Improving Deep Generative Modeling with Practical Applications

Zihang Dai

July 15, 2020

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Yiming Yang (Chair), Carnegie Mellon University
Ruslan Salakhutdinov, Carnegie Mellon University
Yonatan Bisk, Carnegie Mellon University
Quoc V. Le, Google Brain

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

Abstract

At the core of unsupervised learning, probabilistic generative models provide a systematic framework to understanding real world data from various domains in a probabilistic manner. Among many possible desiderata of generative models, density estimation, data generation, and representation learning are widely regarded as the three most wanted properties, whose advancement not only bears important theoretical values but can also lead to a breakthrough for practical applications. In recent years, with the rapid development of deep neural networks and computational hardware, the field of deep generative models has witnessed dramatic advancements in all three aspects, significantly outperforming traditional generative models.

Despite the success, existing neural architectures and training objectives are still subject to certain fundamental drawbacks. With these challenges in mind, this thesis focuses on developing novel neural architectures and training objectives that are highly expressive, allow for efficient optimization, and can scale to a large amount of data for generative modeling.

Notably, to better exploit the optimization advantage of Transformer to capture long-term dependency, we propose Transformer-XL, which integrates segment-level recurrence into self-attention without disrupting the temporal coherence. Further, to combine the benefits of autoregressive and denoising auto-encoding based language pretraining, we propose XLNet, which relies on a permutation language modeling objective to maximize the expected log-likelihood of a sequence w.r.t. all possible permutations of the factorization order and hence capture bidirectional context. By further integrating ideas from Transformer-XL, XLNet consistently outperforms previous best language pretraining method under the same training condition, and achieves the state-of-the-art performance when scaled up. In addition, to further exploit the effectiveness of language pretraining, we propose a more efficient self-attention architecture Funnel-Transformer, which compresses the hidden state sequence to a shorter length and hence reduces the computation cost. With sequence compression, Funnel-Transformer allows one to trade the sequential resolution of the hidden state sequence for a deeper or wider model, leading to substantial gains under the same amount of computation as measured by the FLOPs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivations

In the last decade, with the fast development of Deep Learning and the creation of large-scale labeled datasets, the field of supervised learning has experienced tremendous success, matching or even surpassing human performances on various important real-world problems. Despite the success, supervised learning usually requires a large amount of labeled data, which could be highly expensive, if possible, to obtain. More importantly, models supervised learned for one task may not be able to transfer any knowledge to another problem. As a result, it often requires repetitive data annotation for each newly emerged problem, restricting the fast adaption of supervised learning to a new domain.

In comparison, unlabeled data is usually abundant and easily accessible. This motivates a large body of research to consider utilizing unlabeled data to perform unsupervised or semi-supervised learning. Among many approaches, by combining expressive deep neural architectures and "gradient friendly" training objectives, deep generative models have become one of the most important driving forces behind unsupervised and semi-supervised learning.

Generally speaking, given unlabeled data samples, deep generative modeling mainly targets three fundamental problems with wide applications:

1. **Density estimation**: Providing an accurate estimation of the density (contiguous) or probability (discrete) of a given sample.
2. **Data generation** or **sampling**: Offering a generation or sampling mechanism that can draw natural samples from the underlying distribution.
3. **Representation learning**: Discovering and inferring meaningful latent representations, stochastic or deterministic, of a given sample.

In the last few years, the field of deep generative models has witnessed significant progress in all three problems mentioned above, renewing the state-of-the-art (SOTA) results every a few months (see Chapter 2 for a review). Among the vast amount of progress, there are three recurring trends that largely inspire this thesis.

- **The ubiquitous effectiveness of deep autoregressive models**. Different from latent-variable models that utilize latent factors to capture the correlation in data, autoregressive models simply apply the product rule and factorize the data likelihood in an autoregressive

manner, i.e., $p(\mathbf{x}) = \prod_i^{|\mathbf{x}|} p(x_i \mid \mathbf{x}_{<i})$. With this formulation, as long as each conditional factor is tractable, the data likelihood can be evaluated exactly, making autoregressive models natural density estimators. On the other hand, due to the lack of latent variables, it was conventionally believed that autoregressive models won't be able to capture more complex real world data that exhibits more variability.

However, in the context of deep autoregressive models, each conditional $p(x_i \mid \mathbf{x}_{<i})$ factor is usually modeled by a highly expressive deep neural network, which in theory, does not have a capacity limitation. In addition, modern deep autoregressive models are usually designed to fully differentiable and hence can be efficiently trained with standard maximum likelihood estimation (MLE) via back-propagation. Surprisingly, this simple formula has led to various SOTA autoregressive density estimators in a wide spectrum of fields including natural language, high-fidelity images or human speech.

More interestingly, though not explicitly trained to perform generation, it turns out that deep autoregressive models trained by MLE are also SOTA samplers in many domains like text generation and speech synthesis. What's more, recent research has shown that the hidden activations of deep autoregressive models trained by MLE are superior representations in downstream task, leading to significant performance gain [36, 116, 122]. In other words, deep autoregressive modeling also offers an effective unsupervised representation learning mechanism.

- **Model architecture plays an increasingly important role**. In the last few years, the field of deep generative modeling has seen many key innovations in training algorithm and objective such as variational auto-encoder (VAE) [70, 132], generative adversarial networks (GAN) [46], and normalizing flows [131]. While these breakthroughs are certainly crucial, the underlying model architecture often plays an equally important role, especially when performance is concerned.

  For instance, in computer vision (CV), when standard convolutional neural networks (CNN) are replaced by residual networks (ResNet) [55], the performances of image density estimation, generation and self-supervised feature learning have all been improved to a new level, despite exactly the same training algorithm and objective. Similarly, for sequence modeling, the shift from HMM/CRF to recurrent neural networks (RNN), in particular the Long short-term memory (LSTM) [58], leads to huge performance gains and dramatically changes the natural language processing (NLP) pipelines. More recently, with little change to the training algorithm and objective, the newly invented Transformer [156] not only brings significant performance improvement but also leads to a new era of representation learning for natural language processing.

  Meanwhile, how to find better neural models remains to be a highly challenging topic as it involves many complicated considerations. Firstly, when developing new models, capacity is often not the central concern anymore. Instead, it is often more beneficial to balance the flexibility, capacity, trainability (optimization difficult), and generalization ability properly. Secondly, while neural architecture search provides certain help, it does not create novel search space which is often the most fundamental piece of architecture design. What's more, in the context of deep generative modeling, in order to satisfy the constraints posed by

different training objectives and algorithms, one may need to modify the model architecture accordingly.

- **Scalability is key to the SOTA performance.** In parallel to the fast development of deep generative models, the computational power has also been rapidly improving recently, allowing researchers to consider the abundant unlabeled data at scale as well as a larger model architecture. It turns out increasing the data scale and model size is extremely effective and able to produce dramatic gains that leads to a brand new level of performance of generative modeling [15, 39, 123]. More importantly, most of the recent successes from scaling up are based on relatively simple and standard approaches that do not involve complicated techniques with non-trivial computational burden and hence can consume more unlabeled data within the same amount of training time. Given this trend, it can be foreseen that the scalability of models and algorithms will become increasingly important in achieving the SOTA performance in generative modeling and downstream applications.

In summary, as universal function approximators, deep neural networks have reshaped the field of generative modeling towards holistic approaches that (i) utilize a powerful and flexible model architecture, (ii) allow for efficient optimization, and (iii) can scale to large amount of data. Motivated by these observations, in this thesis, we will focus on developing deep generative models that satisfy the three properties. Specifically, with a focus on the language domain, we will consider the following three concrete directions:

(1) Design novel and generic neural model architectures that have improved expressiveness in important aspects and can be applied to different domains and problems

(2) Develop training objectives and algorithm that can lead to significant performance gains in any of the fundamental problems of generative modeling (i.e., density estimation, data sampling and representation learning)

(3) Systematically scale up (1) and (2) to advance the state-of-the-art results in both generative modeling itself and downstream problems

1.2 Challenges and Contributions

As a long-standing area of research, language modeling is the central problem connecting natural language processing and generative modeling. Despite the rapid development, there are still fundamental challenges awaiting for solutions. Among them, we are mostly interested in the following two due to their particular significance.

- Firstly, existing models architectures all suffer from certain fundamental weaknesses.

  In the last decade, the task of language modeling has been largely driven by various improved variants of RNNs. Despite the wide adaption, RNNs are difficult to optimize due to gradient vanishing and explosion [59], and the introduction of gating in LSTMs and the gradient clipping technique [50] are sufficient to fully address this problem.

  On the other hand, with the direct connections between long-distance word pairs, the recently proposed Transformer has a potential to learn of long-term dependency [156]. However, standard Transformer are built to processed sequences of fixed length. Hence,

when directly adapted to language modeling [2, 122], vanilla Transformer LM will not be able to capture any longer-term dependency beyond the predefined context length. In addition, the fixed-length segments used to train vanilla Transformer LMs are usually 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.

Moreover, the self-attention mechanism in Transformer has quadratic space and time complexity w.r.t. the sequence length. Hence, the expressiveness and optimization advantage of Transformer also come with a non-trivial computational burden.

- Secondly, existing language modeling based unsupervised representation approaches are all subject to certain flaws.

  As demonstrated by ELMo [116] and GPT-1 [122], by scaling up language models, one can obtain good representations or a model initialization that can be transferred to downstream tasks with large gains. However, standard language models depend on a fixed left-to-right (or right-to-left) autoregressive factorization to perform MLE training. As a result, the model only learns uni-directional dependency and simply stacking two networks separately optimized with reverse factorization orders is not able to capture features that require interactions between bi-directional context.

  Faced with this problem, BERT [36] proposes a denoising auto-encoding objective that is able to capture bidirectional context. However, the artificial symbols like `[MASK]` introduced by the corruption process in 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 autoregressive language modeling. Effectively, 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.

Faced with the two challenges, this thesis starts with proposing a novel neural architecture Transformer-XL that enables learning dependency beyond a fixed length without disrupting temporal coherence. In a nutshell, Transformer-XL consists of a segment-level recurrence mechanism and employs a novel positional encoding scheme to ensure the temporal coherence is maintained. With the recurrent mechanism, Transformer-XL not only enables capturing longer-term dependency, but also resolves the optimization inefficiency due to the non-semantic consecutive chunking. As a result, TransformerXL learns dependency that is 80% longer than RNNs and 450% longer than vanilla Transformers, achieves better performance on both short and long sequences, and improves the SOTA on various standard benchmark datasets. Moreover, after trained on only 100 million Wikipedia tokens, Transformer-XL manages to generate reasonably coherent, novel text articles with thousands of tokens. The details will be covered in Chapter 3.

Next, we turn to the question whether there exists a pretraining method that is able to leverage the best of both autoregressive language modeling and denouncing auto-encoding while avoiding their limitations. In this end, we propose XLNet, a generalized autoregressive pretraining method that (1) enables learning bidirectional contexts by maximizing the expected likelihood over all permutations of the factorization order and (2) overcomes the limitations of BERT thanks to its

autoregressive formulation. Also, the permutation language modeling objective is able to provide more effective training signals than denoising auto-encoding. Furthermore, as an autoregressive model, XLNet is able to integrate the improvements of Transformer-XL into pretraining. Empirically, XLNet consistently outperforms BERT under the same training conditions. When further scaled up to more training data, XLNet achieves state-of-the-art results on a wide spectrum of tasks including question answering, natural language inference, sentiment analysis, and document ranking. The technical details and more empirical results are presented in Chapter 4.

Finally, with the success of language pretraining, it is highly desirable to develop more efficient architectures of good scalability that can exploit the abundant unlabeled data at a lower cost. To improve the efficiency, we examine the much-overlooked redundancy in maintaining a full-length token-level presentation, especially for tasks that only require a single-vector presentation of the sequence. With this intuition, we propose Funnel-Transformer which gradually compresses the sequence of hidden states to a shorter one and hence reduces the computation cost. More importantly, by re-investing the saved FLOPs from length reduction in constructing a deeper or wider model, we further improve the model capacity. In addition, to perform token-level predictions as required by common pretraining objectives, Funnel-Transformer is able to recover a deep representation for each token from the reduced hidden sequence via a decoder. Empirically, with comparable or fewer FLOPs, Funnel-Transformer outperforms the standard Transformer on a wide variety of sequence-level prediction tasks, including text classification, language understanding, and reading comprehension.

1.2.1 Additional Contributions

Apart from these two more notable issues in language domain, there are also many other challenges in deep generative modeling. Here, we list some topics that this thesis contributes.

- While language modeling attracts lots of research interests, little attention has been given to the expressiveness of the Softmax-based output distribution in neural language models. Specifically, although the backbone neural network, RNN or Transformer, is highly expressive, it remains an unclear question that whether the combination of dot product and Softmax is capable of modeling the real conditional probability, which can vary dramatically with the change of the context.

  In Chapter 6.1, we carefully study this problem from a perspective of matrix factorization. We show that, when the rank of ground-truth conditional probability matrix is higher than word embedding dimension, a Softmax-based neural language model with the standard formulation can suffer from a capacity limitation termed the Softmax bottleneck. To break this bottleneck, we propose a simple yet effective solution, which employs a Mixture of Softmaxes (MoS) to parameterize the output distribution. Empirically, MoS significantly improves the SOTA the-art perplexities at the time of publication.

- With a totally different training objective, GANs have become the SOTA image generators. However, as implicit generative models, GANs cannot provide an exact density estimation which not only makes their evaluation extremely difficult but also restricts their potential application areas. Also, how GANs can benefit downstream tasks remain largely unknown.

