models, GANs only provide samples rather than an analytic density form. As a result, we cannot evaluate the entropy exactly, which rules out the possibility of naive optimization. More problematically, the entropy is defined in a high-dimensional feature space, which is changing dynamically throughout the training process. Consequently, it is difficult to estimate and optimize the generator entropy in the feature space in a stable and reliable way.
Faced with these difficulties, we consider two practical solutions.

The first method is inspired by the fact that input space is essentially static, where estimating and optimizing the counterpart quantities would be much more feasible. Hence, we instead increase the generator entropy in the input space, i.e., \( H(p_G(x)) \), using a technique derived from an information theoretical perspective and relies on variational inference (VI). Specially, let \( Z \) be the latent variable space, and \( X \) be the input space. We introduce an additional encoder, \( q : \mathcal{X} \rightarrow \mathcal{Z} \), to define a variational upper bound of the negative entropy \( -H(p_G(x)) \leq -\mathbb{E}_{x,z \sim p_G} \log q(z|x) = L_{VI} \). Hence, minimizing the upper bound \( L_{VI} \) effectively increases the generator entropy. In our implementation, we formulate \( q(z|x) \) as a diagonal Gaussian with bounded variance, i.e.
\[
q(z|x) = \mathcal{N}(\mu(x), \sigma^2(x)), \quad 0 < \sigma(x) < \theta,
\]
where \( \mu(\cdot) \) and \( \sigma(\cdot) \) are neural networks, and \( \theta \) is the threshold to prevent arbitrarily large variance.

Alternatively, the second method aims at increasing the generator entropy in the feature space by optimizing an auxiliary objective. Concretely, we adapt the pull-away term (PT) \([175]\) as the auxiliary cost,
\[
L_{PT} = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j \neq i} \left( \frac{h(x_i)^\top h(x_j)}{\|h(x_i)\| \|h(x_j)\|} \right)^2,
\]
where \( N \) is the size of a mini-batch and \( x \) are samples. Intuitively, the pull-away term tries to orthogonalize the features in each mini-batch by minimizing the squared cosine similarity. Hence, it has the effect of increasing the diversity of generated features and thus the generator entropy.

### Generating Low-Density Samples

The second drawback of feature matching GANs is that high-density samples can be generated in the feature space, which is not desirable according to our analysis. Similar to the argument in Section 3.1.5, it is infeasible to directly minimize the density of generated features. Instead, we enforce the generation of samples with low density in the input space. Specifically, given a threshold \( \epsilon \), we minimize the following term as part of our objective:
\[
\mathbb{E}_{x \sim p_G} \log p(x) \mathbb{I}[p(x) > \epsilon]
\]
Equation (3.4)
where \( \mathbb{I}[] \) is an indicator function. Using a threshold \( \epsilon \), we ensure that only high-density samples are penalized while low-density samples are unaffected. Intuitively, this objective pushes the generated samples to “move” towards low-density regions defined by \( p(x) \). To model the probability distribution over images, we simply adapt the state-of-the-art density estimation model for natural images, namely the PixelCNN++ \([121]\) model. The PixelCNN++ model is used to estimate the density \( p(x) \) in Eq. (3.4). The model is pretrained on the training set, and fixed during semi-supervised training.

### Generator Objective and Interpretation

Combining our solutions to the first two drawbacks of feature matching GANs, we have the following objective function of the generator:
\[
\min_G -H(p_G) + \mathbb{E}_{x \sim p_G} \log p(x) \mathbb{I}[p(x) > \epsilon] + \|\mathbb{E}_{x \sim p_G} h(x) - \mathbb{E}_{x \sim U} h(x)\|^2.
\]
Equation (3.5)
This objective is closely related to the idea of complement generator discussed in Section 3.1.3. To see that, let’s first define a target complement distribution in the input space as follows:

\[
p^*(x) = \begin{cases} 
\frac{1}{Z} \frac{1}{p(x)} & \text{if } p(x) > \epsilon \text{ and } x \in B_x \\
C & \text{if } p(x) \leq \epsilon \text{ and } x \in B_x,
\end{cases}
\]

where \( Z \) is a normalizer, \( C \) is a constant, and \( B_x \) is the set defined by mapping \( B \) from the feature space to the input space. With the definition, the KL divergence (KLD) between \( p_G(x) \) and \( p^*(x) \) is

\[
\text{KL}(p_G || p^*) = -\mathcal{H}(p_G) + \mathbb{E}_{x \sim p_G} \log p(x)\mathbb{1}[p(x) > \epsilon] + \mathbb{E}_{x \sim p_G} (\mathbb{1}[p(x) > \epsilon] \log Z - \mathbb{1}[p(x) \leq \epsilon] \log C).
\]

The form of the KLD immediately reveals the aforementioned connection. Firstly, the KLD shares two exactly the same terms with the generator objective equation 3.5. Secondly, while \( p^*(x) \) is only defined in \( B_x \), there is not such a hard constraint on \( p_G(x) \). However, the feature matching term in Eq. equation 3.5 can be seen as softly enforcing this constraint by bringing generated samples “close” to the true data (Cf. Section 3.1.4). Moreover, because the identity function \( \mathbb{1}[] \) has zero gradient almost everywhere, the last term in KLD would not contribute any informative gradient to the generator. In summary, optimizing our proposed objective equation 3.5 can be understood as minimizing the KL divergence between the generator distribution and a desired complement distribution, which connects our practical solution to our theoretical analysis.

**Conditional Entropy**

In order for the complement generator to work, according to condition (3) in Assumption 1, the discriminator needs to have strong true-fake belief on unlabeled data, i.e., \( \max_{k=1}^K w_k^\top h(x) > 0 \). However, the objective function of the discriminator in [120] does not enforce a dominant class. Instead, it only needs \( \sum_{k=1}^K P_D(k|x) > P_D(K + 1|x) \) to obtain a correct decision boundary, while the probabilities \( P_D(k|x) \) for \( k \leq K \) can possibly be uniformly distributed. To guarantee the strong true-fake belief in the optimal conditions, we add a conditional entropy term to the discriminator objective and it becomes

\[
\max_D \mathbb{E}_{x,y \sim \mathcal{L}} \log p_D(y|x, y \leq K) + \mathbb{E}_{x \sim \mathcal{U}} \log p_D(y \leq K|x) + \\
\mathbb{E}_{x \sim p_G} \log p_D(K + 1|x) + \mathbb{E}_{x \sim \mathcal{U}} \sum_{k=1}^K p_D(k|x) \log p_D(k|x).
\] (3.6)

By optimizing Eq. (3.6), the discriminator is encouraged to satisfy condition (3) in Assumption 1. Note that the same conditional entropy term has been used in other semi-supervised learning methods [99, 131] as well, but here we motivate the minimization of conditional entropy based on our theoretical analysis of GAN-based semi-supervised learning.

To train the networks, we alternatively update the generator and the discriminator to optimize Eq. (3.5) and Eq. (3.6) based on mini-batches. If an encoder is used to maximize \( \mathcal{H}(p_G) \), the encoder and the generator are updated at the same time.
Table 3.1: Comparison with state-of-the-art methods on three benchmark datasets. Only methods without data augmentation are included. * indicates using the same (small) discriminator architecture, † indicates using a larger discriminator architecture, and ‡ means self-ensembling.

<table>
<thead>
<tr>
<th>Methods</th>
<th>MNIST (# errors)</th>
<th>SVHN (% errors)</th>
<th>CIFAR-10 (% errors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CatGAN [131]</td>
<td>191 ± 10</td>
<td>-</td>
<td>19.58 ± 0.46</td>
</tr>
<tr>
<td>SDGM [87]</td>
<td>132 ± 7</td>
<td>16.61 ± 0.24</td>
<td>-</td>
</tr>
<tr>
<td>Ladder network</td>
<td>106 ± 37</td>
<td>-</td>
<td>20.40 ± 0.47</td>
</tr>
<tr>
<td>ADGM [87]</td>
<td>96 ± 2</td>
<td>22.86</td>
<td>-</td>
</tr>
<tr>
<td>FM [120] *</td>
<td>93 ± 6.5</td>
<td>8.11 ± 1.3</td>
<td>18.63 ± 2.32</td>
</tr>
<tr>
<td>ALI [35]</td>
<td>-</td>
<td>7.42 ± 0.65</td>
<td>17.99 ± 1.62</td>
</tr>
<tr>
<td>VAT small [99] *</td>
<td>136</td>
<td>6.83</td>
<td>14.87</td>
</tr>
<tr>
<td>Our best model *</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triple GAN [82] ‡</td>
<td>91 ± 58</td>
<td>5.77 ± 0.17</td>
<td>16.99 ± 0.36</td>
</tr>
<tr>
<td>II model [79] †‡</td>
<td>-</td>
<td>5.43 ± 0.25</td>
<td>16.55 ± 0.29</td>
</tr>
<tr>
<td>VAT+EntMin+Large</td>
<td></td>
<td>4.28</td>
<td>13.15</td>
</tr>
</tbody>
</table>

Figure 3.6: Comparing images generated by FM and our model. FM generates collapsed samples, while our model generates diverse “bad” samples.

3.1.6 Experiments

We mainly consider three widely used benchmark datasets, namely MNIST, SVHN, and CIFAR-10. As in previous work, we randomly sample 100, 1,000, and 4,000 labeled samples for MNIST, SVHN, and CIFAR-10 respectively during training, and use the standard data split for testing. We use the 10-quantile log probability to define the threshold $\epsilon$ in Eq. equation 3.5. We add instance noise to the input of the discriminator [8, 129], and use spatial dropout [136] to obtain faster convergence.

Main Results

We compare the the results of our best model with state-of-the-art methods on the benchmarks in Table 3.1. Our proposed methods consistently improve the performance upon feature matching. We achieve new state-of-the-art results on all the datasets when only small discriminator architecture is considered. Our results are also state-of-the-art on MNIST and SVHN among all
Ablation Study

We report the results of ablation study in Table 3.2. In the following, we analyze the effects of several components in our model, subject to the intrinsic features of different datasets.

First, the generator entropy terms (VI and PT) (Section 3.1.5) improve the performance on SVHN and CIFAR by up to 2.2 points in terms of error rate. Moreover, as shown in Fig 3.8, our model significantly reduces the collapsing effects present in the samples generated by FM, which also indicates that maximizing the generator entropy is beneficial. On MNIST, probably due to its simplicity, no collapsing phenomenon was observed with vanilla FM training [120] or in our setting. Under such circumstances, maximizing the generator entropy seems to be unnecessary, and the estimation bias introduced by approximation techniques can even hurt the performance.

Second, the low-density (LD) term is useful when FM indeed generates samples in high-density areas. MNIST is a typical example in this case. When trained with FM, most of the generated hand written digits are highly realistic and have high log probabilities according to the density model (Cf. max log-p in Table 3.2). Hence, when applied to MNIST, LD improves the performance by a clear margin. By contrast, few of the generated SVHN images are realistic (Cf. Fig. 3.6a). Quantitatively, SVHN samples are assigned very low log probabilities (Cf. Table 3.2). As expected, LD has a negligible effect on the performance for SVHN. Moreover, the “max
log-p” column in Table 3.2 shows that while LD can reduce the maximum log probability of the generated MNIST samples by a large margin, it does not yield noticeable difference on SVHN. This further justifies our analysis. Based on the above conclusion, we conjecture LD would not help on CIFAR where sample quality is even lower. Thus, we did not train a density model on CIFAR due to the limit of computational resources.

Third, adding the conditional entropy term has mixed effects on different datasets. While the conditional entropy (Ent) is an important factor of achieving the best performance on SVHN, it hurts the performance on MNIST and CIFAR. One possible explanation relates to the classic exploitation-exploration tradeoff, where minimizing Ent favors exploitation and minimizing the classification loss favors exploration. During the initial phase of training, the discriminator is relatively uncertain and thus the gradient of the Ent term might dominate. As a result, the discriminator learns to be more confident even on incorrect predictions, and thus gets trapped in local minima.

Lastly, we vary the values of the hyper-parameter $\epsilon$ in Eq. (3.5). As shown at the bottom of Table 3.2, reducing $\epsilon$ clearly leads to better performance, which further justifies our analysis in Sections 3.1.4 and 3.1.3 that off-manifold samples are favorable.

**Generated Samples**

We compare the generated samples of FM and our approach in Fig. 3.8. The FM images in Fig. 3.6c are extracted from previous work [120]. While collapsing is widely observed in FM samples, our model generates diverse “bad” images, which is consistent with our analysis.

### 3.2 Semi-Supervised QA by Generating Questions

#### 3.2.1 Motivation

Recently, various neural network models were proposed and successfully applied to the tasks of questions answering (QA) and/or reading comprehension [33, 162, 170]. While achieving state-of-the-art performance, these models rely on a large amount of labeled data. However, it is extremely difficult to collect large-scale question answering datasets. Historically, many of the question answering datasets have only thousands of question answering pairs, such as WebQuestions [111], MCTest [118], WikiQA [165], and TREC-QA [147]. Although larger question answering datasets with hundreds of thousands of question-answer pairs have been collected, including SQuAD [113], MSMARCO [101], and NewsQA [139], the data collection process is expensive and time-consuming in practice. This hinders real-world applications for domain-specific question answering.

Compared to obtaining labeled question answer pairs, it is trivial to obtain unlabeled text data. In this work, we study the following problem of semi-supervised question answering: is it possible to leverage unlabeled text to boost the performance of question answering models, especially when only a small amount of labeled data is available? The problem is challenging because conventional manifold-based semi-supervised learning algorithms [166, 178] cannot be
straightforwardly applied. Moreover, since the main foci of most question answering tasks are extraction rather than generation, it is also not sensible to use unlabeled text to improve language modeling as in machine translation [50].

To better leverage the unlabeled text, we propose a novel neural framework called Generative Domain-Adaptive Nets (GDANs). The starting point of our framework is to use linguistic tags to extract possible answer chunks in the unlabeled text, and then train a generative model to generate questions given the answer chunks and their contexts. The model-generated question-answer pairs and the human-generated question-answer pairs can then be combined to train a question answering model, referred to as a discriminative model in the following text. However, there is discrepancy between the model-generated data distribution and the human-generated data distribution, which leads to suboptimal discriminative models. To address this issue, we further propose two domain adaptation techniques that treat the model-generated data distribution as a different domain. First, we use an additional domain tag to indicate whether a question-answer pair is model-generated or human-generated. We condition the discriminative model on the domain tags so that the discriminative model can learn to factor out domain-specific and domain-invariant representations. Second, we employ a reinforcement learning algorithm to fine-tune the generative model to minimize the loss of the discriminative model in an adversarial way.

In addition, we present a simple and effective baseline method for semi-supervised question answering. Although the baseline method performs worse than our GDAN approach, it is extremely easy to implement and can still lead to substantial improvement when only limited labeled data is available.

3.2.2 Related Work

**Semi-Supervised Learning.** Semi-supervised learning has been extensively studied in literature [177]. A batch of novel models have been recently proposed for semi-supervised learning based on representation learning techniques, such as generative models [70], ladder networks [116] and graph embeddings [166]. However, most of the semi-supervised learning methods are based on combinations of the supervised loss $p(y|x)$ and an unsupervised loss $p(x)$. In the context of reading comprehension, directly modeling the likelihood of a paragraph would not possibly improve the supervised task of question answering. Moreover, traditional graph-based semi-supervised learning [178] cannot be easily extended to modeling the unlabeled answer chunks.

**Domain Adaptation.** Domain adaptation has been successfully applied to various tasks, such as classification [39] and machine translation [19, 63]. Several techniques on domain adaptation [41] focus on learning distribution invariant features by sharing the intermediate representations for downstream tasks. Another line of research on domain adaptation attempt to match the distance between different domain distributions in a low dimensional space [7, 86]. There are also methods seeking a domain transition from the source domain to the target domain [42, 44, 103]. Our work gets inspiration from a practice in [63] and [19] based on appending domain tags. However, our method is different from the above methods in that we apply domain adaptation techniques to the outputs of a generative model rather than a natural data domain.
**Question Answering.** Various neural models based on attention mechanisms [18, 25, 33, 66, 123, 130, 140, 150, 153, 162] have been proposed to tackle the tasks of question answering and reading comprehension. However, the performance of these neural models largely relies on a large amount of labeled data available for training.

**Learning with Multiple Models.** GANs [43] formulated a adversarial game between a discriminative model and a generative model for generating realistic images. Ganin and Lempitsky [39] employed a similar idea to use two models for domain adaptation. Review networks [168] employ a discriminative model as a regularizer for training a generative model. In the context of machine translation, given a language pair, various recent work studied jointly training models to learn the mappings in both directions [141, 160].

### 3.2.3 Problem Definition

Let us first introduce the problem of semi-supervised question answering.

Let \( L = \{q^{(i)}, a^{(i)}, p^{(i)}\}_{i=1}^{N} \) denote a question answering dataset of \( N \) instances, where \( q^{(i)} \), \( a^{(i)} \), and \( p^{(i)} \) are the question, answer, and paragraph of the \( i \)-th instance respectively. The goal of question answering is to produce the answer \( a^{(i)} \) given the question \( q^{(i)} \) along with the paragraph \( p^{(i)} \). We will drop the superscript \( .(i) \) when the context is unambiguous. In our formulation, following the setting in SQuAD [113], we specifically focus on extractive question answering, where \( a \) is always a consecutive chunk of text in \( p \). More formally, let \( p = (p_1, p_2, \ldots, p_T) \) be a sequence of word tokens with \( T \) being the length, then \( a \) can always be represented as \( a = (p_j, p_{j+1}, \ldots, p_{k-1}, p_k) \), where \( j \) and \( k \) are the start and end token indices respectively. The questions can also be represented as a sequence of word tokens \( q = (q_1, q_2, \ldots, q_{T'}) \) with length \( T' \).

In addition to the labeled dataset \( L \), in the semi-supervised setting, we are also given a set of unlabeled data, denoted as \( U = \{a^{(i)}, p^{(i)}\}_{i=1}^{M} \), where \( M \) is the number of unlabeled instances. Note that it is usually trivial to have access to an almost infinite number of paragraphs \( p \) from sources such as Wikipedia articles and other web pages. And since the answer \( a \) is always a consecutive chunk in \( p \), we argue that it is also sensible to extract possible answer chunks from the unlabeled text using linguistic tags. We will discuss the technical details of answer chunk extraction in Section 3.2.5 and in the formulation of our framework, we assume that the answer chunks \( a \) are available.

Given both the labeled data \( L \) and the unlabeled data \( U \), the goal of semi-supervised question answering is to learn a question answering model \( D \) that captures the probability distribution \( \mathbb{P}(a|p, q) \). We refer to this question answering model \( D \) as the **discriminative model**, in contrast to the generative model that we will present in Section 3.2.4.

### A Simple Baseline

We now present a simple baseline for semi-supervised question answering. Given a paragraph \( p = (p_1, p_2, \ldots, p_T) \) and the answer \( a = (p_j, p_{j+1}, \ldots, p_{k-1}, p_k) \), we extract \( (p_{j-W}, p_{j-W+1}, \ldots, p_{j-1}, p_{k+1}, p_{k+2}, p_{k+W}) \) from the paragraph and treat it as the question. Here \( W \) is the window size and is set at 5 in our
experiments so that the lengths of the questions are similar to human-generated questions. The context-based question-answer pairs on \( U \) are combined with human-generated pairs on \( L \) for training the discriminative model. Intuitively, this method extracts the contexts around the answer chunks to serve as hints for the question answering model. Surprisingly, this simple baseline method leads to substantial improvements when labeled data is limited.

3.2.4 Approach

Though the simple method described in Section 3.2.3 can lead to substantial improvement, we aim to design a learning-based model to move even further. In this section, we will describe the model architecture and the training algorithms for the GDANs. We will use a notation in the context of question answering following Section 3.2.3 but one should be able to extend the notion of GDANs to other applications as well.

The GDAN framework consists of two models, a discriminative model and a generative model. We will first discuss the two models in detail in the context of question answering, and then present an algorithm based on reinforcement learning to combine the two models.

Discriminative Model

The discriminative model learns the conditional probability of an answer chunk given the paragraph and the question, i.e., \( P(a|p, q) \). We employ a gated-attention (GA) reader \([33]\) as our base model in this work, but our framework does not make any assumptions about the base models being used. The discriminative model is referred to as \( D \).