  Faced with these problems of GANs, in Chapter 6.2, we propose a general framework to

enable GANs to perform energy (unnormalized probability) estimation by taking a dual perspective of the minimax objective. Further, in Chapter 6.3, we provide a theoretical framework to illustrate how a classifier can be benefited from joint training with a "bad" generator in semi-supervised setting, offering more fundamental understanding about why and how GANs work for semi-supervised learning.

Chapter 2

Related Work

2.1 Deep Generative Models

As a defining feature of deep generative models, multi-layer non-linear neural networks are employed to model the interdependence of data. As a benefit, deep generative models do not suffer from the capacity limitation in theory and can learn to construct high-level representations purely from data. More importantly, when back-propagation is allowed (by the objective and model architecture), the training of deep generative models turns out to be highly efficient, leading to significantly better performance compared to traditional non-deep models. Meanwhile, the capacity of deep generative models almost always comes with some sort of intractability. Here, we focus on three representative types of deep generative models.

- **Deep latent variable models** (Deep LVM) such as restricted Boltzmann machine (RBM) [56] and variational auto-encoder (VAE) [70, 132] are designed to enable efficient (approximate) inference of the latent variables. Specifically, by utilizing a particular parameterization, RBM with binary latent code allows exact posterior inference. In contrast, VAE does not depend on a strong parametric restriction to ease inference. Instead, VAE utilizes an encoder network to perform amortized variational inference.

  However, due to the exponentially many possible values of the latent variables, computing the exact data likelihood (density) $p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})$ becomes intractable. To this end, VAE manages to provide a lower bound of the data likelihood [18] and RBM can only rely on importance sampling to get a rough approximation [17].

  As for data generation, VAE admits easy ancestral sampling thanks to its directed structure. In contrast, the undirected RBM has to resort to a delicate MCMC (Gibbs sampling) procedure to obtain a single sample.

- **Deep autoregressive models** do not possess any latent variable. To model the complicated high dimensional data, they simply factorize the density in an autoregressive manner, e.g., $p(\mathbf{x}) = \prod_i^{\dim(\mathbf{x})} p(x_i \mid \mathbf{x}_{<i})$. Then, each conditional factor is modeled by a potentially parameter shared neural network, making the generative model highly expressive and being able to capture the complex dependency among dimensions despite the lack of latent variables. Due to the autoregressive formulation, density estimation and sampling are both straightforward in this case. However, the sampling can be very slow for high-dimensional

or long-horizon temporal data due to the sequential dependency.

- **Generative adversarial networks** (GANs) [46] are a family of implicit generative models. Instead of prescribing a parametric data distribution, GANs only define a data generation process, which operates by sampling a lower dimensional noise vector ϵ from some simple distribution, say a standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and transforming the noise into a data sample with a generator neural network, i.e., $\mathbf{x} = G(\epsilon)$. To train the generator neural network G, another discriminator is introduced to form a mini-max game, where the discriminator tries to distinguish the true and generated samples, while the generator tries to fool the discriminator by gradually generating better samples. As a natural sampler, GANs are able to generate images with significantly better perceptual fidelity [15]. In addition, by incorporating another encoder, the GAN framework can be easily extended to bidirectional GAN (BiGAN) or equivalently adversarially learned inference (ALI),

  Despite the huge success in generating natural images, GANs also possess various weaknesses. Firstly, as an implicit generative model, GANs do not define an explicit density and thus cannot perform density estimation. Moreover, even for data generation, recent evidence shows that GANs haven't brought any clear advantage to generating discrete data such as natural language [19].

2.2 The Wide Success of Deep Autoregressive Models

In recent years, the field of deep generative models has witnessed some significant progress, renewing some conventional wisdom or belief in the filed. Arguably, one of the most shocking progress is the ubiquitous effectiveness of deep autoregressive models when advanced neural architectures developed recently are employed. Specifically, for the three fundamental problems (i.e., density estimation, data generation and representation learning) of generative modeling discussed in chapter 1, deep autoregressive models have shown a generic effectiveness, leading to SOTA performance in various applications. In the sequel, we provide a brief review of these advancements that are most relevant to this thesis.

2.2.1 Autoregressive Models for Density Estimation and Data Generation

Historically, the idea of deep autoregressive models trace back the NADE architecture proposed by Larochelle and Murray [85], which models the density of binary observations with simple feed-forward neural networks. The idea was later extended to continuous values input [151] and more efficient variants were subsequently introduced [44, 152]. In these early works, it has been shown that deep autoregressive models can sometimes yield superior density estimation compared to latent variable models, though the sample quality still lags behind. However, due to the lack of stochastic latent variables, it was widely believed that autoregressive models won't be able to capture more complex real world data such as high-fidelity images or human speech, which exhibits significantly more uncertainty.

This perception started to change when LSTM based autoregressive model achieved a breakthrough performance on the task of language modeling (LM) [67, 172] and machine translation

(MT) [147]. Notably, the problem of MT requires generating samples from the model, which suggests that given powerful enough architecture for modeling each conditional factor, the autoregressive models can also be a good sampler. More recently, a series of work based on very similar convolutional autoregressive models, ranging from PixelCNN to the WaveNet, VideoNet, and ByteNet, has largely renew the perception of autoregressive models. Specifically, based on causal convolution, the PixelCNN [155] reaches a new level of SOTA performance for image density estimation, and soon becomes a standard component in image density estimator. Moreover, in terms of the perceptual quality of generated images, PixelCNN usually performs better than the VAE variants, steadily approaching the perceptual quality of GANs as architectures continue to improve. Soon after that, by utilizing dilated convolution, WaveNet [154] achieved a new SOTA naturalness in text to speech, once again showing that autoregressive models can work as an excellent sampler.

2.2.2 Autoregressive Models for Representation Learning

Meanwhile, it has been shown that deep autoregressive generative models can also be used for unsupervised representation learning. Previously, since autoregressive models do not possess any latent variable, it remains an open question how to utilize autoregressive models to learn and extract useful representations.

Recently, it has been shown that the hidden states of deep autoregressive models can capture different levels of abstraction of the data and thus serve as unsupervised learned representation. For instance, in NLP, by training a deep autoregressive using the standard language modeling objective [31, 60, 116, 122] and/or other variants such as denoising auto-encoding sentences [31] and inpainting masked words [36], the learned hidden states can be used in downstream tasks. Notably, two of the most recent methods, ELMo [116] and BERT [36], have essentially revolutionized unsupervised representation learning in the NLP, achieving SOTA performances on a wide spectrum of tasks. In the field of computer vision, similar approaches have also been used to learn image features by predicting future frames in a video streams with autoregressive models [145, 160].

In shear comparison, latent variable model or (Bi)GAN based representation learning cannot achieve such a substantial improvement so far. One widely held explanation of this phenomenon is that autoregressive models are much easier to train with the optimization techniques we have.

2.3 From RNN and CNN to Self-Attention

As discussed in Chapter 1, the model architecture plays an increasingly important role in generative modeling. Given the focus of this thesis, we review some key developments of the neural model architectures for deep autoregressive models.

As discussed earlier, typical deep autoregressive models factorize the data distribution into the product of *homogeneous* conditional factors, i.e., $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid \mathbf{x}_{<t})$, and employ a *parameter-shared* neural model to parameterize the conditional factor. For a long period of time, the default choice of this conditional model has been RNN or causal (dilated) CNN (or a combination of both) depending on the data type in consideration. But more recently, relying on

the self-attention mechanism, a very successful alternative choice often referred to as Transformer has been introduced [156].

Originally, the attention mechanism was proposed for the encoder-decoder architecture [6] in neural machine translation to mimic the idea of alignment in phrase based machine translation. Specifically, the basic idea of the attention mechanism is to gather information from a sequence of encoder vectors dynamically based on the current decoder state. From a higher level, it defines a dynamic message passing or computation process based on the pairwise relationship between vectors, which essentially represents a type of relational bias [8]. In comparison, the self-attention [156] considers the pairwise relationship within a single set of vectors rather than that across two sets of vectors. In this case, self-attention can be seen as a standard neural network layer that processes a sequential input and outputs a sequence of hidden states.

While the functionality of self-attention is similar to that of an RNN, a critical distinction lies in that RNN processes the input sequence in a strictly sequential order, forming a computational recurrence. In contrast, self-attention processes the input in parallel, where each output location independently gather information from all input locations using attention, with all output locations sharing the same model parameters. As a result, the longest computational path in an RNN layer is $O(T)$, where T is the sequence length. Consequently, RNNs will suffer from the vanishing gradient problem [59] when the sequence length is long. In comparison, every input location is directly linked to each output location in self-attention, leaving the longest computational path only $O(1)$. Hence, self-attention has an optimization advantage over RNN for modeling long sequences.

On the other hand, when compared to CNNs, self-attention is not restricted to a fixed receptive field as in CNNs and can absorb information from the entire sequence for each operation, establishing a direct link between faraway positions In addition, the number of model parameters in self-attention does not change w.r.t. the attention span, while the number of CNN parameters grows linearly w.r.t. the size of its receptive field. As a result, self-attention is significantly more effective and parameter efficient in learning the representation of long sequences.

Empirically, the Transformer architecture achieved the SOTA result on neural machine translation [156]. In addition, when Transformer is used as the base model in unsupervised representation training via LM objectives [36, 122], the learned features are significantly better than those from an LSTM based model [116], leading to super-human performances in some important question answering problems. Moreover, when employed as an autoregressive model for image density estimation, the image Transformer consistently outperforms the PixelCNN on different datasets [115], although image Transformer does not utilize convolution to model local invariance of images while PixelCNN does.

Chapter 3

Transformer-XL: Attentive Language Modeling beyond a Fixed-Length Context

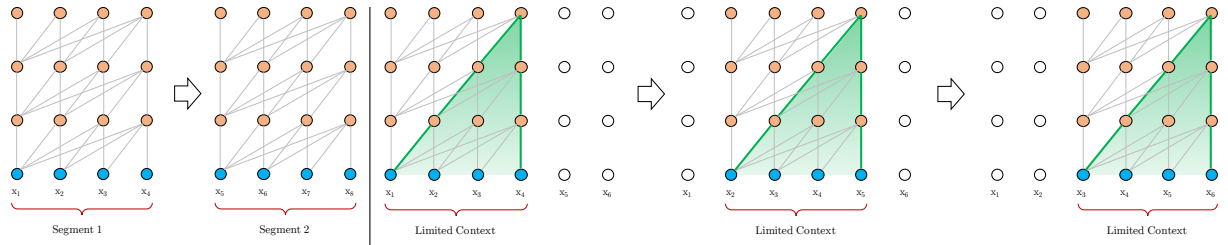
3.1 Background and Motivation

As discussed in section 2.3, the fundamental building block for deep auto-regressive models has gradually shifted from RNN and CNN to Transformer due to its advantage in both capacity and optimization. Following this trend, Al-Rfou et al. [2] 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, during the fixed-length nature of the standard Transformer, the LM training in Al-Rfou et al. [2] is performed on separated fixed-length segments of a few hundred characters, without any information flow across segments. To see why this poses a significant limitation, let's first review how Transformer can be applied to language modeling.

Given a *long* corpus of tokens $\mathbf{x} = (x_1, \ldots, x_T)$, the task of language modeling is to estimate the joint probability $P(\mathbf{x})$. When deep auto-regressive models are employed, we factorize the joint distribution as $P(\mathbf{x}) = \prod_t P(x_t \mid \mathbf{x}_{<t})$ and utilize a neural model to learn the conditional distribution. Ideally, we would like to use Transformer as the neural model to encode the entire context sequence to $\mathbf{x}_{<t}$ into a fixed size hidden state h_t. Then, the hidden state can be multiplied with the word embeddings to obtain the logits and hence the conditional distribution $P(X_t \mid \mathbf{x}_{<t})$ by taking the Softmax. However, in practice, due the limited memory and computation resource, it is infeasible to compress the entire context, which can contain millions of token, and then perform end-to-end training with back-propagation.

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. [2]. We refer to this model as the *vanilla model* and visualize it in Fig. 3.1a. 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 only a few hundred on character-level language modeling [2]. Therefore, although the self-attention mechanism is less affected by the vanishing gradient problem compared to RNNs, the vanilla model is not able to



(a) Train phase. (b) Evaluation phase.

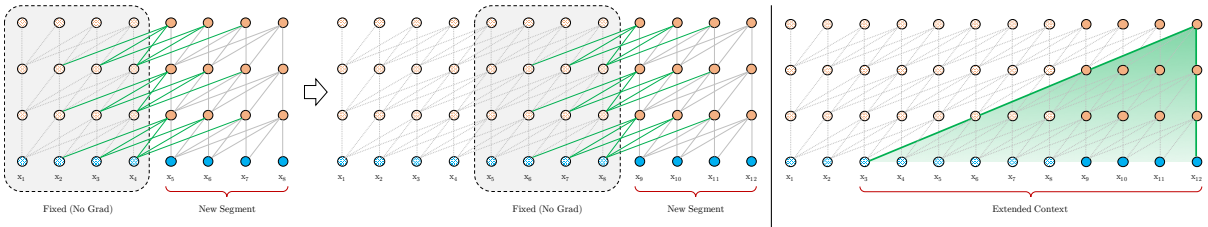
Figure 3.1: Illustration of the vanilla model with a segment length 4.

fully exploit this optimization advantage. Second, in practice, the fixed-length segments are created by selecting a consecutive chunk of symbols without respecting the sentence or any other semantic boundary to improved efficiency [2, 36, 116]. 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*.

In addition, another practical weakness of the vanilla model is that during evaluation (i.e. density estimation), in order for each prediction utilizes the longest possible context exposed during training, the vanilla model has to recompute the entire context. As shown in Fig. 3.1b, at each step of the evaluation, the vanilla model consumes a segment of the same length as in training, but only makes one prediction at the last position. Then, for 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. Clearly, this evaluation procedure is extremely expensive.

3.2 Proposed Approach

3.2.1 Segment-Level Recurrence with State Reuse



(a) Training phase. (b) Evaluation phase.

Figure 3.2: 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. 3.2a. 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 $\mathbf{s}_\tau = [x_{\tau,1}, \cdots, x_{\tau,L}]$ and $\mathbf{s}_{\tau+1} = [x_{\tau+1,1}, \cdots, x_{\tau+1,L}]$ respectively. Denoting the n-th layer hidden state sequence produced for the τ-th segment $\mathbf{s}_\tau$ by $\mathbf{h}_\tau^n \in \mathbb{R}^{L\times d}$, where d is the hidden dimension. Then, the n-th layer hidden state for segment $\mathbf{s}_{\tau+1}$ is produced (schematically) as follows,

$$\begin{aligned}
\widetilde{\mathbf{h}}_{\tau+1}^{n-1} &= \left[\text{SG}(\mathbf{h}_\tau^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right], && \text{(extended context)}\\
\mathbf{q}_{\tau+1}^n, \mathbf{k}_{\tau+1}^n, \mathbf{v}_{\tau+1}^n &= \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_q^\top, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_k^\top, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_v^\top, && \text{(query, key, value vectors)}\\
\mathbf{h}_{\tau+1}^n &= \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^n, \mathbf{k}_{\tau+1}^n, \mathbf{v}_{\tau+1}^n\right). && \text{(self-attention + feed-forward)}
\end{aligned}$$

where the function $\text{SG}(\cdot)$ stands for stop-gradient, the notation $[\mathbf{h}_u \circ \mathbf{h}_v]$ indicates the concatenation of two hidden sequences along the length dimension, and $\mathbf{W}_\cdot$ denotes model parameters. Compared to the standard Transformer, the critical difference lies in that the key $\mathbf{k}_{\tau+1}^n$ and value $\mathbf{v}_{\tau+1}^n$ are conditioned on the extended context $\widetilde{\mathbf{h}}_{\tau+1}^{n-1}$ and hence $\mathbf{h}_\tau^{n-1}$ cached from the previous segment. We emphasize this particular design by the green paths in Fig. 3.2a.

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 $\mathbf{h}_{\tau+1}^n$ and $\mathbf{h}_\tau^{n-1}$ 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. 3.2b. This is analogous to truncated BPTT [103], 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.

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.

Moreover, 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/TPU 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 $\mathbf{m}_\tau^n \in \mathbb{R}^{M\times d}$, due to a clear connection to the memory augmented neural networks [51, 162]. In our experiments, we set M equal to the segment length during training, and increase it by multiple times during evaluation.

3.2.2 Relative Positional Encodings

While we found the idea presented above 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 $\mathbf{U} \in \mathbb{R}^{L_{\max}\times d}$, where the i-th row $\mathbf{U}_i$ corresponds to the i-th *absolute* position within a segment and

$L_{\max}$ 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

$$\mathbf{h}_{\tau+1} = f(\mathbf{h}_\tau, \mathbf{E}_{\mathbf{s}_{\tau+1}} + \mathbf{U}_{1:L}) \quad \text{and} \quad \mathbf{h}_\tau = f(\mathbf{h}_{\tau-1}, \mathbf{E}_{\mathbf{s}_\tau} + \mathbf{U}_{1:L}),$$

where $\mathbf{E}_{\mathbf{s}_\tau} \in \mathbb{R}^{L\times d}$ is the word embedding sequence of $\mathbf{s}_\tau$, and f represents a transformation function. Notice that, both $\mathbf{E}_{\mathbf{s}_\tau}$ and $\mathbf{E}_{\mathbf{s}_{\tau+1}}$ are associated with the same positional encoding $\mathbf{U}_{1:L}$. 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 $q_{\tau,i}$ attends on the key vectors $\mathbf{k}_{\tau,\leq i}$, 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 $k_{\tau,j}$ and itself $q_{\tau,i}$, i.e. $i - j$. Practically, one can create a set of relative positional encodings $\mathbf{R} \in \mathbb{R}^{L_{\max}\times d}$, where the i-th row $\mathbf{R}_i$ 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 $x_{\tau,j}$ and $x_{\tau+1,j}$ 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 [136] and music generation [61]. 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. Firstly, in the standard Transformer [156], the attention score between query q_i and key vector k_j within the same segment can be decomposed as

$$\mathbf{A}^{\text{abs}}_{i,j} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)}.$$

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

$$\mathbf{A}^{\text{rel}}_{i,j} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}.$$

- The first change we make is to replace all appearances of the absolute positional embedding $\mathbf{U}_j$ for computing key vectors in term (b) and (d) with its relative counterpart $\mathbf{R}_{i-j}$. This essentially reflects the prior that only the relative distance matters for where to attend. Note that $\mathbf{R}$ is a sinusoid encoding matrix [156] without learnable parameters.

- Secondly, we introduce a trainable parameter $u \in \mathbb{R}^d$ to replace the query $\mathbf{U}_i^\top \mathbf{W}_q^\top$ in term (c). With the query vector being 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 $\mathbf{U}_i^\top \mathbf{W}_q^\top$ in term (d).
- Finally, we deliberately separate the two weight matrices $\mathbf{W}_{k,E}$ and $\mathbf{W}_{k,R}$ 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. [136] only has terms (a) and (b), dropping the two bias terms (c) and (d). Moreover, Shaw et al. [136] merge the multiplication $\mathbf{W}_k \mathbf{R}$ into a single trainable matrix $\hat{\mathbf{R}}$, which abandons the inductive bias built into the original sinusoid positional encoding [156]. In contrast, our relative positional embedding $\mathbf{R}$ 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 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 $n = 1, \ldots, N$:

$$
\begin{aligned}
\widetilde{\mathbf{h}}_\tau^{n-1} &= \Big[\mathrm{SG}(\mathbf{m}_\tau^{n-1}) \circ \mathbf{h}_\tau^{n-1}\Big] \\
\mathbf{q}_\tau^n, \mathbf{k}_\tau^n, \mathbf{v}_\tau^n &= \mathbf{h}_\tau^{n-1}\mathbf{W}_q^{n\top}, \widetilde{\mathbf{h}}_\tau^{n-1}\mathbf{W}_{k,E}^{n\ \top}, \widetilde{\mathbf{h}}_\tau^{n-1}\mathbf{W}_v^{n\top} \\
\mathbf{A}_{\tau,i,j}^n &= \mathbf{q}_{\tau,i}^{n\ \top}\mathbf{k}_{\tau,j}^n + \mathbf{q}_{\tau,i}^{n\ \top}\mathbf{W}_{k,R}^n\mathbf{R}_{i-j} + u^\top \mathbf{k}_{\tau,j} + v^\top \mathbf{W}_{k,R}^n \mathbf{R}_{i-j} \\
\mathbf{a}_\tau^n &= \text{Masked-Softmax}(\mathbf{A}_\tau^n)\mathbf{v}_\tau^n \\
\mathbf{o}_\tau^n &= \text{LayerNorm}\Big(\text{Linear}(\mathbf{a}_\tau^n) + \mathbf{h}_\tau^{n-1}\Big) \\
\mathbf{h}_\tau^n &= \text{LayerNorm}(\text{Positionwise-Feed-Forward}(\mathbf{o}_\tau^n) + \mathbf{o}_\tau^n)
\end{aligned}
$$

with $\mathbf{h}_\tau^0 \coloneqq \mathbf{E}_{\mathbf{s}_\tau}$ defined as the word embedding sequence. In addition, it is worth mentioning that a naive way to compute $\mathbf{A}$ requires computing $\mathbf{W}_{k,R}^n \mathbf{R}_{i-j}$ 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 can reduce the cost to be linear w.r.t. the sequence length.

3.3 Empirical Evaluation for Density Estimation

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 [99], enwik8 [95], text8 [95], One Billion Word [22], and Penn Treebank [102].

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 [5, 48]. As shown in Table 3.1, Transformer-XL reduces the previous

Model	#Param	PPL
Grave et al. [49] - LSTM	-	48.7
Bai et al. [7] - TCN	-	45.2
Dauphin et al. [35] - GCNN-8	-	44.9
Grave et al. [49] - LSTM + Neural cache	-	40.8
Dauphin et al. [35] - GCNN-14	-	37.2
Merity et al. [101] - QRNN	151M	33.0
Rae et al. [124] - Hebbian + Cache	-	29.9
Ours - Transformer-XL Standard	151M	**24.0**
Baevski and Auli [5] - Adaptive Input$^{\diamond}$	247M	20.5
Ours - Transformer-XL Large	257M	**18.3**

Table 3.1: Comparison with state-of-the-art results on WikiText-103. $^{\diamond}$ indicates contemporary work.

Model	#Param	bpc
Ha et al. [54] - LN HyperNetworks	27M	1.34
Chung et al. [27] - LN HM-LSTM	35M	1.32
Zilly et al. [180] - RHN	46M	1.27
Mujika et al. [109] - FS-LSTM-4	47M	1.25
Krause et al. [77] - Large mLSTM	46M	1.24
Knol [74] - cmix v13	-	1.23
Al-Rfou et al. [2] - 12L Transformer	44M	1.11
Ours - 12L Transformer-XL	41M	**1.06**
Al-Rfou et al. [2] - 64L Transformer	235M	1.06
Ours - 18L Transformer-XL	88M	1.03
Ours - 24L Transformer-XL	277M	**0.99**

Table 3.2: Comparison with state-of-the-art results on enwik8.

Model	#Param	bpc
Cooijmans et al. [29] - BN-LSTM	-	1.36
Chung et al. [27] - LN HM-LSTM	35M	1.29
Zilly et al. [180] - RHN	45M	1.27
Krause et al. [77] - Large mLSTM	45M	1.27
Al-Rfou et al. [2] - 12L Transformer	44M	1.18
Al-Rfou et al. [2] - 64L Transformer	235M	1.13
Ours - 24L Transformer-XL	277M	**1.08**

Table 3.3: Comparison with state-of-the-art results on text8.

state-of-the-art (SoTA) perplexity from 20.5 to 18.3, which demonstrates the superiority of the Transformer-XL architecture.

Model	#Param	PPL
Shazeer et al. [137] - Sparse Non-Negative	33B	52.9
Chelba et al. [22] - RNN-1024 + 9 Gram	20B	51.3
Kuchaiev and Ginsburg [79] - G-LSTM-2	-	36.0
Dauphin et al. [35] - GCNN-14 bottleneck	-	31.9
Jozefowicz et al. [67] - LSTM	1.8B	30.6
Jozefowicz et al. [67] - LSTM + CNN Input	1.04B	30.0
Shazeer et al. [138] - Low-Budget MoE	∼5B	34.1
Shazeer et al. [138] - High-Budget MoE	∼5B	28.0
Shazeer et al. [139] - Mesh Tensorflow	4.9B	24.0
Baevski and Auli [5] - Adaptive Input$^{\diamond}$	0.46B	24.1
Baevski and Auli [5] - Adaptive Input$^{\diamond}$	1.0B	23.7
Ours - Transformer-XL Base	0.46B	23.5
Ours - Transformer-XL Large	0.8B	**21.8**

Table 3.4: Comparison with state-of-the-art results on One Billion Word. $^{\diamond}$ indicates contemporary work.

Model	#Param	PPL
Inan et al. [64] - Tied Variational LSTM	24M	73.2
Zilly et al. [180] - Variational RHN	23M	65.4
Zoph and Le [181] - NAS Cell	25M	64.0
Merity et al. [100] - AWD-LSTM	24M	58.8
Pham et al. [117] - Efficient NAS	24M	58.6
Liu et al. [91] - Differentiable NAS	23M	56.1
Yang et al. [169] - AWD-LSTM-MoS	22M	55.97
Melis et al. [98] - Dropout tuning	24M	55.3
Ours - Transformer-XL	24M	**54.52**
Merity et al. [100] - AWD-LSTM+Finetune†	24M	57.3
Yang et al. [169] - MoS+Finetune†	22M	**54.44**

Table 3.5: Comparison with state-of-the-art results on PTB. † indicates using two-step finetuning.

The dataset enwik8 contains 100M bytes of unprocessed Wikipedia text. We compare our architecture with the previous results in Table 3.2. 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. [2] 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. [2], 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. [2], 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 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 3.3. 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 3.4. 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 [5], suggesting the advantage of Transformer-XL is generalizable to modeling short sequences.

We also report the results on word-level Penn Treebank in Table 3.5. Similar to AWD-LSTM [100], we apply variational dropout and weight average to Transformer-XL. With proper 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.

Given the substantial performance improvement of Transformer-XL for density estimation, we next consider how to extend its effectiveness to representation learning and data generation.

Chapter 4

XLNet: Generalized Autoregressive Pretraining for Language Understanding

4.1 Motivations

Unsupervised representation learning has been highly successful in the domain of natural language processing [31, 36, 96, 116, 122]. 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 [31, 116, 122]. Specifically, given a text sequence $\mathbf{x} = (x_1, \cdots, x_T)$, AR language modeling factorizes the likelihood into a forward product $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid \mathbf{x}_{<t})$ or a backward one $p(\mathbf{x}) = \prod_{t=T}^{1} p(x_t \mid \mathbf{x}_{>t})$. 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 [36], 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 `[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 `[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 [34].

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.