The GA model consists of \( K \) layers with \( K \) being a hyper-parameter. Let \( H^k_p \) be the intermediate paragraph representation at layer \( k \), and \( H_q \) be the question representation. The paragraph representation \( H^k_p \) is a \( T \times d \) matrix, and the question representation \( H_q \) is a \( T' \times d \) matrix, where \( d \) is the dimensionality of the representations. Given the paragraph \( p \), we apply a bidirectional Gated Recurrent Unit (GRU) network \([20]\) on top of the embeddings of the sequence \((p_1, p_2, \cdots, p_T)\), and obtain the initial paragraph representation \( H^0_p \). Given the question \( q \), we also apply another bidirectional GRU to obtain the question representation \( H_q \).

The question and paragraph representations are combined with the gated-attention (GA) mechanism \([33]\). More specifically, for each paragraph token \( p_i \), we compute

\[
\alpha_j = \frac{\exp h^T_{q,j} h^k_{p,i}}{\sum_{j'=1}^{T'} \exp h^T_{q,j'} h^k_{p,i}}
\]

\[
h^k_{p,i} = \sum_{j=1}^{T'} \alpha_j h_{q,j} \odot h^{k-1}_{p,i}
\]

where \( h^k_{p,i} \) is the \( i \)-th row of \( H^k_p \) and \( h_{q,j} \) is the \( j \)-th row of \( H_q \).

Since the answer \( a \) is a sequence of consecutive word tokens in the paragraph \( p \), we apply two softmax layers on top of \( H^K_p \) to predict the start and end indices of \( a \), following Yang et al. \([170]\).
**Domain Adaptation with Tags**

We will train our discriminative model on both model-generated question-answer pairs and human-generated pairs. However, even a well-trained generative model will produce questions somewhat different from human-generated ones. Learning from both human-generated data and model-generated data can thus lead to a biased model. To alleviate this issue, we propose to view the model-generated data distribution and the human-generated data distribution as two different data domains and explicitly incorporate domain adaptation into the discriminative model.

More specifically, we use a domain tag as an additional input to the discriminative model. We use the tag “d_true” to represent the domain of human-generated data (i.e., the true data), and “d_gen” for the domain of model-generated data. Following a practice in domain adaptation [19, 63], we append the domain tag to the end of both the questions and the paragraphs. By introducing the domain tags, we expect the discriminative model to factor out domain-specific and domain-invariant representations. At test time, the tag “d_true” is appended.

**Generative Model**

The generative model learns the conditional probability of generating a question given the paragraph and the answer, i.e., \(P(q|p, a)\). We implement the generative model as a sequence-to-sequence model [132] with a copy mechanism [49, 51].

The generative model consists of an encoder and a decoder. An encoder is a GRU that encodes the input paragraph into a sequence of hidden states \(H\). We inject the answer information by appending an additional zero/one feature to the word embeddings of the paragraph tokens; i.e., if a word token appears in the answer, the feature is set at one, otherwise zero.

The decoder is another GRU with an attention mechanism over the encoder hidden states \(H\). At each time step, the generation probabilities over all word types are defined with a copy mechanism:

\[
P_{\text{overall}} = g_t P_{\text{vocab}} + (1 - g_t) P_{\text{copy}}
\]

where \(g_t\) is the probability of generating the token from the vocabulary, while \((1 - g_t)\) is the probability of copying a token from the paragraph. The probability \(g_t\) is computed based on the current hidden state \(h_t\):

\[
g_t = \sigma(w_g^T h_t)
\]

where \(\sigma\) denotes the logistic function and \(w_g\) is a vector of model parameters. The generation probabilities \(P_{\text{vocab}}\) are defined as a softmax function over the word types in the vocabulary, and the copying probabilities \(P_{\text{copy}}\) are defined as a softmax function over the word types in the paragraph. Both \(P_{\text{vocab}}\) and \(P_{\text{copy}}\) are defined as a function of the current hidden state \(h_t\) and the attention results [49].

**Training Algorithm**

We first define the objective function of the GDANs, and then present an algorithm to optimize the given objective function. Similar to the Generative Adversarial Nets (GANs) [43] and adversarial domain adaptation [39], the discriminative model and the generative model have...
Algorithm 1 Training Generative Domain-Adaptive Nets

**Input:** labeled data \( L \), unlabeled data \( U \), #iterations \( T_G \) and \( T_D \)

Initialize \( G \) by MLE training on \( L \)

Randomly initialize \( D \)

**while** not stopping **do**

for \( t \leftarrow 1 \) to \( T_D \) **do**

Update \( D \) to maximize \( J(L, d_{true}, D) + J(U_G, d_{gen}, D) \) with SGD

**end for**

for \( t \leftarrow 1 \) to \( T_G \) **do**

Update \( G \) to maximize \( J(U_G, d_{true}, D) \) with Reinforce and SGD

**end for**

**end while**

**return** model \( D \)

---

**Figure 3.7:** Model architecture and training. Red boxes denote the modules being updated. “d_true” and “d_gen” are two domain tags. \( D \) is the discriminative model and \( G \) is the generative model. The objectives for the three cases are all to minimize the cross entropy loss of the answer chunks.

---

Different objectives in our framework. However, rather than formulating the objective as an adversarial game between the two models [39, 43], in our framework, the discriminative model relies on the data generated by the generative model, while the generative model aims to match the model-generated data distribution with the human-generated data distribution using the signals from the discriminative model.

Given a labeled dataset \( L = \{p^{(i)}, q^{(i)}, a^{(i)}\}_{i=1}^{N} \), the objective function of a discriminative model \( D \) for a supervised learning setting can be written as:

\[
J(L, \text{tag}, D) = \frac{1}{|L|} \sum_{p^{(i)}, q^{(i)}, a^{(i)} \in L} \log \mathbb{P}_D(a^{(i)} | p^{(i)}, q^{(i)})
\]

where \( \mathbb{P}_D \) is a probability distribution defined by the model \( D \). Since we also incorporate domain tags into the model \( D \), we denote the objective function as:

\[
J(L, \text{tag}, D) = \frac{1}{|L|} \sum_{p^{(i)}, q^{(i)}, a^{(i)} \in L} \log \mathbb{P}_{D, \text{tag}}(a^{(i)} | p^{(i)}, q^{(i)})
\]
meaning that the domain tag, “tag”, is appended to the dataset $L$. We use $|L| = N$ to denote the number of the instances in the dataset $L$. The objective function is averaged over all instances such that we can balance labeled and unlabeled data.

Let $U_G$ denote the dataset obtained by generating questions on the unlabeled dataset $U$ with the generative model $G$. The objective of the discriminative model is then to maximize $J$ for both labeled and unlabeled data under the domain adaptation notions, i.e., $J(L, d_{true}, D) + J(U_G, d_{gen}, D)$.

Now we discuss the objective of the generative model. Similar to the dual learning framework, one can define an auto-encoder objective. In this case, the generative model aims to generate questions that can be reconstructed by the discriminative model, i.e., maximizing $J(U_G, d_{gen}, D)$. However, this objective function can lead to degenerate solutions because the questions can be thought of as an overcomplete representation of the answers. For example, given $p$ and $a$, the generative model might learn to generate trivial questions such as copying the answers, which does not contributed to learning a better $D$.

Instead, we leverage the discriminative model to better match the model-generated data distribution with the human-generated data distribution. We propose to define an adversarial training objective $J(U_G, d_{true}, D)$. We append the tag “d_true” instead of “d_gen” for the model-generated data to “fool” the discriminative model. Intuitively, the goal of $G$ is to generate "useful" questions where the usefulness is measured by the probability that the generated questions can be answered correctly by $D$.

The overall objective function now can be written as

$$\max_D J(L, d_{true}, D) + J(U_G, d_{gen}, D)$$

$$\max_G J(U_G, d_{true}, D)$$

With the above objective function in mind, we present a training algorithm in Algorithm 1 to train a GDAN. We first pretrain the generative model on the labeled data $L$ with maximum likelihood estimation (MLE): 

$$\max_G \sum_{i=1}^N \sum_{t=1}^{T'} \log P_G(q_t^{(i)}|q_{<t}^{(i)}, p^{(i)}, a^{(i)})$$

where $P_G$ is the probability defined by Eq. 3.7.

We then alternatively update $D$ and $G$ based on their objectives. To update $D$, we sample one batch from the labeled data $L$ and one batch from the unlabeled data $U_G$, and combine the two batches to perform a gradient update step. Since the output of $G$ is discrete and non-differentiable, we use the Reinforce algorithm to update $G$. The action space is all possible questions with length $T'$ (possibly with padding) and the reward is the objective function $J(U_G, d_{true}, D)$. Let $\theta_G$ be the parameters of $G$. The gradient can be written as

$$\frac{\partial J(U_G, d_{true}, D)}{\partial \theta_G} = E_{P_G(q|p,a)}(\log P_D, d_{true}(a|p, q) - b) \frac{\partial \log P_G(q|p, a)}{\partial \theta_G}$$
where we use an average reward from samples as the baseline $b$. We approximate the expectation $\mathbb{E}_{P_G(q|p,a)}$ by sampling one instance at a time from $P_G(q|p,a)$ and then do an update step. This training algorithm is referred to as reinforcement learning (RL) training in the following sections. The overall architecture and training algorithm are illustrated in Figure 3.10.

**MLE vs RL.** The generator $G$ has two training phases—MLE training and RL training, which are different in that: 1) RL training does not require labels, so $G$ can explore a broader data domain of $p$ using unlabeled data, while MLE training requires labels; 2) MLE maximizes $\log P(q|p,a)$, while RL maximizes $\log P_{D}(a|q,p)$. Since $\log P(q|a,p)$ is the sum of $\log P(q|p)$ and $\log P(a|q,p)$ (plus a constant), maximizing $\log P(a|q,p)$ does not require modeling $\log P(q|p)$ that is irrelevant to QA, which makes optimization easier. Moreover, maximizing $\log P(a|q,p)$ is consistent with the goal of QA.