4.2 Proposed Method

4.2.1 Background

To better motivate the proposed approach, we first review and compare the conventional AR language modeling and BERT for language pretraining and then detail the proposed generalized autoregressive pretraining objective. Given a text sequence $\mathbf{x} = [x_1, \cdots, x_T]$, AR language modeling performs pretraining by maximizing the likelihood under the forward autoregressive factorization:

$$\max_{\theta} \quad \log p_\theta(\mathbf{x}) = \sum_{t=1}^{T} \log p_\theta(x_t \mid \mathbf{x}_{<t}) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x')\right)}, \tag{4.1}$$

where $h_\theta(\mathbf{x}_{1:t-1})$ is a context representation produced by neural models, such as RNNs or Transformers, and $e(x)$ denotes the embedding of x. After pretraining, the neural model h_θ can be directly used to extract static features for other applications as in ELMo [116] or further finetuned on downstream task to produce task specific representation as in GPT-1 [122].

In comparison, BERT is based on denoising auto-encoding. Specifically, for a text sequence $\mathbf{x}$, BERT first constructs a corrupted version $\hat{\mathbf{x}}$ by randomly setting a portion (e.g. 15%) of tokens in $\mathbf{x}$ to a special symbol `[MASK]`. Let the set of masked tokens be $\bar{\mathbf{x}}$. The training objective is to reconstruct $\bar{\mathbf{x}}$ from the corrupted input sequence $\hat{\mathbf{x}}$:

$$\max_{\theta} \quad \log p_\theta(\bar{\mathbf{x}} \mid \hat{\mathbf{x}}) \approx \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{\mathbf{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x')\right)}, \tag{4.2}$$

where $m_t = 1$ indicates x_t is masked, and H_θ is a Transformer that maps a length-T text sequence $\mathbf{x}$ into a sequence of hidden vectors $H_\theta(\mathbf{x}) = [H_\theta(\mathbf{x})_1, H_\theta(\mathbf{x})_2, \cdots, H_\theta(\mathbf{x})_T]$.

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. (4.2), BERT factorizes the joint conditional probability $p(\bar{\mathbf{x}} \mid \hat{\mathbf{x}})$ based on an independence assumption that all masked tokens $\bar{\mathbf{x}}$ are separately reconstructed. In comparison, the AR language modeling objective (4.1) factorizes $p_\theta(\mathbf{x})$ using the product rule that holds universally without such an independence assumption.
- **Input noise**: The input to BERT contains artificial symbols like `[MASK]` that never occur in downstream tasks, which creates a pretrain-finetune discrepancy. Replacing `[MASK]` with original tokens as in [36] does not solve the problem because original tokens can be

only used with a small probability — otherwise Eq. (4.2) 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 $h_\theta(\mathbf{x}_{1:t-1})$ is only conditioned on the tokens up to position t (i.e. tokens to the left), while the BERT representation $H_\theta(\mathbf{x})_t$ 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.

4.2.2 Objective: Permutation Language Modeling

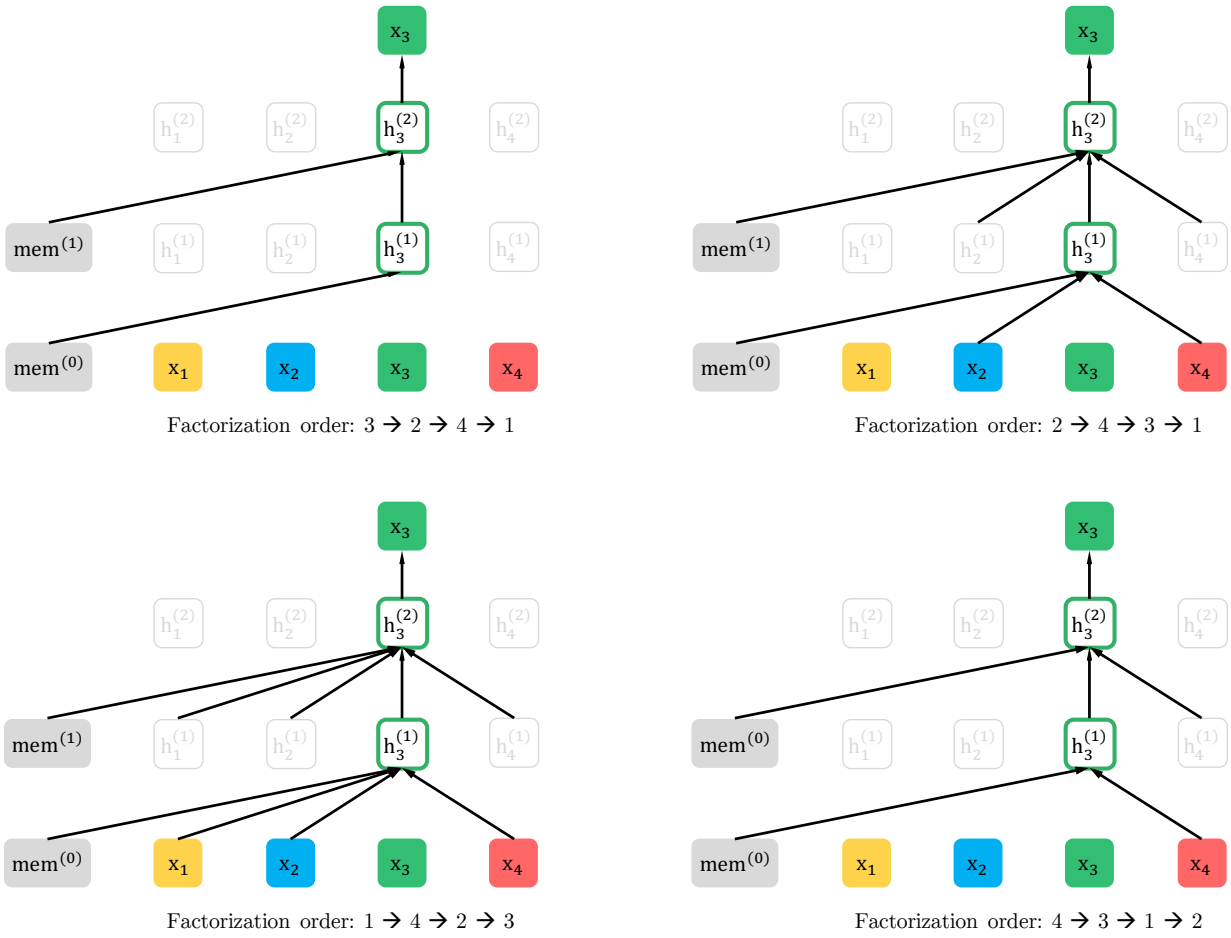


Figure 4.1: Illustration of the permutation language modeling objective for predicting x_3 given the same input sequence $\mathbf{x}$ but with different factorization orders.

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.

Borrowing ideas from orderless NADE [153], 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 $\mathbf{x}$ 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 $\mathcal{Z}_T$ be the set of all possible permutations of the length-T index sequence $[1, 2, \ldots, T]$. We use z_t and $\mathbf{z}_{<t}$ to denote the t-th element and the first $t-1$ elements of a permutation $\mathbf{z} \in \mathcal{Z}_T$. Then, our proposed permutation language modeling objective can be expressed as follows:

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=1}^{T} \log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right]. \tag{4.3}$$

Essentially, for a text sequence $\mathbf{x}$, we sample a factorization order $\mathbf{z}$ at a time and decompose the likelihood $p_\theta(\mathbf{x})$ according to factorization order. Since the same model parameter θ is shared across all factorization orders during training, in expectation, x_t has seen every possible element $x_i \neq x_t$ in the sequence, hence being able to capture the bidirectional context. Moreover, as this objective fits into the AR framework, it naturally avoids the independence assumption and the pretrain-finetune discrepancy discussed in Section 4.2.1.

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 $\mathbf{x}$ but under different factorization orders in Figure 4.1.

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

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 $p_\theta(X_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})$ using the standard Softmax formulation, i.e., $p_\theta(X_{z_t} = x \mid \mathbf{x}_{\mathbf{z}_{<t}}) = \frac{\exp\left(e(x)^\top h_\theta(\mathbf{x}_{\mathbf{z}_{<t}})\right)}{\sum_{x'} \exp\left(e(x')^\top h_\theta(\mathbf{x}_{\mathbf{z}_{<t}})\right)}$, where $h_\theta(\mathbf{x}_{\mathbf{z}_{<t}})$ denotes the hidden representation of $\mathbf{x}_{\mathbf{z}_{<t}}$ produced by the shared Transformer network after proper masking. Now notice that the representation $h_\theta(\mathbf{x}_{\mathbf{z}_{<t}})$ 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. To avoid this problem, we propose to re-parameterize the next-token distribution to be target position aware:

$$p_\theta(X_{z_t} = x \mid \mathbf{x}_{z_{<t}}) = \frac{\exp\left(e(x)^\top g_\theta(\mathbf{x}_{\mathbf{z}_{<t}}, z_t)\right)}{\sum_{x'} \exp\left(e(x')^\top g_\theta(\mathbf{x}_{\mathbf{z}_{<t}}, z_t)\right)}, \tag{4.4}$$

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

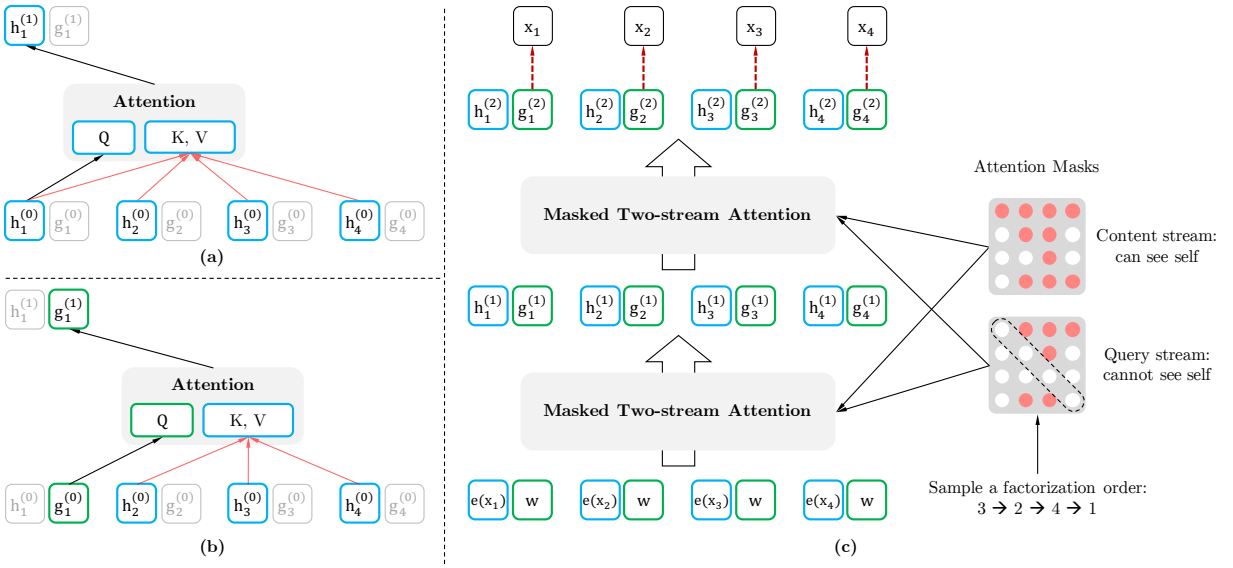


Figure 4.2: (a): Content stream attention, which is the same as the standard self-attention. (b): Query stream attention, which does not have access information about the content x_{z_t}. (c): Overview of the permutation language modeling training with two-stream attention.

Two-Stream Self-Attention While the idea of target-aware representations removes the ambiguity in target prediction, how to formulate $g_\theta(\mathbf{x}_{\mathbf{z}<t}, z_t)$ 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 $\mathbf{x}_{\mathbf{z}<t}$ 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_{z_t}, $g_\theta(\mathbf{x}_{\mathbf{z}<t}, z_t)$ should only use the *position* z_t and not the *content* x_{z_t}, otherwise the objective becomes trivial; (2) to predict the other tokens x_{z_j} with $j > t$, $g_\theta(\mathbf{x}_{\mathbf{z}<t}, z_t)$ should also encode the content x_{z_t} 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 $h_\theta(\mathbf{x}_{\mathbf{z}\leq t})$, or abbreviated as h_{z_t}, which serves a similar role to the standard hidden states in Transformer. This representation encodes *both* the context and x_{z_t} itself.
- The query representation $g_\theta(\mathbf{x}_{\mathbf{z}<t}, z_t)$, or abbreviated as g_{z_t}, which only has access to the contextual information $\mathbf{x}_{\mathbf{z}<t}$ and the position z_t, but not the content x_{z_t}, 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*[1] updated with a shared set of parameters as follows (illustrated in Figures 4.2 (a) and (b)):

$$g_{z_t}^{(m)} \leftarrow \text{Attention}(\text{Q} = g_{z_t}^{(m-1)}, \text{KV} = \mathbf{h}_{\mathbf{z}<t}^{(m-1)}; \theta), \quad \text{(query stream: use } z_t \text{ but cannot see } x_{z_t})$$
$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\text{Q} = h_{z_t}^{(m-1)}, \text{KV} = \mathbf{h}_{\mathbf{z}\leq t}^{(m-1)}; \theta), \quad \text{(content stream: use both } z_t \text{ and } x_{z_t}),$$

[1]To 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).

where Q, K, V denote the query, key, and value in an attention operation [156]. The update rule of the content representations is exactly the same as the standard self-attention, so during finetuning, 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. (4.4).

Partial Prediction While the permutation language modeling objective (4.3) 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 $\mathbf{z}$ into a non-target subsequence $\mathbf{z}_{\leq c}$ and a target subsequence $\mathbf{z}_{>c}$, 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.,

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\Big[\log p_\theta(\mathbf{x}_{\mathbf{z}_{>c}} \mid \mathbf{x}_{\mathbf{z}_{\leq c}})\Big] = \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=c+1}^{|\mathbf{z}|} \log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right]. \tag{4.5}$$

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

4.2.4 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 [34], 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 $\mathbf{s}$; i.e., $\tilde{\mathbf{x}} = \mathbf{s}_{1:T}$ and $\mathbf{x} = \mathbf{s}_{T+1:2T}$. Let $\tilde{\mathbf{z}}$ and $\mathbf{z}$ be permutations of $[1\cdots T]$ and $[T+1\cdots 2T]$ respectively. Then, based on the permutation $\tilde{\mathbf{z}}$, we process the first segment, and then cache the obtained content representations $\tilde{\mathbf{h}}^{(m)}$ for each layer m. Then, for the next segment $\mathbf{x}$, the attention update with memory can be written as

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

where $[.,.]$ 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 $\tilde{\mathbf{z}}$ once the representations $\tilde{\mathbf{h}}^{(m)}$ 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 4.2 (c) presents an overview of the proposed permutation language modeling with two-stream attention (see Appendix **??** for more detailed illustration).

4.2.5 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 [36] as it does not show consistent improvement in our ablation study (see Section 5.3.3).

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 $\mathbf{s}_{ij} = \mathbf{s}_+$ or otherwise $\mathbf{s}_{ij} = \mathbf{s}_-$, where $\mathbf{s}_+$ and $\mathbf{s}_-$ 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 $\mathbf{s}_{ij}$ is used to compute an attention weight $a_{ij} = (\mathbf{q}_i + \mathbf{b})^\top \mathbf{s}_{ij}$, where $\mathbf{q}_i$ is the query vector as in a standard attention operation and $\mathbf{b}$ 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 [34]. Second, it opens the possibility of finetuning on tasks that have more than two input segments, which is not possible using absolute segment encodings.

4.2.6 Discussion and Analysis

Comparison with BERT

Comparing Eq. (4.2) and (4.5), 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 4.2.1 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 $\log p(\text{New York} \mid \text{is a city})$. 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:

$$\mathcal{J}_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city}),$$

$$\mathcal{J}_{\text{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New}, \text{is a city}).$$

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 [169], given a sequence $\mathbf{x} = [x_1, \cdots, x_T]$, we define a set of target-context pairs of interest, $\mathcal{I} = \{(x, \mathcal{U})\}$, where $\mathcal{U}$ is a set of tokens in $\mathbf{x}$ 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:

$$\mathcal{I} = \Big\{\big(x = \text{York}, \mathcal{U} = \{\text{New}\}\big),\ \big(x = \text{York}, \mathcal{U} = \{\text{city}\}\big),\ \big(x = \text{York}, \mathcal{U} = \{\text{New, city}\}\big),\ \cdots\Big\}.$$

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 $\mathcal{N} = \mathbf{x} \backslash \mathcal{T}$, BERT and XLNet both maximize $\log p(\mathcal{T} \mid \mathcal{N})$ but with different formulations:

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

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 4.3.

Comparison with Language Modeling

Borrowing examples and notations from Section 4.2.6, a standard AR language model like GPT [122] 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 [116] concatenate forward and backward language models in a shallow manner, which is not sufficient for modeling deep interactions between the two directions.

4.3 Experiments

4.3.1 Pretraining and Implementation

Following BERT [36], we use the BooksCorpus [178] and English Wikipedia as part of our pretraining data, which have 13GB plain text combined. In addition, we include Giga5 (16GB text) [114], ClueWeb 2012-B (extended from [20]), and Common Crawl [30] 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 with SentencePiece [80], 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 5.3.3.

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 4.2.3). Our finetuning procedure follows BERT [36] 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.

4.3.2 RACE Dataset

The RACE dataset [81] 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 [127]. As a result, this dataset serves as a challenging benchmark for long text

RACE	Accuracy	Middle	High
GPT [122]	59.0	62.9	57.4
BERT [113]	72.0	76.6	70.1
BERT+OCN* [128]	73.5	78.4	71.5
BERT+DCMN* [173]	74.1	79.5	71.8
XLNet	**81.75**	**85.45**	**80.21**

Table 4.1: 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.

understanding. We use a sequence length of 640 during finetuning. As shown in Table 5.5, a 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 5.3.3.

4.3.3 SQuAD Dataset

SQuAD1.1	EM	F1	SQuAD2.0	EM	F1
Dev set results without data augmentation					
BERT [36]	84.1	90.9	BERT† [36]	78.98	81.77
XLNet	**88.95**	**94.52**	XLNet	**86.12**	**88.79**
Test set results on leaderboard, with data augmentation (as of June 19, 2019)					
Human [126]	82.30	91.22	BERT+N-Gram+Self-Training [36]	85.15	87.72
ATB	86.94	92.64	SG-Net	85.23	87.93
BERT* [36]	87.43	93.16	BERT+DAE+AoA	85.88	88.62
XLNet	**89.90**	**95.08**	XLNet	**86.35**	**89.13**

Table 4.2: 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.

SQuAD is a large-scale reading comprehension dataset with two tasks. SQuAD1.1 [126] contains questions that always have a corresponding answer in the given passages, while SQuAD2.0 [127] 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 [36]. 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 [149] for our leaderboard submission. As shown in Table 4.2, 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.

4.3.4 Text Classification

Model	IMDB	Yelp-2	Yelp-5	DBpedia	AG	Amazon-2	Amazon-5
CNN [66]	-	2.90	32.39	0.84	6.57	3.79	36.24
DPCNN [66]	-	2.64	30.58	0.88	6.87	3.32	34.81
Mixed VAT [106, 133]	4.32	-	-	0.70	4.95	-	-
ULMFiT [60]	4.6	2.16	29.98	0.80	5.01	-	-
BERT [165]	4.51	1.89	29.32	0.64	-	2.63	34.17
XLNet	**3.79**	**1.55**	**27.80**	**0.62**	**4.49**	**2.40**	**32.26**

Table 4.3: 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).

Following previous work on text classification [106, 174], we evaluate XLNet on the following benchmarks: IMDB, Yelp-2, Yelp-5, DBpedia, AG, Amazon-2, and Amazon-5. According to Table 4.3, XLNet achieves new state-of-the-art results on all the considered datasets, reducing the error rate by 16%, 18%, 5%, 9% and 5% on IMDB, Yelp-2, Yelp-5, Amazon-2, and Amazon-5 respectively compared to BERT.

4.3.5 GLUE Dataset

The GLUE dataset [159] 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 4.4, 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 [92] 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 loss described in [75]. 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

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI
Single-task single models on dev									
BERT [3]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-
XLNet	**89.8/-**	**93.9**	**91.8**	**83.8**	**95.6**	**89.2**	**63.6**	**91.8**	-
Single-task single models on test									
BERT [36]	86.7/85.9	91.1	89.3	70.1	94.9	89.3	60.5	87.6	65.1
Multi-task ensembles on test (from leaderboard as of June 19, 2019)									
Snorkel* [130]	87.6/87.2	93.9	89.9	80.9	96.2	91.5	63.8	90.1	65.1
ALICE*	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8
MT-DNN* [92]	87.9/87.4	96.0	89.9	**86.3**	96.5	92.7	68.4	91.1	89.0
XLNet*	**90.2/89.7**†	**98.6**†	90.3†	**86.3**	**96.8**†	**93.0**	67.8	**91.6**	**90.4**

Table 4.4: 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.

Model	NDCG@20	ERR@20
DRMM [53]	24.3	13.8
KNRM [32]	26.9	14.9
Conv [32]	28.7	18.1
BERT†	30.53	18.67
XLNet	**31.10**	**20.28**

Table 4.5: Comparison with state-of-the-art results on the test set of ClueWeb09-B, a document ranking task. † indicates our implementations.

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 4.4, 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.

4.3.6 ClueWeb09-B Dataset

Following the setting in previous work [32], 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

#	Model	RACE	SQuAD2.0		MNLI	SST-2
			F1	EM	m/mm	
1	BERT-Base	64.3	76.30	73.66	84.34/84.65	92.78
2	DAE + Transformer-XL	65.03	79.56	76.80	84.88/84.45	92.60
3	XLNet-Base ($K = 7$)	66.05	**81.33**	**78.46**	**85.84/85.43**	92.66
4	XLNet-Base ($K = 6$)	66.66	80.98	78.18	85.63/85.12	**93.35**
5	- memory	65.55	80.15	77.27	85.32/85.05	92.78
6	- span-based pred	65.95	80.61	77.91	85.49/85.02	93.12
7	- bidirectional data	66.34	80.65	77.87	85.31/84.99	92.66
8	+ next-sent pred	**66.76**	79.83	76.94	85.32/85.09	92.89

Table 4.6: 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 4.2.3). All models are pretrained on the same data.

the documents and queries without finetuning, and employ a kernel pooling network [167] to rank the documents. According to Table 4.5, 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 [166] as it uses additional entity-related data.

4.3.7 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 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 C.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 C.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 5

Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing

5.1 Introduction

With the recent success of unsupervised language pretraining [28, 36, 76, 84, 88, 92, 93, 116, 125, 142, 143, 171], the power of neural self-attention models (a.k.a. Transformer) [156] has been pushed to a new level, leading to dramatic advancements in machine learning and natural language processing (NLP). More importantly, it has been observed that with more FLOPs invested in longer pretraining and/or larger models, the performance of pretrained Transformer models consistently improve. However, it is extremely expensive to pretrain or even just to finetune the state-of-the-art self-attention models, as they require much more FLOPs and memory resources compared to traditional models in NLP. This largely limits their applications and success in more fields.

Given this challenge, there has been an increasing amount of efforts to reduce the costs of pretraining and finetuning self-attention models. From the perspective of post-pretraining processing, typical approaches include distillation, pruning and quantization of various kinds, which try to derive a lighter model from an well-pretrained model by taking advantage of the richer signals in the larger model or learning to remove less important operations. Another line of research aims at designing an architecture that not only has a lower resource-to-performance ratio (more efficient) but also *scales as well as* the Transformer, at least in certain domains. Most of such methods build upon the Transformer backbone and focus on redesigning its building blocks. Representative solutions include searching for better micro operation or macro module designs [23, 140], replacing the full pairwise attention with local operations such as convolution [164] and dynamic convolution [163], and optimizing the hidden size combinations for existing blocks [146].

Across the wide variety of ideas mentioned above, a common strategy is to identify redundant operations or representations and replace them with more efficient ones. Inspired by this line of thinking, in this chapter, we will be focusing on the potential redundancy induced by always maintaining a *full-length sequence* of hidden representations across all layers in Transformer. Intuitively, for many sequence-level NLP tasks such as text classification and ranking, the most

common use case is to extract a *single* vector from the entire sequence, which does not necessarily preserve all information down to the token-level granularity. Hence, for such tasks, the full-length sequence of hidden states may contain significant redundancy. This is analogous to the case of image recognition, where the convolution neural network gradually reduces the spatial resolution/size of feature maps as the neural network goes deeper. In addition, linguistic prior also encourages gradually merging nearby tokens (words) into larger semantic units (phrases), which naturally leads to a shorter sequence of representations.

Concretely, we propose to gradually reduce the sequential resolution (i.e. length) of the hidden representation in self-attention models. Immediately, the reduction in sequence length can lead to significant savings in both FLOPs and memory. More importantly, the saved computational resource can be directly re-invested in constructing a deeper (or wider) model to boost the model capacity without additional computational burden. In addition, to address the challenge that common pretraining objectives such as masked language modeling (MLM) [36] require separate representations for each token, we design a simple strategy to decode a full-length sequence of deep representations from the hidden state of reduced length. As a result, the proposed model can be directly trained without modifying the pretraining objectives, as well as adopted for downstream tasks that require token-level representations.

Empirically, with comparable or even fewer FLOPs, by trading sequential resolution for depth, our proposed model achieves an improved performance over the standard Transformer on a wide variety of sequence-level prediction tasks, including text classification, language understanding, and reading comprehension.

5.2 Method

5.2.1 Background

Transformer Architecture The Transformer architecture [156] is a highly modularized neural network, where each Transformer layer consists of two sub-modules, namely the multi-head self-attention (S-Attn) and position-wise feed-forward network (P-FFN). Both sub-modules are wrapped by a residual connection and layer normalization. Schematically, given a length T sequence of hidden states $\mathbf{h} = [h_1, \ldots, h_T]$, the computation of a single Transformer layer can be expressed as

$$\mathbf{h} \leftarrow \text{LayerNorm}(\mathbf{h} + \text{S-Attn}(\text{Q} = \mathbf{h}, \text{KV} = \mathbf{h})), \tag{5.1}$$

$$h_i \leftarrow \text{LayerNorm}(h_i + \text{P-FFN}(h_i)), \quad \forall i = 1, \cdots, T. \tag{5.2}$$

Pretraining Objectives The most commonly used pretraining objective is the masked language modeling (MLM) proposed by BERT [36]. For a length-T natural language sequence $\mathbf{x}$ sample from a large unlabeled set $\mathcal{D}$, the MLM objective first constructs a corrupted sequence $\hat{\mathbf{x}}$ by randomly replacing 15% of the tokens of $\mathbf{x}$ with a special token `[MASK]` and then trains a Transformer model [36] to reconstruct the original $\mathbf{x}$ based on $\hat{\mathbf{x}}$, i.e.,

$$\max_{\theta} \ \mathcal{J}_{\text{MLM}}(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\mathbb{E}_{\mathcal{I}} \sum_{i\in\mathcal{I}} \log P_\theta(x_i \mid \hat{\mathbf{x}}_{\mathcal{I}}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\mathbb{E}_{\mathcal{I}} \sum_{i\in\mathcal{I}} \log \frac{\exp\left(e(x_i)^\top h_i(\hat{\mathbf{x}}_{\mathcal{I}})\right)}{\sum_{x'} \exp\left(e(x')^\top h_i(\hat{\mathbf{x}}_{\mathcal{I}})\right)},$$

where $\mathcal{I}$ is the positions of masked tokens, the subscript in $\hat{\mathbf{x}}_{\mathcal{I}}$ emphasizes its dependence on $\mathcal{I}$, $e(x)$ denotes the embedding of the token x, and $h_i(\hat{\mathbf{x}}_{\mathcal{I}})$ the last-layer hidden state at position i produced by the Transformer model. After pretraining, the entire model is finetuned in downstream tasks.