### 3.2.5 Experiments

**Answer Extraction**

As discussed in Section 3.2.3, our model assumes that answers are available for unlabeled data. In this section, we introduce how we use linguistic tags and rules to extract answer chunks from unlabeled text.

To extract answers from massive unlabelled Wikipedia articles, we first sample 205,511 Wikipedia articles that are not used in the training, development and test sets in the SQuAD dataset. We extract the paragraphs from each article, and limit the length of each paragraph at the word level to be less than 850. In total, we obtain 950,612 paragraphs from unlabelled articles.

Answers in the SQuAD dataset can be categorized into ten types, i.e., “Date”, “Other Numeric”, “Person”, “Location”, “Other Entity”, “Common Noun Phrase”, “Adjective Phrase”, “Verb Phrase”, “Clause” and “Other” [113]. For each paragraph from the unlabeled articles, we utilize Stanford Part-Of-Speech (POS) tagger [137] to label each word with the corresponding POS tag, and implement a simple constituency parser to extract the noun phrase, verb phrase, adjective and clause based on a small set of constituency grammars. Next, we use Stanford Named Entity Recognizer (NER) [37] to assign each word with one of the seven labels, i.e., “Date”, “Money”, “Percent”, “location”, “Organization” and “Time”. We then categorize a span of consecutive words with the same NER tags of either “Money” or “Percent” as the answer of the type “Other Numeric”. Similarly, we categorize a span of consecutive words with the same NER tags of “Organization” as the answer of the type “Other Entity”. Finally, we subsample five answers from all the extracted answers for each paragraph according to the percentage of answer types in the SQuAD dataset. We obtain 4,753,060 answers in total, which is about 50 times larger than the number of answers in the SQuAD dataset.

**Settings and Comparison Methods**

The original SQuAD dataset consists of 87,636 training instances and 10,600 development instances. Since the test set is not published, we split 10% of the training set as the test set, and the remaining 90% serves as the actual training set. Instances are split based on articles; i.e.,
paragraphs in one article always appear in only one set. We tune the hyper-parameters and perform early stopping on the development set using the F1 scores, and the performance is evaluated on the test set using both F1 scores and exact matching (EM) scores [113].

We compare the following methods. SL is the supervised learning setting where we train the model \( D \) solely on the labeled data \( L \). Context is the simple context-based method described in Section 3.2.3. Context + domain is the “Context” method with domain tags as described in Section 3.2.4. Gen is to train a generative model and use the generated questions as additional training data. Gen + GAN refers to the domain adaptation method using GANs [39]; in contrast to the original work, the generative model is updated using Reinforce. Gen + dual refers to the dual learning method [160]. Gen + domain is “Gen” with domain tags, while the generative model is trained with MLE and fixed. Gen + domain + adv is the approach we propose (Cf. Figure 3.10 and Algorithm 1), with “adv” meaning adversarial training based on Reinforce. We use our own implementation of “Gen + GAN” and “Gen + dual”, since the GAN model [39] does not handle discrete features and the dual learning model [160] cannot be directly applied to question answering. When implementing these two baselines, we adopt the learning schedule introduced by Ganin and Lempitsky [39], i.e., gradually increasing the weights of the gradients for the generative model \( G \).

Results and Analysis

We study the performance of different models with varying labeling rates and unlabeled dataset sizes. Labeling rates are the percentage of training instances that are used to train \( D \). The results are reported in Table 3.4. Though the unlabeled dataset we collect consists of around 5 million instances, we also sample a subset of around 50,000 instances to evaluate the effects of the size of unlabeled data. The highest labeling rate in Table 3.4 is 0.9 because 10% of the training instances are used for testing. Since we do early stopping on the development set using the F1 scores, we also report the development F1. We report two metrics, the F1 scores and the exact matching (EM) scores [113], on the test set. All metrics are computed using the official evaluation scripts.

SL v.s. SSL. We observe that semi-supervised learning leads to consistent improvements over supervised learning in all cases. Such improvements are substantial when labeled data is limited. For example, the GDANs improve over supervised learning by 9.87 points in F1 and 7.26 points in EM when the labeling rate is 0.1. With our semi-supervised learning approach, we can use only 0.1 training instances to obtain even better performance than a supervised learning approach with 0.2 training instances, saving more than half of the labeling costs.

Comparison with Baselines. By comparing “Gen + domain + adv” with “Gen + GAN” and “Gen + Dual”, it is clear that the GDANs perform substantially better than GANs and dual learning. With labeling rate 0.1, GDANs outperform dual learning and GANs by 2.47 and 4.29 points respectively in terms of F1.

Ablation Study. We also perform an ablation study by examining the effects of “domain” and “adv” when added to “gen”. It can be seen that both the domain tags and the adversarial training contribute to the performance of the GDANs when the labeling rate is equal to or less than 0.5. With labeling rate 0.9, adding domain tags still leads to better performance but adversarial training does not seem to improve the performance by much.
Figure 3.8: Comparison of discriminator training loss $-J(U_G, d_{gen}, D)$ on generated QA pairs. The lower the better. MLE refers to questions generated by maximum likelihood training, and RL refers to questions generated by reinforcement learning.

Unlabeled Data Size. Moreover, we observe that the performance can be further improved when a larger unlabeled dataset is used, though the gain is relatively less significant compared to changing the model architectures. For example, increasing the unlabeled dataset size from 50K to 5M, the performance of GDANs increases by 0.38 points in F1 and 0.52 points in EM.

Context-Based Method. Surprisingly, the simple context-based method, though performing worse than GDANs, still leads to substantial gains; e.g., 7.00 points in F1 with labeling rate 0.1. Adding domain tags can improve the performance of the context-based method as well.

MLE vs RL. We plot the loss curve of $-J(U_G, d_{gen}, D)$ for both the MLE-trained generator (“Gen + domain”) and the RL-trained generator (“Gen + domain + adv”) in Figure 3.8. We observe that the training loss for D on RL-generated questions is lower than MLE-generated questions, which confirms that RL training maximizes $\log P(a|p, q)$.

Samples of Generated Questions. We present some questions generated by our model in Table 3.3. The generated questions are post-processed by removing repeated sub-sequences. Compared to MLE-generated questions, RL-generated questions are more informative (Cf., P1, P2, and P4), and contain less “UNK” (unknown) tokens (Cf., P1). Moreover, both semantically and syntactically, RL-generated questions are more accurate (Cf., P3 and P5).
Table 3.3: Sampled generated questions given the paragraphs and the answers. *P* means paragraphs, *A* means answers, *GQ* means groundtruth questions, and *Q* means questions generated by our models. *MLE* refers to maximum likelihood training, and *RL* refers to reinforcement learning so as to maximize $J(U, d_{true}, D)$. We truncate the paragraphs to only show tokens around the answer spans with a window size of 20.

| P1: is mediated by IgE, which triggers degranulation of mast cells and basophils when cross-linked by antigen. Type II hypersensitivity occurs when antibodies bind to antigens on the patient’s own cells, marking them for destruction. This | A: Type II hypersensitivity  
GQ: Antibody-dependent hypersensitivity belongs to what class of hypersensitivity?  
Q (MLE): What was the UNK of the patient’s own cells?  
Q (RL): What occurs when antibodies bind to antigens on the patient’s own cells by antigen when cross- 

| P2: An additional warming of the earth’s surface. They calculate with confidence that CO2 has been responsible for over half the enhanced greenhouse effect. They predict that under a “business as usual” (BAU) scenario, | A: Over half  
GQ: How much of the greenhouse effect is due to carbon dioxide?  
Q (MLE): What is the enhanced greenhouse effect?  
Q (RL): What the enhanced greenhouse effect that CO2 been responsible for 

| P3: Narrow gauge lines, which are the remnants of five formerly government-owned lines which were built in mountainous areas. | A: Mountainous areas  
GQ: Where were the narrow gauge rail lines built in Victoria?  
Q (MLE): What is the government government government-owned lines built?  
Q (RL): What were the remnants of government-owned lines built in 

| P4: But not both). In 0000, BankAmericard was renamed and spun off into a separate company known today as Visa Inc. | A: Visa Inc.  
GQ: What present-day company did BankAmericard turn into?  
Q (MLE): What was the separate company BankAmericard?  
Q (RL): What today as BankAmericard off into a separate company known today as spun off into a separate company known today 

| P5: Legrande writes that “the formulation of a single all-encompassing definition of the term is extremely difficult, if | A: Legrande  
GQ: Who wrote that it is difficult to produce an all-inclusive definition of civil disobedience?  
Q (MLE): What is the term of a single all all all all encompassing definition of a single all  
Q (RL): What writes "the formulation of a single all - encompassing definition of the term all encompassing encompassing encompassing encompassing
Table 3.4: Performance with various labeling rates, unlabeled data sizes $|U|$, and methods. “Dev” denotes the development set, and “test” denotes the test set. F1 and EM are two metrics.