To show the generality of our proposed model, we also experiment with another pretraining objective ELECTRA [28]. Different from MLM, ELECTRA relies a pair of jointly trained generator and discriminator. Specifically, the generator usually has a smaller size (1/4 of that of the discriminator) and is directly trained via the MLM objective, i.e., $\max_{\theta_G} \mathcal{J}_{\text{MLM}}(\theta_G)$. Then, for each masked position, a token is sampled from the reconstruction distribution of the generator to replace the `[MASK]`token and form a new sequence $\tilde{\mathbf{x}}$, i.e., if $i \in \mathcal{I}$, $\tilde{x}_i \sim P_{\theta_G}(x_i \mid \hat{\mathbf{x}}_{\mathcal{I}})$ else $\tilde{x}_i = x_i$. Given the new sequence $\tilde{\mathbf{x}}$, the discriminator is then trained to distinguish whether each token in $\tilde{\mathbf{x}}$ is real (same as $\mathbf{x}$) or fake (different from $\mathbf{x}$) via binary classification. After pretraining, only the discriminator will be used during finetuning and the generator is simply discarded.

Discussion Note that both pretraining objectives introduced above require the ability to produce a hidden state for each input token, i.e., $h_i(\hat{\mathbf{x}}_{\mathcal{I}})$ and $h_i(\tilde{\mathbf{x}})$. Due to this requirement, it seems natural to keep a full sequence of hidden states. However, in contrast, many sequence-level downstream tasks like classification or ranking only need a single-vector summary of the entire sequence. Fundamentally, this suggests that some kind of compression is usually required to remove the unnecessary redundancy during finetuning. This observation immediately leads to the following two questions:

- Can we design a general model that is equally expressive but more efficient by compressing the full sequence of hidden states into a more compact form?
- With the compressed representations, how can the model retain the ability to produce token-level representations for pretraining?

To answer these two questions, we next present our proposed architecture.

5.2.2 Proposed Architecture

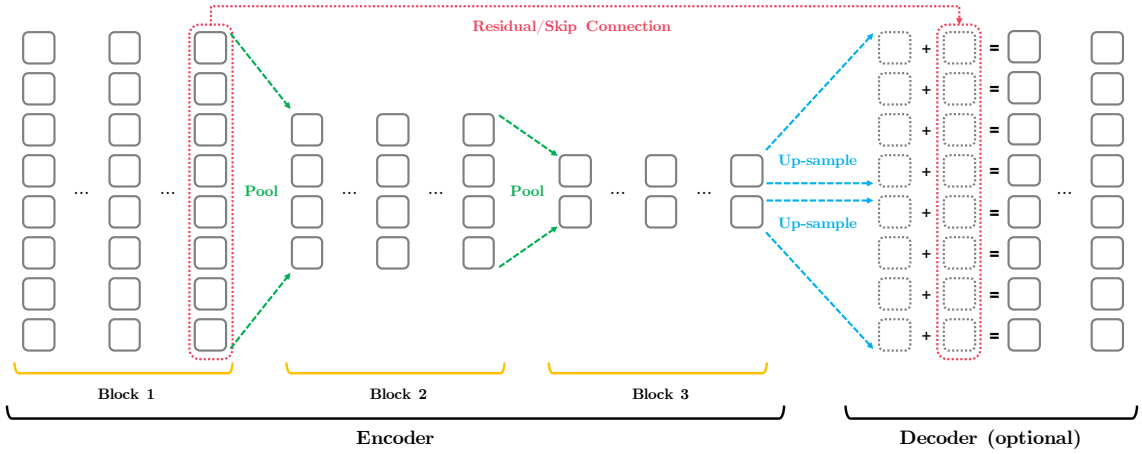


Figure 5.1: High-level visualization of the proposed Funnel-Transformer.

To inherit the high capacity and optimization advantages of the Transformer architecture, the proposed model keeps the same overall skeleton of interleaved S-Attn and P-FFN sub-modules wrapped by residual connection and layer normalization. But differently, to achieve representation compression and computation reduction, our model employs an encoder that gradually reduces the sequence length of the hidden states as the layer gets deeper. In addition, for tasks involving per-token predictions like pretraining, a simple decoder is used to reconstruct a full sequence of token-level representations from the compressed encoder output.

Encoder As illustrated in the left part of Fig. 5.1, the encoder consists of several blocks of consecutive Transformer layers. Within each block, the sequence length of the hidden states always remains the same. But when going from a lower-level block to a higher-level block, the length of the hidden sequence is reduced by performing certain type of pooling along the sequence dimension, i.e.,

$$\mathbf{h}' \leftarrow \text{Pooling}(\mathbf{h}), \tag{5.3}$$

where $\mathbf{h} \in \mathbb{R}^{T \times D}$ and $\mathbf{h}' \in \mathbb{R}^{T' \times D}$ for some $T' < T$. Importantly, instead of directly feeding the pooled sequence $\mathbf{h}'$ into the first S-Attn layer of the new block, we only use pooled sequence $\mathbf{h}'$ to construct the query vector (and the residual signal) of the self-attention, while the unpooled sequence $\mathbf{h}$ serves that role of key and value vectors, i.e.

$$\mathbf{h} \leftarrow \text{LayerNorm}\Big(\mathbf{h}' + \text{S-Attn}\Big(\text{Q} = \mathbf{h}', \text{KV} = \mathbf{h}\Big)\Big). \tag{5.4}$$

Note that the output sequence of this special S-Attn module has the same length as the pooled sequence $\mathbf{h}'$. To understand the advantage of this particular design, it is helpful to compare the proposed "pool-query-only" variant with the naive alternative of using $\mathbf{h}'$ for both the query and key-value vectors, i.e., $\text{S-Attn}\Big(\text{Q} = \mathbf{h}', \text{KV} = \mathbf{h}'\Big)$:

- Under the naive approach, the compression is solely controlled by the pooling operation, which is finished before the attention module. Hence, relatively simple pooling methods such as average/mean pooling won't be able to achieve good compression.
- Under the pool-query-only variant, the compression depends on not only how the pooling is performed, but also how the self-attention weighted sums the unpooled sequence to form each pooled vector. Effectively, the particular attention here can be seen as a type of linear compression that combines T bases into a smaller number of T' "compressed bases". Therefore, with minimum computational overhead, this variant makes compression operation more expressive.

With this particular pool-query-only design in place, we find the simplest strided mean pooling applied to each sliding window of the sequence work very well in practice. For simplicity, we only experiment with stride 2 and window size 2 in this work. Hence, the pooling operation will reduce the sequence by half and each pooled hidden state corresponds to a window of 2 unpooled hidden vectors. Intuitively, this type of pooling roughly follows the linguistic prior that nearby tokens could be gradually merged (or compressed) into a larger semantic component. Once the sequence length is halved after the pooling and pool-query-only attention, the rest of the encoder computation simply follows the standard updates in Eqn. (5.2) and (5.1).

Finally, as an extra implementation detail, recall that a particular design in language pretraining is to add a special token `[CLS]` to the beginning of the original input sequence, and use the last-layer hidden state corresponding to `[CLS]` (i.e., h_1) as the representation of the sequence. To prevent the pooling from destroying this special structure, we first separate the `[CLS]` hidden state and the rest of hidden states and only apply the pooling to the rest of hidden states.

Decoder In order to recover a full sequence of hidden states from the encoder output of reduced length, a natural idea would be performing some kind of up-sampling. For instance, in image generation or super-resolution, deconvolution (transposed convolution) or parameter-free resizing with bilinear interpolation are often used to increase the spatial resolution of the feature map. Hence, we can simply adapt these ideas from 2D processing to our 1D case and apply proper up-sampling to the encoder output.

However, instead of performing multiple up-samplings with small expansion rate (e.g. increasing the sequence length by 2x each time) as in image domain, we here choose to employ a single up-sampling with a large expansion rate, as shown on the right part of Fig. 5.1. Specifically, given the output sequence $\mathbf{h}^{\mathrm{M}}$ of length $T_M = T/2^{M-1}$ from an M-block encoder, we directly up-sample it to a full-length sequence $\mathbf{h}^{\mathrm{up}} = [h_1^{\mathrm{up}}, \cdots, h_T^{\mathrm{up}}]$ by repeating each hidden vector 2^{M-1} times:

$$\forall i = 1, \cdots, T, \quad h_i^{\mathrm{up}} = h^M_{i//2^{M-1}}, \tag{5.5}$$

where $\cdot//\cdot$ denotes floor division. However, note that every 2^{M-1} consecutive vectors in $\mathbf{h}^{\mathrm{up}}$ are exactly the same and hence do not contain detailed token-level information. Hence, we further extract the last-layer hidden states from the *first block* of the encoder $\mathbf{h}^1$, which still has the full length T and contains the uncompressed token-level information. Then, the lower-level representation $\mathbf{h}^1$ and up-sampled higher-level representation $\mathbf{h}^{\mathrm{up}}$ are added together to form a deep token-level representation $\mathbf{g} = \mathbf{h}^1 + \mathbf{h}^{\mathrm{up}}$. Effectively, this forms a residual/skip connection that enables detailed token information and potentially easier optimization. In addition, we stack a few more Transformer layers upon $\mathbf{g}$ to achieve a better deep fusion of the low-level and high-level features. In this work, we always use 2 Transformer layers in decoder.

It is important to emphasize that the decoder is *only used if the task requires token-level prediction*, such as in standard pretraining or sequence labeling. For tasks that only requires a single vectorial representation of the sequence like classification, the decoder is discarded after pretraining and only the encoder is finetuned. Finally, to emphasize the filtering/compression property of the encoder as well as its shape, we name the proposed model `Funnel-Transformer` (F-TFM).

5.2.3 Complexity & Capacity Analysis

With the architecture design specified, we now analyze how the sequence compression affects the complexity and capacity of the proposed model, especially compared to the standard Transformer.

Firstly, for a Transformer layer with an S-Attn and a P-FFN module of hidden size D, the

complexity of processing a length-T sequence is $O(T^2D+TD^2)$.[1] Hence, every time the sequence length is reduced by half in the encoder, we enjoy a *super-linear* (more than half) complexity drop. In practice, as the $O(TD^2)$ term has a large constant, a near-linear speedup is observed more often. The super-linear effect is more detectable when the sequence length is relatively long like in pretraining. Therefore, given the same FLOPs, we can at least trade a full-length layer in the 1st block for 2^{m-1} layers in the m-th block, which provides an economical way to increase the depth of network.

On the other hand, the capacity of a compressed-length layer is clearly upper-bounded by that of a normal full-length layer. In most cases where the compression is lossy, reducing the sequence length will inevitably lead to capacity drop. The good news is that the capacity drop of a single layer could be well compensated by re-investing the saved FLOPs in stacking more cheaper layers of reduced length or increasing the width of the model.

As a concrete example, for a Transformer of $\text{BERT}_{\text{Base}}$ size, i.e., 12 layers of hidden size 768 (L12H768), we may construct a Funnel-Transformer of 3 blocks where each block has 6 layers of hidden size 768 (B6-6-6H768). Despite having 18 layers in total, when finetuned for classification, the FLOPs of the B6-6-6H768 architecture only corresponds to at most $6 + 6/2 + 6/4 = 10.5$ full-length layers, clearly fewer than that of L12H768. More importantly, as we will show in the experiments, B6-6-6H768 significantly outperforms L12H768. While intuitive, how to construct an optimal block layout given this *depth-length trade-off* remains an open challenge. For this work, we only consider relatively regular layout and leave more systematic studies for future work.

Finally, notice that trading sequential resolution for depth or width has a side effect of increasing the total number of parameters. For instance, B6-6-6H768 has 1.5x Transformer parameters compared to L12H768. In practice, more parameters may increase communication cost in distributed training as well as the memory consumption and memory access time. A simple remedy is to perform certain parameter sharing, as used in ALBERT, to recover the same parameter count. Taking B6-6-6H768 as an example, one may tie the parameters for every two layers in the 2nd and 3rd blocks, denoted as B6-3x2-3x2H768, which gives back the same number of parameters to L12H768. However, parameter sharing could result in performance loss. Fundamentally, this brings us another trade-off between the gain (capacity) and cost (memory and communication cost) of using more parameters, which can be highly device dependent.

5.3 Experiment

In this section, we empirically evaluate the proposed F-TFM by first pretraining it and then finetuning it in downstream tasks. Following previous work, for pretraining, we consider two common settings:

- **Base scale**: Pretraining models for 1M steps with batch size 256 on Wikipedia + Book Corpus. This is the setting used by original BERT [36]. We will rely on this setting to perform fair comparison between F-TFM and the standard Transformer as well as some ablation studies.

[1] Since the corresponding memory complexity is simply $O(T^2 + TD)$, which is always offset by a multiplier $1/D$, we will focus on the computation complexity with the conclusion directly carried through.

- **Large scale**: Pretraining models for 500K steps with batch size 8K on the five datasets used by XLNet [171] and ELECTRA [28] (Wikipedia + Book Corpus + ClueWeb + Gigaword + Common Crawl). We will compare F-TFM trained at this scale with previous state-of-the-art methods.

For finetuning, we mainly focus on sequence-level tasks that only requires a single vectorial representation of the input sequence, since F-TFM is designed with such a purpose in mind. Specifically, such tasks include the GLUE benchmark for language understanding [158], 7 widely used text (sentiment / topic) classification tasks (IMDB, AD, DBpedia, Yelp-2, Yelp-5, Amazon-2, Amazon-5) [174], and the RACE reading comprehension dataset [82]. In addition, to see how F-TFM performs when token-level prediction is needed, we consider the SQuAD question answering task which requires the model to select a token span from the context paragraph as the answer.

Finally, for all models implemented in this work including Transformer baselines in the base-scale comparison section 5.3.1, we always use the relative positional attention parameterization proposed by Transformer-XL [34].

5.3.1 Base-scale Results

Firstly, we evaluate how F-TFM performs compared to the standard Transformer under similar amount of computation (i.e., FLOPs). For this purpose, we consider three commonly used model sizes for the standard Transformer, namely large (L24H1024), base (L12H768) and small (L6H768). Then, for each Transformer baseline, we construct F-TFMs of different block layouts and parameters, while ensuring the F-TFMs always have fewer or similar FLOPs. Based on the MLM pretraining objective, the results on GLUE benchmark and text classification are presented in Table 5.1, where we also include the *relative* FLOPs and #Params. Here, we can make a few key observations:

- Given similar or fewer FLOPs, by trading sequential resolution for more layers, the F-TFM outperforms the standard Transformer in most tasks except STS-B, especially for smaller models.
- When we only compress the sequence length without increasing the depth (and #Params), F-TFM could suffer from some performance loss in certain settings on the GLUE datasets. However, as the model size increases, such performance gaps become smaller or even disappear.
- In addition, we find partial parameter-sharing often harms the performance. Therefore, the practical trade-off should be made according to the actual task and computation device.

To further test generality of F-TFM, we additionally consider ELECTRA for pretraining. The results are summarized in Table 5.2. Overall, we see a similar trend, though the gain is slightly smaller on the GLUE benchmark. This could be attributed to reusing two key hyper-parameters (discriminator loss coefficient and generator size multiplier) tuned for Transformer to train F-TFMs without any adjustment at all.

5.3.2 Large-scale Results

Given the encouraging results of F-TFM at base-scale, we next consider training F-TFM under the large-scale setting and compare it with previous models pretrained in similar settings. Due to

Model size	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE-AVG
L24H1024	63.2	94.8	91.8/88.5	91.1	88.7/91.7	88.7	94.0	80.5	86.6
B10-10-10	64.8	95.0	92.5/89.5	90.7	88.6/91.5	88.9	94.0	81.5	**87.0**
B8-8-8	63.5	94.7	92.2/89.0	90.7	88.9/91.7	88.8	93.6	81.2	86.7
L12H768	60.5	93.0	92.2/89.0	89.4	88.1/91.2	86.0	92.2	73.6	84.4
B6-6-6	62.5	94.0	92.2/89.0	89.5	88.4/91.4	87.0	92.7	76.5	**85.3**
B6-3x2-3x2	60.5	93.6	92.4/89.2	89.4	88.2/91.3	86.4	92.5	75.0	84.7
B4-4-4	59.1	92.7	91.8/88.7	89.1	88.2/91.3	85.5	92.0	73.2	83.9
L6H768	55.2	91.5	91.1/87.8	88.1	87.2/90.6	82.7	90.0	64.6	81.3
B3-4-4	59.0	92.8	91.8/88.5	88.5	87.8/90.9	84.8	91.8	73.2	**83.7**

Model size	IMDB	AG	DBpedia	Yelp2	Yelp5	Amazon2	Amazon5	FLOPs	#Params
L24H1024	4.440	**4.987**	0.646	1.758	28.73	2.409	32.78	1.00x	1.00x
B10-10-10	**4.404**	5.026	**0.617**	**1.734**	**28.52**	**2.400**	**32.65**	0.73x	1.22x
B8-8-8	4.552	5.079	0.664	1.713	28.84	2.438	32.87	0.58x	1.00x
L12H768	5.328	5.184	0.663	2.013	29.35	2.571	33.14	1.00x	1.00x
B6-6-6	**4.908**	**5.079**	0.654	**1.939**	**29.03**	**2.518**	**32.91**	0.88x	1.39x
B6-3x2-3x2	5.144	5.342	**0.649**	1.892	**29.03**	2.570	33.01	0.88x	1.00x
B4-4-4	5.348	5.250	0.670	1.979	29.37	2.596	33.16	0.58x	1.00x
L6H768	6.252	5.421	0.697	2.203	30.33	2.801	33.69	1.00x	1.00x
B3-4-4	**5.520**	**5.342**	**0.670**	**2.042**	**29.51**	**2.603**	**33.16**	1.00x	1.53x

Table 5.1: MLM pretraining results at the base scale: GLUE dev *performances (the higher the better)* in the upper panel and text classification *error rates (the lower the better)* in the lower panel . The FLOPs and #Params both refer to the finetuning setting with only the encoder. The FLOPs is a rough estimation assuming linear complexity w.r.t. the sequence length. The #Params is exact including the embedding matrix.

the slightly better performance of ELECTRA over MLM, we will use the ELECTRA objective for all large-scale experiments. Given the pretrained F-TFM of different sizes, we first compare the finetuning performance on the GLUE benchmark in Table 5.3. Similar to the base-scale results, with fewer or comparable FLOPs, F-TFM outperforms the corresponding baselines in the majority of tasks, suggesting the good scalability of F-TFM.

We also test the models on the 7 text classification tasks. Table 5.4 includes the performance comparison on 7 text classification tasks under the large-scale training setting. Similar to the GLUE benchmark results, compared with the previous result based on Transformer, with fewer FLOPs, the proposed F-TFM achieves comparable results.

Next, we consider the RACE dataset, which is quite different from the GLUE benchmark. At the core, RACE is a multiple-choice reading comprehension task requiring complex reasoning, which though, can be formulated as classifying the correct choice. Also, paragraphs in RACE are much longer. To F-TFM, this presents both a challenge, as it requires detailed reasoning, and an opportunity to compress long paragraph. As we can see in Table 5.5, F-TFM achieves better

Model size	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE-AVG
L24H1024	66.5	94.3	92.8/90.0	91.5	89.6/92.2	89.4	94.1	84.5	87.8
B10-10-10	68.6	95.0	93.0/90.0	91.0	88.9/91.7	89.1	93.6	84.5	**87.9**
B8-8-8	66.6	94.8	92.6/89.7	90.7	88.8/91.7	89.0	93.6	82.1	87.3
L12H768	64.3	93.1	92.1/89.2	90.8	88.7/91.7	86.4	92.1	75.4	85.4
B6-6-6	64.3	94.2	92.8/89.7	90.1	88.7/91.6	87.4	92.5	78.3	**86.0**
B6-3x2-3x2	63.9	94.2	93.0/90.2	89.5	88.4/91.4	87.0	92.2	77.6	85.7
B4-4-4	62.8	93.6	92.5/89.2	89.2	88.4/91.3	86.0	91.6	74.3	84.8
L6H768	62.1	91.1	90.8/86.8	88.9	88.2/91.3	83.9	89.7	66.7	82.6
B3-4-4	59.0	93.1	90.8/87.5	88.7	88.1/91.0	85.8	91.1	72.5	**83.6**

Model size	IMDB	AG	DBpedia	Yelp2	Yelp5	Amazon2	Amazon5	FLOPs	#Params
L24H1024	4.724	**5.053**	0.653	1.874	28.84	2.425	32.85	1.00x	1.00x
B10-10-10	**4.324**	5.250	**0.639**	**1.789**	**28.68**	**2.419**	**32.72**	0.73x	1.22x
B8-8-8	4.364	5.408	0.651	1.729	28.76	2.447	32.85	0.58x	1.00x
L12H768	5.248	5.355	0.657	1.953	29.24	2.596	33.04	1.00x	1.00x
B6-6-6	**4.792**	**5.237**	**0.650**	**1.850**	**28.73**	**2.499**	**32.79**	0.88x	1.39x
B6-3x2-3x2	4.924	5.342	0.671	1.913	29.00	2.523	32.85	0.88x	1.00x
B4-4-4	5.152	5.382	0.659	2.032	29.33	2.566	33.03	0.58x	1.00x
L6H768	6.220	5.395	0.674	2.287	30.16	2.759	33.57	1.00x	1.00x
B3-4-4	**5.396**	**5.342**	**0.653**	**2.000**	**29.60**	**2.591**	**33.09**	1.00x	1.53x

Table 5.2: ELECTRA pretraining results at the base scale.

performances compared to all previous models. In particular, within the base model group, the gain is very significant. It shows that F-TFM can also excel for sequence-level task that involves long text and reasoning.

Finally, although F-TFM is mainly designed for tasks that only require a sequence-level representation, it is possible to apply F-TFM to token-level tasks by additionally finetuning the decoder. To test this ability, we finetune F-TFM on the SQuAD datasets and compare it with previous models in Table 5.6. While F-TFM outperforms previous models in the base group by a large margin, in the large model group, the F-TFM with about 83% FLOPs (B10-10-10) still falls behind the standard Transformer that always maintains a full-length token-level representations. This suggests sequential compression could harm the performance when detailed token-level information is critical. On the other hand, compared to the results on SQuAD1.1, F-TFMs perform relatively better on SQuAD2.0, which additionally requires the model to make a sequence-level prediction on whether the question is answerable. This again shows the general effectiveness of the F-TFM in sequence-level tasks.

Model	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI	AVG
Dev set results (single model)										
$\text{ROBERTA}_{\text{Large}}$ [93]	68.0	96.4	-/90.9	92.4	-/92.2	90.2	94.7	86.6	-	88.9
$\text{XLNet}_{\text{Large}}$ [171]	69.0	97.0	-/90.8	92.5	-/92.3	90.8	94.9	85.9	-	89.2
$\text{ELECTRA}_{\text{Large}}$ [28]	69.1	96.9	-/90.8	92.6	-/92.4	90.9	95.0	88.0	-	89.5
B10-10-10H1024	72.4	96.8	93.5/90.9	92.1	89.8/92.4	91.1/-	95.1	89.5	-	**90.0**
B8-8-8H1024	71.3	96.8	93.1/90.7	91.7	89.8/92.4	90.8/-	94.7	89.2	-	89.7
$\text{ROBERTA}_{\text{Base}}$ [93]	63.6	94.8	-/90.2	91.2	-/91.9	87.6/-	92.8	78.7	-	86.4
$\text{MPNet}_{\text{Base}}$ [143]	65.0	95.4	-/91.5	90.9	-/91.9	88.5/-	93.3	85.2	-	87.7
B6-6-6H768	70.1	96.3	93.2/90.4	91.1	89.2/92.0	89.7/-	93.7	83.4	-	**88.3**
B6-3x2-3x2H768	68.5	95.6	92.5/89.5	91.0	89.3/92.0	89.1/-	93.0	83.4	-	87.8
B4-4-4H768	68.2	95.0	92.8/90.2	90.3	89.0/91.8	88.6/-	92.6	79.1	-	87.0
Leaderboard test set results (single task & single model)										
$\text{ELECTRA}_{\text{Large}}$ [28]	68.1	96.7	89.2/92.0	92.1/91.7	74.8/90.4	90.7/90.2	95.5	86.1	65.1	85.2
B10-10-10H1024	68.9	97.2	89.4/92.1	91.6/91.3	74.3/90.2	90.9/90.9	95.5	86.5	65.1	**85.4**
B8-8-8H1024	68.3	96.9	89.2/92.0	91.5/91.1	73.8/90.1	90.7/90.7	95.1	85.3	65.1	85.0
$\text{ELECTRA}_{\text{Base}}$ [28]	64.6	96.0	88.1/91.2	91.0/90.2	73.2/89.5	88.5/88.0	93.1	75.2	65.1	82.7
B6-6-6H768	68.3	96.5	89.1/91.9	90.6/89.9	73.3/89.9	89.7/89.4	94.0	80.4	65.1	**84.0**
B6-3x2-3x2H768	65.9	96.0	87.8/91.0	90.0/89.6	73.3/89.8	88.9/88.7	93.8	79.9	65.1	83.4
Leaderboard test set results (multi-task & ensemble)										
$\text{ROBERTA}_{\text{Large}}$ [93]	67.8	96.7	89.8/92.3	92.2/91.9	74.3/90.2	90.8/90.2	95.4	88.2	89.0	88.1
$\text{ELECTRA}_{\text{Large}}$ [28]	71.7	97.1	90.7/93.1	92.9/92.5	75.6/90.8	91.3/90.8	95.8	89.8	91.8	89.4
B10-10-10H1024	70.5	97.5	91.2/93.4	92.6/92.3	75.4/90.7	91.4/91.1	95.8	90.0	94.5	**89.7**

Table 5.3: Comparison with previous methods on the GLUE benchmark under large-scale pre-training.