| Labeling rate | $|U|$ | Method             | Dev F1 | Test F1 | Test EM |
|---------------|------|--------------------|--------|---------|---------|
| 0.1           | 50K  | SL                 | 0.4262 | 0.3815  | 0.2492  |
| 0.1           | 50K  | Context            | 0.5046 | 0.4515  | 0.2966  |
| 0.1           | 50K  | Context + domain   | 0.5139 | 0.4575  | 0.3036  |
| 0.1           | 50K  | Gen                | 0.5049 | 0.4553  | 0.3018  |
| 0.1           | 50K  | Gen + GAN          | 0.4897 | 0.4373  | 0.2885  |
| 0.1           | 50K  | Gen + dual         | 0.5036 | 0.4555  | 0.3005  |
| 0.1           | 50K  | Gen + domain       | 0.5234 | 0.4703  | 0.3145  |
| 0.1           | 50K  | Gen + domain + adv | 0.5313 | 0.4802  | 0.3218  |
| 0.2           | 50K  | SL                 | 0.5134 | 0.4674  | 0.3163  |
| 0.2           | 50K  | Context            | 0.5652 | 0.5132  | 0.3573  |
| 0.2           | 50K  | Context + domain   | 0.5672 | 0.5200  | 0.3581  |
| 0.2           | 50K  | Gen                | 0.5643 | 0.5159  | 0.3618  |
| 0.2           | 50K  | Gen + GAN          | 0.5525 | 0.5037  | 0.3470  |
| 0.2           | 50K  | Gen + dual         | 0.5720 | 0.5192  | 0.3612  |
| 0.2           | 50K  | Gen + domain       | 0.5749 | 0.5216  | 0.3658  |
| 0.2           | 50K  | Gen + domain + adv | 0.5867 | 0.5394  | 0.3781  |
| 0.5           | 50K  | SL                 | 0.6280 | 0.5722  | 0.4187  |
| 0.5           | 50K  | Context            | 0.6300 | 0.5740  | 0.4195  |
| 0.5           | 50K  | Context + domain   | 0.6307 | 0.5791  | 0.4237  |
| 0.5           | 50K  | Gen                | 0.6237 | 0.5717  | 0.4155  |
| 0.5           | 50K  | Gen + GAN          | 0.6110 | 0.5590  | 0.4044  |
| 0.5           | 50K  | Gen + dual         | 0.6368 | 0.5746  | 0.4163  |
| 0.5           | 50K  | Gen + domain       | 0.6378 | 0.5826  | 0.4261  |
| 0.5           | 50K  | Gen + domain + adv | 0.6375 | 0.5831  | 0.4267  |
| 0.9           | 50K  | SL                 | 0.6611 | 0.6070  | 0.4534  |
| 0.9           | 50K  | Context            | 0.6560 | 0.6028  | 0.4507  |
| 0.9           | 50K  | Context + domain   | 0.6553 | 0.6105  | 0.4557  |
| 0.9           | 50K  | Gen                | 0.6464 | 0.5970  | 0.4445  |
| 0.9           | 50K  | Gen + GAN          | 0.6396 | 0.5874  | 0.4317  |
| 0.9           | 50K  | Gen + dual         | 0.6511 | 0.5892  | 0.4340  |
| 0.9           | 50K  | Gen + domain       | 0.6611 | 0.6102  | 0.4573  |
| 0.9           | 50K  | Gen + domain + adv | 0.6585 | 0.6043  | 0.4497  |
| 0.1           | 5M   | SL                 | 0.4262 | 0.3815  | 0.2492  |
| 0.1           | 5M   | Context            | 0.5140 | 0.4641  | 0.3014  |
| 0.1           | 5M   | Context + domain   | 0.5166 | 0.4599  | 0.3083  |
| 0.1           | 5M   | Gen                | 0.5099 | 0.4619  | 0.3103  |
| 0.1           | 5M   | Gen + domain       | 0.5301 | 0.4703  | 0.3227  |
| 0.1           | 5M   | Gen + domain + adv | 0.5442 | 0.4840  | 0.3270  |
| 0.9           | 5M   | SL                 | 0.6611 | 0.6070  | 0.4534  |
| 0.9           | 5M   | Context            | 0.6605 | 0.6026  | 0.4473  |
| 0.9           | 5M   | Context + domain   | 0.6642 | 0.6066  | 0.4548  |
| 0.9           | 5M   | Gen                | 0.6647 | 0.6065  | 0.4600  |
| 0.9           | 5M   | Gen + domain       | 0.6726 | 0.6092  | 0.4599  |
| 0.9           | 5M   | Gen + domain + adv | 0.6670 | 0.6102  | 0.4531  |
3.3 Semi-Supervised Learning on Graphs

In this section, we consider semi-supervised learning on graphs. We define a generative process to generate the random walk paths on a graph, and this generative modeling objective is jointly trained with the supervised learning objective for classification on graphs.

3.3.1 Motivation

A large number of semi-supervised learning algorithms jointly optimize two training objective functions: the supervised loss over labeled data and the unsupervised loss over both labeled and unlabeled data. Graph-based semi-supervised learning defines the loss function as a weighted sum of the supervised loss over labeled instances and a graph Laplacian regularization term \([8, 155, 176, 179]\). The graph Laplacian regularization is based on the assumption that nearby nodes in a graph are likely to have the same labels. Graph Laplacian regularization is effective because it constrains the labels to be consistent with the graph structure.

Recently developed unsupervised representation learning methods learn embeddings that predict a distributional context, e.g. a word embedding might predict nearby context words \([96, 107]\), or a node embedding might predict nearby nodes in a graph \([108, 134]\). Embeddings trained with distributional context can be used to boost the performance of related tasks. For example, word embeddings trained from a language model can be applied to part-of-speech tagging, chunking and named entity recognition \([22, 167]\).

In this paper we consider not word embeddings but graph embeddings. Existing results show that graph embeddings are effective at classifying the nodes in a graph, such as user behavior prediction in a social network \([108, 134]\). However, the graph embeddings are usually learned separately from the supervised task, and hence do not leverage the label information in a specific task. Hence graph embeddings are in some sense complementary to graph Laplacian regularization that does not produce useful features itself and might not be able to fully leverage the distributional information encoded in the graph structure.

The main highlight of our work is to incorporate embedding techniques into the graph-based semi-supervised learning setting. We propose a novel graph-based semi-supervised learning framework, Planetoid (Predicting Labels And Neighbors with Embeddings Transductively Or Inductively from Data).

3.3.2 Background

Graph-Based Semi-Supervised Learning

Let \(L\) and \(U\) be the number of labeled and unlabeled instances. Let \(x_{1:L}\) and \(x_{L+1:L+U}\) denote the feature vectors of labeled and unlabeled instances respectively. The labels \(y_{1:L}\) are also given. Based on both labeled and unlabeled instances, the problem of semi-supervised learning is defined as learning a classifier \(f : x \rightarrow y\). There are two learning paradigms, transductive learning and inductive learning. Transductive learning \([176, 179]\) only aims to apply the classifier \(f\)
on the unlabeled instances observed at training time, and the classifier does not generalize to unobserved instances. For instance, transductive support vector machine (TSVM) [62] maximizes the “unlabeled data margin” based on the low-density separation assumption that a good decision hyperplane lies on a sparse area of the feature space. Inductive learning [8, 155], on the other hand, aims to learn a parameterized classifier \( f \) that is generalizable to unobserved instances.

In addition to labeled and unlabeled instances, a graph, denoted as a \((L + U) \times (L + U)\) matrix \( A \), is also given to graph-based semi-supervised learning methods. Each entry \( a_{ij} \) indicates the similarity between instance \( i \) and \( j \), which can be either labeled or unlabeled. The graph \( A \) can either be derived from distances between instances [179], or be explicitly derived from external data, such as a knowledge graph [157] or a citation network between documents [60]. In this paper, we mainly focus on the setting that a graph is explicitly given and represents additional information not present in the feature vectors (e.g., the graph edges correspond to hyperlinks between documents, rather than distances between the bag-of-words representation of a document).

Graph-based semi-supervised learning is based on the assumption that nearby nodes tend to have the same labels. Generally, the loss function of graph-based semi-supervised learning in the binary case can be written as

\[
\sum_{i=1}^{L} l(y_i, f(x_i)) + \lambda \sum_{i,j} a_{ij} \left\| f(x_i) - f(x_j) \right\|^2
\]

In Eq. (3.8), the first term is the standard supervised loss function, where \( l(\cdot, \cdot) \) can be log loss, squared loss or hinge loss. The second term is the graph Laplacian regularization, which incurs a large penalty when similar nodes with a large \( w_{ij} \) are predicted to have different labels \( f(x_i) \neq f(x_j) \). The graph Laplacian matrix \( \Delta \) is defined as \( \Delta = A - D \), where \( D \) is a diagonal matrix with each entry defined as \( d_{ii} = \sum_j a_{ij} \). \( \lambda \) is a constant weighting factor. (Note that we omit the parameter regularization terms for simplicity.) Various graph-based semi-supervised learning algorithms define the loss functions as variants of Eq. (3.8). Label propagation [179] forces \( f \) to agree with labeled instances \( y_1:L \); \( f \) is a label lookup table for unlabeled instances in the graph, and can be obtained with a closed-form solution. Learning with local and global consistency [176] defines \( l \) as squared loss and \( f \) as a label lookup table; it does not force \( f \) to agree with labeled instances. Modified Adsorption (MAD) [133] is a variant of label propagation that allows prediction on labeled instances to vary and incorporates node uncertainty. Manifold regularization [8] parameterizes \( f \) in the Reproducing Kernel Hilbert Space (RKHS) with \( l \) being squared loss or hinge loss. Since \( f \) is a parameterized classifier, manifold regularization is inductive and can naturally handle unobserved instances.

Semi-supervised embedding [155] extends the regularization term in Eq. (3.8) to be \( \sum_{i,j} a_{ij} \left\| g(x_i) - g(x_j) \right\|^2 \), where \( g \) represents embeddings of instances, which can be the output labels, hidden layers or auxiliary embeddings in a neural network. By extending the regularization from \( f \) to \( g \), this method imposes stronger constraints on a neural network. Iterative classification algorithm
ICA) uses a local classifier that takes the labels of neighbor nodes as input, and employs an iterative process between estimating the local classifier and assigning new labels.

## Learning Embeddings

Extensive research was done on learning graph embeddings. A probabilistic generative model was proposed to learn node embeddings that generate the edges in a graph. A clustering method was proposed to learn latent social states in a social network to predict social ties.

More recently, a number of embedding learning methods are based on the Skipgram model, which is a variant of the softmax model. Given an instance and its context, the objective of Skipgram is usually formulated as minimizing the log loss of predicting the context using the embedding of an instance as input features. Formally, let \( \{(i, c)\} \) be a set of pairs of instance \( i \) and context \( c \), the loss function can be written as

\[
- \sum_{(i,c)} \log p(c|i) = - \sum_{(i,c)} \left( w^T_c e_i - \log \sum_{c' \in C} \exp(w^T_c e_i) \right)
\]

(3.9)

where \( C \) is the set of all possible context, \( w \)'s are parameters of the Skipgram model, and \( e_i \) is the embedding of instance \( i \). Skipgram was first introduced to learn representations of words, known as word2vec. In word2vec, for each training pair \((i, c)\), the instance \( i \) is the current word whose embedding is under estimation; the context \( c \) is each of the surrounding words of \( i \) within a fixed window size in a sentence; the context space \( C \) is the vocabulary of the corpus. Skipgram was later extended to learn graph embeddings. Deepwalk uses the embedding of a node to predict the context in the graph, where the context is generated by random walk. More specifically, for each training pair \((i, c)\), the instance \( i \) is the current node whose embedding is under estimation; the context \( c \) is each of the neighbor nodes within a fixed window size in a generated random walk sequence; the context space \( C \) is all the nodes in the graph. LINE extends the model to have multiple context spaces \( C \) for modeling both first and second order proximity.

Although Skipgram-like models for graphs have received much recent attention, many other models exist. TransE learns the embeddings of entities in a knowledge graph jointly with their relations. Autoencoders were used to learn graph embeddings for clustering on graphs.

### 3.3.3 Approach

Following the notations in the previous section, the input to our method includes labeled instances \( x_{1:L}, y_{1:L} \), unlabeled instances \( x_{L+1:L+U} \) and a graph denoted as a matrix \( A \). Each instance \( i \) has an embedding denoted as \( e_i \).

We formulate our framework based on feed-forward neural networks. Given the input feature vector \( x \), the \( k \)-th hidden layer of the network is denoted as \( h^k \), which is a nonlinear function of the previous hidden layer \( h^{k-1} \) defined as: \( h^k(x) = \text{ReLU}(W^k h^{k-1}(x) + b^k) \), where \( W^k \) and \( b^k \) are parameters of the \( k \)-th layer, and \( h^0(x) = x \). We adopt rectified linear unit \( \text{ReLU}(x) = \max(0, x) \) as the nonlinear function in this work.
Figure 3.9: An example of sampling from context distribution $p(i, c, \gamma)$ when $\gamma = 1$ and $d = 2$. In circles, $+1$ denotes positive instances, $-1$ denotes negative instances, and $?$ denotes unlabeled instances. If $\text{random} < r_2$, we first sample a random walk $2 \rightarrow 1 \rightarrow 4 \rightarrow 6$, and then sample two nodes in the random walk within distance $d$. If $\text{random} \geq r_2$, we sample two instances with the same labels.

The loss function of our framework can be expressed as

$$L_s + \lambda L_u,$$

where $L_s$ is a supervised loss of predicting the labels, and $L_u$ is an unsupervised loss of predicting the graph context. In the following sections, we first formulate $L_u$ by introducing how to sample context from the graph, and then formulate $L_s$ to form our semi-supervised learning framework.

**Sampling Context**

We formulate the unsupervised loss $L_u$ as a variant of Eq. (3.9). Given a graph $A$, the basic idea of our approach is to sample pairs of instance $i$ and context $c$, and then formulate the loss $L_u$ using the log loss $- \log p(c|i)$ as in Eq. (3.9). We first present the formulation of $L_u$ by introducing negative sampling, and then discuss how to sample pairs of instance and context.

It is usually intractable to directly optimize Eq. (3.9) due to normalization over the whole context space $C$. Negative sampling was introduced to address this issue [96], which samples negative examples to approximate the normalization term. In our case, we are sampling $(i, c, \gamma)$ from a distribution, where $i$ and $c$ denote instance and context respectively, $\gamma = +1$ means $(i, c)$ is a positive pair and $\gamma = -1$ means negative. Given $(i, c, \gamma)$, we minimize the cross entropy loss of classifying the pair $(i, c)$ to a binary label $\gamma$:

$$-\mathbb{I}(\gamma = 1) \log \sigma(w_e^T c_i) - \mathbb{I}(\gamma = -1) \log \sigma(-w_e^T c_i),$$

where $\sigma$ is the sigmoid function defined as $\sigma(x) = 1/(1 + e^{-x})$, and $\mathbb{I}()$ is an indicator function that outputs 1 when the argument is true, otherwise 0. Therefore, the unsupervised loss with
Algorithm 2 Sampling Context Distribution $p(i, c, \gamma)$

**Input:** graph $A$, labels $y_1 : L$, parameters $r_1, r_2, q, d$

1. Initialize triplet $(i, c, \gamma)$
2. If $\text{random} < r_1$ then $\gamma \leftarrow +1$ else $\gamma \leftarrow -1$
3. If $\text{random} < r_2$ then
   1. Uniformly sample a random walk $S$ of length $q$
   2. Uniformly sample $(S_j, S_k)$ with $|j - k| < d$
   3. $i \leftarrow S_j$, $c \leftarrow S_k$
   4. If $\gamma = -1$ then uniformly sample $c$ from $1 : L + U$
      else
      1. If $\gamma = +1$ then
         Uniformly sample $(i, c)$ with $y_i = y_c$
      else
         Uniformly sample $(i, c)$ with $y_i \neq y_c$
5. end if
6. end if
7. Return $(i, c, \gamma)$

Negative sampling can be written as

$$L_u = -\mathbb{E}_{(i, c, \gamma)} \log \sigma(\gamma w_c^T e_i)$$

(3.10)

The distribution $p(i, c, \gamma)$ is conditioned on labels $y_1 : L$ and the graph $A$. However, since they are the input to our algorithm and kept fixed, we drop the conditioning in our notation.

We now define the distribution $p(i, c, \gamma)$ directly using a sampling process, which is illustrated in Algorithm 2. There are two types of context that are sampled in this algorithm. The first type of context is based on the graph $A$, which encodes the structure (distributional) information, and the second type of context is based on the labels, which we use to inject label information into the embeddings. We use a parameter $r_1 \in (0, 1)$ to control the ratio of positive and negative samples, and use $r_2 \in (0, 1)$ to control the ratio of two types of context.

With probability $r_2$, we sample the context based on the graph $A$. We first uniformly sample a random walk sequence $S$. More specifically, we uniformly sample the first instance $S_1$ from the set $1 : L + U$. Given the previous instance $S_{k-1} = i$, the next instance $S_k = j$ is sampled with probability $a_{ij} / \sum_{j' = 1}^{L+U} a_{ij'}$. With probability $r_1$, we sample a positive pair $(i, c)$ from the set $\{(S_j, S_k) : |j - k| < d\}$, where $d$ is another parameter determining the window size. With probability $(1 - r_1)$, we uniformly corrupt the context $c$ to sample a negative pair.

With probability $(1 - r_2)$, we sample the context based on the class labels. Positive pairs have the same labels and negative pairs have different labels. Only labeled instances $1 : L$ are sampled.

Our random walk based sampling method is built upon Deepwalk [108]. In contrast to their method, our method handles real-valued $A$, incorporates negative sampling, and explicitly samples from labels with probability $(1 - r_2)$ to inject supervised information.

An example of sampling when $\gamma = 1$ is shown in Figure 3.9.
Transductive Formulation

In this section, we present a method that infers the labels of unlabeled instances $y_{L+1:L+U}$ without generalizing to unobserved instances. Transductive learning usually performs better than inductive learning because transductive learning can leverage the unlabeled test data when training the model [62].

We apply $k$ layers on the input feature vector $x$ to obtain $h^k(x)$, and $l$ layers on the embedding $e$ to obtain $h^l(e)$, as illustrated in Figure 3.10a. The two hidden layers are concatenated, and fed to a softmax layer to predict the class label of the instance. More specifically, the probability of predicting the label $y$ is written as:

$$p(y|x, e) = \frac{\exp[h^k(x)^T, h^l(e)^T]w^T_y}{\sum_{y'} \exp[h^k(x)^T, h^l(e)^T]w^T_{y'}}$$

where $[\cdot, \cdot]$ denotes concatenation of two row vectors, the super script $h^T$ denotes the transpose of vector $h$, and $w$ represents the model parameter.

Combined with Eq. (3.10), the loss function of transductive learning is defined as:

$$-\frac{1}{L} \sum_{i=1}^{L} \log p(y_i|x_i, e_i) - \lambda E(i, c, \gamma) \log \sigma(\gamma w^T_c e_i),$$

where the first term is defined by Eq. (3.11), and $\lambda$ is a constant weighting factor. The first term is the loss function of class label prediction and the second term is the loss function of context prediction. This formulation is transductive because the prediction of label $y$ depends on the embedding $e$, which can only be learned for instances observed in the graph $A$ during training time.

Inductive Formulation

While we consider transductive learning in the above formulation, in many cases, it is desirable to learn a classifier that can generalize to unobserved instances, especially for large-scale tasks.
For example, machine reading systems [16] very frequently encounter novel entities on the Web and it is not practical to train a semi-supervised learning system on the entire Web. However, since learning graph embeddings is transductive in nature, it is not straightforward to do it in an inductive setting. Perozzi et al. [108] addressed this issue by retraining the embeddings incrementally, which is time consuming and does not scale (and not inductive essentially).

To make the method inductive, the prediction of label \( y \) should only depend on the input feature vector \( x \). Therefore, we define the embedding \( e \) as a parameterized function of feature \( x \), as shown in Figure 3.11d. Similar to the transductive formulation, we apply \( k \) layers on the input feature vector \( x \) to obtain \( h^k(x) \). However, rather than using a “free” embedding, we apply \( l_1 \) layers on the input feature vector \( x \) and define it as the embedding \( e = h^{l_1}(x) \). Then another \( l_2 \) layers are applied on the embedding \( h^{l_2}(e) = h^{l_2}(h^{l_1}(x)) \), denoted as \( h^l(x) \) where \( l = l_1 + l_2 \). The embedding \( e \) in this formulation can be viewed as a hidden layer that is a parameterized function of the feature \( x \).

With the above formulation, the label \( y \) only depends on the feature \( x \). More specifically, \[
p(y|x) = \frac{\exp[h^k(x)^T, h^l(x)^T]w_y}{\sum_{y'}\exp[h^k(x)^T, h^l(x)^T]w_{y'}}
\]

Replacing \( e \) in Eq. (3.10) with \( h^{l_1}(x_i) \), the loss function of inductive learning is

\[
-\frac{1}{L} \sum_{i=1}^{L} \log p(y_i|x_i) - \lambda E_{(i,c,\gamma)} \log \sigma(\gamma w^T_c h^{l_1}(x_i))
\]

where the first term is defined by Eq. (3.12).