Model	IMDB	AG	DBpedia	Yelp-2	Yelp-5	Amazon-2	Amazon-5
BERT-Large	4.51	-	0.64	1.89	29.32	2.63	34.17
ROBERTA-Large	3.50	-	-	-	-	-	-
XLNet-Large	**3.20**	**4.45**	0.64	1.37	**27.05**	2.11	31.67
B10-10-10H1024	3.36	4.66	**0.60**	**1.33**	27.14	**2.10**	**31.64**
B8-8-8H1024	3.42	4.96	0.63	1.39	27.20	2.14	31.74
MPNet	4.40	-	-	-	-	-	-
B6-6-6H768	**3.72**	**5.00**	**0.64**	**1.50**	**27.73**	**2.27**	**32.11**
B6-3x2-3x2H768	3.82	5.12	0.64	1.58	27.96	2.32	32.23
B4-4-4H768	4.12	5.09	0.67	1.70	28.40	2.35	32.46

Table 5.4: Text classification performance comparison under the large-scale pretraining.

Model	RACE		
	Total	High	Middle
$\text{ROBERTA}_{\text{Large}}$ [93]	83.2	81.3	86.5
$\text{XLNet}_{\text{Large}}$ [171]	85.4	84.0	88.6
B10-10-10	**85.7**	**84.4**	**88.8**
B8-8-8	85.2	83.9	88.4
$\text{ALBERT}_{\text{Base}}$ [84]	66.0	-	-
$\text{MPNet}_{\text{Base}}$ [143]	72.0	76.3	70.3
B6-6-6	**79.7**	**78.2**	**83.4**
B6-3x2-3x2	78.8	77.5	82.0
B4-4-4	76.2	74.6	80.0

Table 5.5: RACE test performance comparison.

Model	SQuAD2.0		SQuAD1.1	
	EM	F1	EM	F1
$\text{ROBERTA}_{\text{Large}}$ [93]	86.5	89.4	88.9	94.6
$\text{ELECTRA}_{\text{Large}}$ [28]	**88.0**	**90.6**	**89.7**	**94.9**
B10-10-10	87.6	90.4	89.0	94.7
B8-8-8	87.1	89.8	88.7	94.4
$\text{ROBERTA}_{\text{Base}}$ [93]	80.5	83.7	84.6	91.5
$\text{MPNet}_{\text{Base}}$ [90]	80.5	83.3	86.8	92.5
B6-6-6	**85.1**	**87.7**	**87.4**	**93.3**
B6-3x2-3x2	84.2	87.0	87.0	93.0
B4-4-4	82.6	85.5	85.9	92.2

Table 5.6: SQuAD dev performance comparison.

ID	Layout	(FLOPs / Params)	Pool-Op	Pool-query-only	Sep `[CLS]`	Rel-Attn	GLUE-AVG
(1)	B6-6-6	(1.00x / 1.00x)	Mean	✓	✓	✓	**83.5**
(2)			Mean	✓		✓	82.9
(3)			Mean		✓	✓	83.0
(4)			Mean	✓	✓		81.4
(5)			Max	✓	✓	✓	83.4
(6)			Top-Attn	✓	✓	✓	75.8
(7)	B8-8	(1.14x / 0.91x)	Mean	✓	✓	✓	83.4
(8)	B5-5-5-5	(0.89x / 1.08x)	Mean	✓	✓	✓	82.9

Table 5.7: Ablation study of F-TFMs with different designs.

5.3.3 Ablation Study

Finally, based on the GLUE benchmark, we perform a series of ablation studies on the importance of various designs in F-TFM, including the block layout design, the type of pooling operation, the pool-query-only technique, maintaining a separate `[CLS]` vector and the usage of Transformer-XL parameterization.

- Pooling operation: Including the mean pooling we finally employ in F-TFM, we actually test two types of pooling operations.
  (1) The first type is just the strided mean/max pooling as described in section 5.2.
  (2) The second type aims to select a subset of "hub" states, which refer to those hidden vectors that are attended most in the previous S-Attn layer and hence likely to carry most critical information about the sequence. Concretely, given the attention map from the previous S-Attn layer, we reduce sum the scores along the number of head and query length dimensions to a score for each position. Then, we simply choose the top 50% of states to achieve the same compression rate. Note that, this type of pooling operation is essentially the same as the important states selection procedure in Power-BERT [47].
- Pool-query-only design

- Separating `[CLS]` in the pooling operation
- Block layout design: In our experiments, all models actually utilize a 3-block design. Here, we compare the 3-blocks design with the 2-blocks and the 4-blocks design.
- Relative attention parameterization proposed in Transformer-XL [34]. We compare this parameterization with the learned absolute position embedding as used in the BERT [36].

The ablation results are included in Table C.2. To save the computation resources, the size of model hidden states in table C.2 is set as 512. From the ablation results, we can make the following observations:

- Comparing pooling different operation ((1), (5), and (6)), we found that the performance of the mean and max pooling operation is similar. But they are significantly better than the idea of utilizing attention score (Top-Attn pooling) to select the "hub" states.
- Comparing (1) with (2) and (3) respectively, we see that the two special designs, i.e. "pool-query-only" and maintaining a separate non-pooled `[CLS]`, can both bring a clear improvement to the proposed model.
- Comparing (1) and (4), we find that the relative positional parameterization is key to the performance of the proposed F-TFM. We suspect that the pooling operation could destroy the positional information carried by the absolute position encoding, which is only injected to the model in the input embedding layer. As a result, the higher blocks may not have enough positional information to learn a good enough attention pattern. In comparison, the positional information is injected to each layer under the relative positional attention scheme. Therefore, to achieve good result with F-TFM based on absolute positional embedding, one may inject the absolute positional embedding into each attention layer. Actually, a contemporary application of Transformer to the detection problem in computer vision shows injecting positional embedding into each layer is important [21].
- Finally, we study the influence of block layout design in our framework. With B6-6-6 as the 3-block benchmark, we consider two other layout design with similar FLOPs and number of parameters. Specifically, we consider B8-8 for the 2-block design and B5-5-5-5 for the 4-block design. Comparing the results in (1), (7), and (8), we find that the performance of the 3-block (B6-6-6) design achieves the best performance, which is significantly better than the 4-block design and slightly better than the 2-block design. However, if we further taking the FLOPs/#Params into consideration, it is more clear that the 3-block design is superior. Therefore, in the main paper, we always use the 3-block design.

5.3.4 Training Cost Comparison

While FLOPs count offers a general idea of the model speed, it still differs from the actual running time, especially when other overhead exists. Hence, for completeness, we compare the actual pretraining and finetuning running time between F-TFM and the standard Transformer on the TPU and GPU platform. For the pretraining speed evaluation, we test F-TFM on TPU v3-16 (16 cores x 16Gb) with TensorFlow. For the finetuning speed evaluation, we test F-TFM on TPU v2-8 (8 cores x 8Gb) with TensorFlow and on Nvidia-V100 (16Gb) GPU with the PyTorch. The TensorFlow version is 2.2.0, and the PyTorch version is 1.5.0. For the GPU experiments, we use

an 8-GPU node on the Google Cloud Platform. All running speeds are reported with the FP16 optimizer. In the PyTorch implementation, we use "O2" options of AMP manager in the apex[2] package to handle the FP16 optimization. For finetuning, we consider three different sequence lengths, namely 128, 256 and 512. For pretraining, we only consider the sequence length 512. In each case, we choose the maximum possible batch size allowed by the memory size of the device(s). We measure the actual model *running time* by performing 1000 steps gradient descent with random input sequences with the fixed length.

Sequence length	128			256			512		
Metrics	Run time 1 GPU	Run time 8 GPUs	Mem	Run time 1 GPU	Run time 8 GPUs	Mem	Run time 8 GPUs	Mem	GLUE
Batch size / GPU	64			32			16		
L12H768	1.00x	1.00x	9.2G	1.00x	1.00x	11.0G	1.00x	14.3G	84.40
B6-6-6	0.97x	0.99x	9.1G	0.95x	0.97x	10.3G	0.94x	12.5G	85.37
B6-3x2-3x2	0.93x	0.93x	8.4G	0.91x	0.92x	9.5G	0.90x	11.8G	84.78
B4-4-4	0.67x	0.67x	6.6G	0.65x	0.66x	7.5G	0.64x	9.0G	83.99
Batch size / GPU	32			12			4		
L24H1024	1.00x	1.00x	14.8G	1.00x	1.00x	14.4G	1.00x	13.9G	86.62
B10-10-10	0.87x	0.92x	14.0G	0.90x	0.93x	13.0G	0.96x	12.7G	87.03
B8-8-8	0.70x	0.73x	11.6G	0.73x	0.75x	10.8G	0.78x	10.5G	86.70

Table 5.8: Running time and memory consumption comparison between F-TFMs and the standard Transformer on the GPU. In each model group, the standard Transformer (first model) is used as the benchmark for the rest of F-TFM models. Note that, given the same batch size per GPU, the memory consumption is roughly the same for 1 GPU and 8 GPUs.

Firstly, we compare the model speed in the finetuning stage. Note that the decoder is not used in this setting. Table 5.8 and 5.9 summarize the finetuning running time comparison on GPUs and TPUs, respectively.

- In the base model (L12H768) group, we observe that the speed of B6-6-6H768 is similar or faster than the base Transformer model, despite the fact that B6-6-6 is deeper, has more parameters. Moreover, B6-6-6H768 achieves better results compared with the base Transformer model. The similar conclusion applies to the B6-3x2-3x2 model, which has the same amount of parameters as the base model. The B4-4-4 model, which has the same depth and model parameters as the base model, is able to provide 30%-50% speedup without losing too much performance.
- In the large model (L24H1024) group, the conclusion is similar. The speed of the larger model B10-10-10 is almost the same as the large model, and the speed of B8-8-8 is significantly faster than the large model. In addition, when sequence length equals 512, the acceleration of F-TFM on the TPU is more obvious than the GPU.

[2] `https://github.com/NVIDIA/apex`

Sequence length	128	256	512	
Metrics	Run time on 8 TPU cores (TPUv2-8)			GLUE
Batch size / TPU core	64	32	16	
L12H768	1.00x	1.00x	1.00x	84.40
B6-6-6	0.99x	0.88x	0.81x	85.37
B6-3x2-3x2	0.97x	0.87x	0.77x	84.78
B4-4-4	0.69x	0.62x	0.55x	83.99
Batch size / TPU core	16	8	4	
L24H1024	1.00x	1.00x	1.00x	86.62
B10-10-10	0.89x	0.81x	0.73x	87.03
B8-8-8	0.66x	0.60x	0.56x	86.70

Table 5.9: Running time between F-TFMs and the standard Transformer on the TPU v2-8. In each model group, the standard Transformer (first model) is used as the benchmark for the rest of F-TFM models.

- In the both groups, all the tested F-TFM variants have smaller memory footprint compared with the standard TFM models, showing the memory efficiency of F-TFM.

Next, we compare the model speed during pretraining under the MLM objective in table 5.10, which has an additional cost due to the decoder. The results show that the proposed method can still substantially improve the pretraining speed compared to the standard Transformer, though the speed gain is slightly smaller than the finetuning stage. In summary, this study demonstrates that the proposed method is more efficient in both the finetuning and pretraining stages in modern parallel computing platforms.

Sequence Length	512	
	Running Time	FLOPs
#TPU cores / Total bsz	16 / 512	
L12H768	1.00x	1.00x
B6-6-6H768D2	0.99x	1.04x
B6-3x2-3x2H768D2	0.97x	1.04x
B4-4-4H768D2	0.79x	0.75x
#TPU cores / Total bsz	16 / 128	
L24H1024	1.00x	1.00x
B10-10-10H1024D2	0.83x	0.81x
B8-8-8H1024D2	0.71x	0.66x

Table 5.10: TPU pretraining speed comparison. The suffix "D2" means that the F-TFM model has 2 decoder layers.

Chapter 6

Additional Completed Work

In this chapter, we sketch a series of completed work to be included in this thesis. Overall, section 6.1 present a theoretical framework to analyze a fundamental problem, termed Softmax Bottleneck, in language modeling and present a simply solution to it. Then, section 6.2 and 6.3 present two works related to GANs with a focus on image data.

6.1 Breaking the Softmax Bottleneck

As a fundamental task in natural language processing, statistical language modeling has gone through significant development from traditional Ngram language models to neural language models in the last decade [12, 103, 108]. Despite the huge variety of models, as a density estimation problem, language modeling mostly relies on an auto-regressive factorization of the joint probability and then models each conditional factor using different approaches. Specifically, given a corpus of tokens $\mathbf{X} = [X_1, \ldots, X_T]$, the joint probability $P(\mathbf{X})$ factorizes as $P(\mathbf{X}) = \prod_t P(X_t \mid X_{<t}) = \prod_t P(X_t \mid C_t)$, where $C_t = X_{<t}$ is often referred to as the *context* of the conditional probability.

Based on the factorization, the most widely adapted approach [67, 97, 100, 172] employs a recurrent network to encode the context C_t into a fixed size vector h_t, which is then multiplied by the word embeddings [64, 119] using dot product to obtain the logits. The logits are consumed by the Softmax function to give a categorical probability distribution over the next token. In spite of the expressiveness of RNNs as universal approximators [135], an unclear question is whether the combination of *dot product* and *Softmax* is capable of modeling the true conditional probability, which can vary dramatically with the change of the context.

In order to analyze this problem, we consider a natural language as a finite set of pairs of a context and its conditional next-token distribution $\mathcal{L} = \{(c_1, P^*(X \mid c_1)), \cdots, (c