Training

We adopt stochastic gradient descent (SGD) [14] to train our model in the mini-batch mode. We first sample a batch of labeled instances and take a gradient step to optimize the loss function of class label prediction. We then sample a batch of context \((i,c,\gamma)\) and take another gradient step to optimize the loss function of context prediction. We repeat the above procedures for \( T_1 \) and \( T_2 \) iterations respectively to approximate the weighting factor \( \lambda \). Algorithm 3 illustrates the SGD-based training algorithm for the transductive formulation. Similarly, we can replace \( p(y_i|x_i,e_i) \) with \( p(y_i|x_i) \) in \( L_s \) to obtain the training algorithm for the inductive formulation. Let \( \theta \) denote all model parameters. We update both embeddings \( e \) and parameters \( \theta \) in transductive learning, and update only parameters \( \theta \) in inductive learning. Before the joint training procedure, we apply a number of training iterations that optimize the unsupervised loss \( L_u \) alone and use the learned embeddings \( e \) as initialization for joint training.

3.3.4 Experiments

In our experiments, Planetoid-T and Planetoid-I denote the transductive and inductive formulation of our approach. We compare our approach with label propagation (LP) [179], semi-supervised
Algorithm 3 Model Training (Transductive)

Input: $A$, $x_{1:L+U}$, $y_{1:L}$, $\lambda$, batch iterations $T_1, T_2$ and sizes $N_1, N_2$

repeat
  for $t \leftarrow 1$ to $T_1$ do
    Sample a batch of labeled instances $i$ of size $N_1$
    $L_s = -\frac{1}{N_1} \sum_i p(y_i | x_i, e_i)$
    Take a gradient step for $L_s$
  end for
  for $t \leftarrow 1$ to $T_2$ do
    Sample a batch of context from $p(i, c, \gamma)$ of size $N_2$
    $L_u = -\frac{1}{N_2} \sum_{(i,c,\gamma)} \log \sigma(\gamma^T w_c e_i)$
    Take a gradient step for $L_u$
  end for
until stopping

Table 3.5: Dataset statistics.

<table>
<thead>
<tr>
<th>DATASET</th>
<th>#CLASSES</th>
<th>#NODES</th>
<th>#EDGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITESEER</td>
<td>6</td>
<td>3,327</td>
<td>4,732</td>
</tr>
<tr>
<td>CORA</td>
<td>7</td>
<td>2,708</td>
<td>5,429</td>
</tr>
<tr>
<td>PUBMED</td>
<td>3</td>
<td>19,717</td>
<td>44,338</td>
</tr>
<tr>
<td>DIEI</td>
<td>4</td>
<td>4,373,008</td>
<td>4,464,261</td>
</tr>
<tr>
<td>NELL</td>
<td>210</td>
<td>65,755</td>
<td>266,144</td>
</tr>
</tbody>
</table>

embedding (SemiEmb) [155], manifold regularization (ManiReg) [8], TSVM [62], and graph embeddings (GraphEmb) [108]. Another baseline method, denoted as Feat, is a linear softmax model that takes only the feature vectors $x$ as input. We also derive a variant Planetoid-G that learns embeddings to jointly predict class labels and graph context without use of feature vectors. The architecture of Planetoid-G is similar to Figure 3.10a except that the input feature and the corresponding hidden layers are removed. Among the above methods, LP, GraphEmb and Planetoid-G do not use the features $x$, while TSVM and Feat do not use the graph $A$. We include these methods into our experimental settings to better evaluate our approach. Our preliminary experiments on the text classification datasets show that the performance of our model is not very sensitive to specific choices of the network architecture. We adapt the implementation of GraphEmb to our Skipgram implementation. We use the Junto library [133] for label propagation, and SVMLight [3] for TSVM. We also use our own implementation of ManiReg and SemiEmb by modifying the symbolic objective function in Planetoid. In all of our experiments, we set the model hyper-parameters to $r_1 = 5/6$, $q = 10$, $d = 3$, $N_1 = 200$ and $N_2 = 200$ for Planetoid. We use the same $r_1$, $q$ and $d$ for GraphEmb, and the same $N_1$ and $N_2$ for ManiReg and SemiEmb.

1We note that it is possible to develop other architectures for different applications, such as using a shared hidden layer for feature vectors and embeddings.

https://github.com/phanein/deepwalk
http://svmlight.joachims.org/
Table 3.6: Accuracy on text classification. Upper rows are inductive methods and lower rows are transductive methods.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>CITESEER</th>
<th>CORA</th>
<th>PUBMED</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEAT</td>
<td>0.572</td>
<td>0.574</td>
<td>0.698</td>
</tr>
<tr>
<td>MANIREG</td>
<td>0.601</td>
<td>0.595</td>
<td>0.707</td>
</tr>
<tr>
<td>SEMIEMB</td>
<td>0.596</td>
<td>0.590</td>
<td>0.711</td>
</tr>
<tr>
<td>PLANETOID-I</td>
<td>0.647</td>
<td>0.612</td>
<td>0.772</td>
</tr>
<tr>
<td>TSVM</td>
<td>0.640</td>
<td>0.575</td>
<td>0.622</td>
</tr>
<tr>
<td>LP</td>
<td>0.453</td>
<td>0.680</td>
<td>0.630</td>
</tr>
<tr>
<td>GRAPHEMB</td>
<td>0.432</td>
<td>0.672</td>
<td>0.653</td>
</tr>
<tr>
<td>PLANETOID-G</td>
<td>0.493</td>
<td>0.691</td>
<td>0.664</td>
</tr>
<tr>
<td>PLANETOID-T</td>
<td>0.629</td>
<td>0.757</td>
<td>0.757</td>
</tr>
</tbody>
</table>

tune $r_2$, $T_1$, $T_2$, the learning rate and hyper-parameters in other models based on an additional data split with a different random seed.

The statistics for five of our benchmark datasets are reported in Table 3.5. For each dataset, we split all instances into three parts, labeled data, unlabeled data, and test data. Inductive methods are trained on the labeled and unlabeled data, and tested on the test data. Transductive methods, on the other hand, are trained on the labeled, unlabeled data, and test data without labels.

Text Classification

We first considered three text classification datasets\[4\] Citeseer, Cora and Pubmed \[122\]. Each dataset contains bag-of-words representation of documents and citation links between the documents. We treat the bag-of-words as feature vectors $x$. We construct the graph $A$ based on the citation links; if document $i$ cites $j$, then we set $a_{ij} = a_{ji} = 1$. The goal is to classify each document into one class. We randomly sample 20 instances for each class as labeled data, 1,000 instances as test data, and the rest are used as unlabeled data. The same data splits are used for different methods, and we compute the average accuracy for comparison.

The experimental results are reported in Table 3.6. Among the inductive methods, Planetoid-I achieves the best performance on all the three datasets with the improvement of up to 6.1% on Pubmed, which indicates that our embedding techniques are more effective than graph Laplacian regularization. Among the transductive methods, Planetoid-T achieves the best performance on Cora and Pubmed, while TSVM performs the best on Citeseer. However, TSVM does not perform well on Cora and Pubmed. Planetoid-I slightly outperforms Planetoid-T on Citeseer and Pubmed, while Planetoid-T gets up to 14.5% improvement over Planetoid-I on Cora. We conjecture that in Planetoid-I, the feature vectors impose constraints on the learned embeddings, since they are represented by a parameterized function of the input feature vectors. If such constraints are appropriate, as is the case on Citeseer and Pubmed, it improves the non-convex optimization of

\[http://lings.umiacs.umd.edu/projects//projects/lbc/\]
Table 3.7: Recall@$k$ on DIEL distantly-supervised entity extraction. Upper rows are inductive methods and lower rows are transductive methods. Results marked with * are taken from the original DIEL paper [12] with the same data splits.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>RECALL@$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>*FEAT</td>
<td>0.349</td>
</tr>
<tr>
<td>MANIREG</td>
<td>0.477</td>
</tr>
<tr>
<td>SEMIEmb</td>
<td>0.486</td>
</tr>
<tr>
<td>PLANETOID-I</td>
<td>0.501</td>
</tr>
<tr>
<td>*DIEL</td>
<td>0.405</td>
</tr>
<tr>
<td>*LP</td>
<td>0.162</td>
</tr>
<tr>
<td>GRAPHEmb</td>
<td>0.258</td>
</tr>
<tr>
<td>PLANETOID-G</td>
<td>0.394</td>
</tr>
<tr>
<td>PLANETOID-T</td>
<td>0.500</td>
</tr>
<tr>
<td>*UPPER BOUND</td>
<td>0.617</td>
</tr>
</tbody>
</table>

Table 3.8: Accuracy on NELL entity classification with labeling rates of 0.1, 0.01, and 0.001. Upper rows are inductive methods and lower rows are transductive methods.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>0.1</th>
<th>0.01</th>
<th>0.001</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEAT</td>
<td>0.621</td>
<td>0.404</td>
<td>0.217</td>
</tr>
<tr>
<td>MANIREG</td>
<td>0.634</td>
<td>0.413</td>
<td>0.218</td>
</tr>
<tr>
<td>SEMIEmb</td>
<td>0.654</td>
<td>0.438</td>
<td>0.267</td>
</tr>
<tr>
<td>PLANETOID-I</td>
<td>0.702</td>
<td>0.598</td>
<td>0.454</td>
</tr>
<tr>
<td>LP</td>
<td>0.714</td>
<td>0.448</td>
<td>0.265</td>
</tr>
<tr>
<td>GRAPHEmb</td>
<td>0.795</td>
<td>0.725</td>
<td>0.581</td>
</tr>
<tr>
<td>PLANETOID-G/T</td>
<td>0.845</td>
<td>0.757</td>
<td>0.619</td>
</tr>
</tbody>
</table>

embedding learning and leads to better performance. However, if such constraints rule out the optimal embeddings, the inductive model will suffer.

Planetoid-G consistently outperforms GraphEmb on all three datasets, which indicates that joint training with label information can improve the performance over training the supervised and unsupervised objectives separately. Figure 3.11 displays the 2-D embedding spaces on the Cora dataset using t-SNE [144]. Note that different classes are better separated in the embedding space of Planetoid-T than that of GraphEmb and SemiEmb, which is consistent with our empirical findings. We also observe similar results for the other two datasets.

**Distantly-Supervised Entity Extraction**

We next considered the DIEL (Distant Information Extraction using coordinate-term Lists) dataset [12]. The DIEL dataset contains pre-extracted features for each entity mention in text, and a graph
Figure 3.11: t-SNE Visualization of embedding spaces on the Cora dataset. Each color denotes a class.

that connects entity mentions to coordinate lists. The goal is to extract medical entities from text given feature vectors and the graph.

We follow the exact experimental setup as in the original DIEL paper [12], including data splits of different runs, preprocessing of entity mentions and coordinate lists, and evaluation. We treat the top-$k$ entities given by a model as positive instances, and compute recall@$k$ for evaluation ($k$ is set to 240,000 following the DIEL paper). We report the average result of 10 runs in Table 3.7, where $\text{Feat}$ refers to a result obtained by SVM (referred to as DS-Baseline in the DIEL paper). The result of LP was also taken from [12]. DIEL in Table 3.7 refers to the method proposed by the original paper, which is an improved version of label propagation that trains classifiers on feature vectors based on the output of label propagation. We did not include TSVM into the comparison since it does not scale. Since we use Freebase as ground truth and some entities are not present in text, the upper bound of recall as shown in Table 3.7 is 0.617.

Both Planetoid-I and Planetoid-T significantly outperform all other methods. Each of Planetoid-I and Planetoid-T achieves the best performance in 5 out of 10 runs, and they give a similar recall on average, which indicates that there is no significant difference between these two methods on this dataset. Planetoid-G clearly outperforms GraphEmb, which again shows the benefit of joint training.
**Entity Classification**

We sorted out an entity classification dataset from the knowledge base of Never Ending Language Learning (NELL) [16] and a hierarchical entity classification dataset [30] that links NELL entities to text in ClueWeb09. We extracted the entities and the relations between entities from the NELL knowledge base, and then obtained text description by linking the entities to ClueWeb09. We use text bag-of-words representation as feature vectors of the entities.

We next describe how to construct the graph based on the knowledge base. We first remove relations that are not populated in NELL, including “generalizations”, “haswikipediaurl”, and “atdate”. In the knowledge base, each relation is denoted as a triplet \((e_1, r, e_2)\), where \(e_1, r, e_2\) denote head entity, relation, and tail entity respectively. We treat each entity \(e\) as a node in the graph, and each relation \(r\) is split as two nodes \(r_1\) and \(r_2\) in the graph. For each \((e_1, r, e_2)\), we add two edges in the graph, \((e_1, r_1)\) and \((e_2, r_2)\).

We removed all classes with less than 10 entities. The goal is to classify the entities in the knowledge base into one of the 210 classes given the feature vectors and the graph. Let \(\beta\) be the labeling rate. We set \(\beta\) to 0.1, 0.01, and 0.001. \(\max(\beta N, 1)\) instances are labeled for a class with \(N\) entities, so each class has at least one entity in the labeled data.

We report the results in Table 3.8. We did not include TSVM since it does not scale to such a large number of classes with the one-vs-rest scheme. Adding feature vectors does not improve the performance of Planetoid-T, so we set the feature vectors for Planetoid-T to be all empty, and therefore Planetoid-T is equivalent to Planetoid-G in this case.

Planetoid-I significantly outperforms the best of the other compared inductive methods—i.e., SemiEmb—by 4.8%, 16.0%, and 18.7% respectively with three labeling rates. As the labeling rate decreases, the improvement of Planetoid-I over SemiEmb becomes more significant.

Graph structure is more informative than features in this dataset, so inductive methods perform worse than transductive methods. Planetoid-G outperforms GraphEmb by 5.0%, 3.2% and 3.8%.
Chapter 4

Conclusions

4.1 Summary and Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th>Model G</th>
<th>Loss $l_g$</th>
<th>Loss $l_t$</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-XL</td>
<td>AR</td>
<td>Likelihood</td>
<td>-</td>
<td>One-Phase</td>
</tr>
<tr>
<td>XLNet</td>
<td>AR</td>
<td>Likelihood</td>
<td>Target task loss</td>
<td>Pretraining &amp; Finetuning</td>
</tr>
<tr>
<td>Comp GAN</td>
<td>GAN</td>
<td>Feature matching</td>
<td>$(K + 1)$-class loss</td>
<td>Joint training</td>
</tr>
<tr>
<td>Semi QA</td>
<td>AR</td>
<td>Adversarial</td>
<td>QA loss</td>
<td>Joint training</td>
</tr>
<tr>
<td>Planetoid</td>
<td>MLP</td>
<td>Likelihood</td>
<td>Classification loss</td>
<td>Joint training</td>
</tr>
</tbody>
</table>

Table 4.1: Summary and comparison of how different methods proposed in this work instantiate generative feature learning. AR means autoregressive modeling. “Comp GAN” refers to Complement GAN, “Semi QA” means semi-supervised QA, and “Planetoid” is our graph-based semi-supervised learning method.

We have presented the following generative feature learning methods:

1. **Transformer-XL.** Transformer-XL consists of a segment-level recurrence mechanism and a new relative positional encoding scheme. As a result, the architecture allows capturing longer-range dependency and resolves the context fragmentation problem. Transformer-XL achieves better language modeling performance on both long and short sequences, learns dependency that is 80% longer than RNNs and 450% longer than vanilla Transformers, and is up to 1,800 times faster than vanilla Transformers during language modeling evaluation.

2. **XLNet.** XLNet is a pretraining method based on the novel permutation language modeling objective. By using this objective, XLNet enables modeling bidirectional contexts using autoregressive models, avoiding making the independence assumption. XLNet also consists of a series of techniques to properly optimize the permutation language modeling objective, including a new way of parameterizing the conditional distribution and the two-stream attention mechanism. XLNet bridges the gap between language modeling and unsupervised pretraining and shows that progress in language modeling can be used to improve pretraining and thus downstream tasks.
3. **Complement GAN.** Theoretically we show that given the \((K + 1)\)-class classifier objective, good semi-supervised learning requires a complement generator that generates off-manifold samples. Our theoretical analysis answers the questions of how the classifier benefits from joint training with a generator and why good semi-supervised classification performance and a good generator cannot be obtained at the same time. Driven by theoretical understanding, we propose the definition of a preferred generator and use a new training objective to realize the goal.

4. **Generative Domain-Adaptive Nets.** Our semi-supervised question answering method trains a generative model to generate questions based on the unlabeled text, and combines model-generated questions with human-generated questions for training question answering models. We develop novel domain adaptation algorithms, based on reinforcement learning, to alleviate the discrepancy between the model-generated data distribution and the human-generated data distribution.

5. **Planetoid.** Given a graph between instances, Planetoid trains an embedding for each instance to jointly predict the class label and the neighborhood context in the graph. Planetoid has both transductive and inductive variants. In the transductive variant of our method, the class labels are determined by both the learned embeddings and input feature vectors, while in the inductive variant, the embeddings are defined as a parametric function of the feature vectors, so predictions can be made in instances not seed during training. Our method extends the concept of generative modeling to semi-supervised learning on graphs, which is essentially generating random walk paths on the graphs.

The above methods are summarized and compared in Table 4.1. We follow the notations in the definition of generative feature learning and describe how different methods instantiate the generative feature learning framework with custom loss functions, model formulation, and training paradigm.

Overall, our approaches obtained state-of-the-art results on over 30 benchmarks at the time of their publication, including natural language inference, question answering, text classification, language modeling, semi-supervised learning, etc., demonstrating the practical effectiveness.

### 4.2 Future Work

Although the generative feature learning methods proposed in this thesis have achieved remarkable success, there are still many questions remaining open.

- **Combining unsupervised pretraining and semi-supervised learning.** In the literature, unsupervised pretraining and semi-supervised learning have been developed to tackle different challenges. Unsupervised learning focuses on leveraging all-domain data, while semi-supervised learning is more about learning from unlabeled data with the same distribution. Clearly, there is a quality-quantity tradeoff. While unsupervised pretraining benefits from quantity, semi-supervised learning has the advantages of data quality (i.e., closer to the i.i.d assumption). Recently [161] attempted to combine the advantages of both by performing semi-supervised learning on top of pretrained models. However, the main focus
was classification and it will be interesting to study unifying or combining semi-supervised learning and unsupervised learning in a more effective way with applications to more tasks.

- **Efficient unsupervised pretraining.** Unsupervised pretraining methods heavily rely on large-scale computation and data, which prohibits most of the research community from working on meaningful related research and might eventually slow down the development of the area. There are two possible, interesting future directions to alleviate this problem. The first is to develop more efficient architectures that are more data efficient. The second is to find a medium-scale surrogate setting where progress could be translated to benefits in pretraining. For example, since XLNet bridges the gap between language modeling and unsupervised pretraining, progress in language modeling might improve pretraining. At the same time, medium-scale language modeling benchmarks are prevalent, which forms a meaningful surrogate.

- **Pretraining for few-shot learning.** Unsupervised pretraining still does not solve all the NLP tasks, especially on tasks with little labeled data. For one thing, as mentioned above, combination with semi-supervised learning approaches might lead to improvements. For another, it is critical to develop unsupervised pretraining methods that are able to perform few-shot learning.

- **Effects of data scale/quality in unsupervised pretraining.** The effects of adding more data to unsupervised pretraining are not yet clear. Since curated corpora like Wikipedia and BooksCorpus are relatively scarce, additional data often come from the Web, which results in decrease of quality. It becomes a critical question of how to learn from relatively low quality but more abundant data. It will also be interesting to study how to jointly learn from curated text and Web text without being dominated by Web text.

- **XLNet for computer vision.** It will be intriguing to extend XLNet to computer vision, because Transformer-XL might have the potential of modeling longer-range dependency in images, which is very hard for CNNs.

- **GAN-based semi-supervised learning for text.** Applying our complement GANs approach to text requires generating discrete data, which results in the problem of non-differentiability. It will be interesting to explore ideas to tackle this challenge, such as methods that do not use explicit generators [97] or generation in the feature space.

- **Improving generator quality in semi-supervised QA.** The generator in semi-supervised QA was trained on limited data and thus was not able to generate realistic data. We conjecture that this is one of the reasons why semi-supervised QA was not able to improve performance when all training data was used. In the future, it will be interesting to also pretrain the generator using unlabeled data to address this problem.
Bibliography


[60] Ming Ji, Yizhou Sun, Marina Danilevsky, Jiawei Han, and Jing Gao. Graph regularized


[76] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent sub-


[139] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip


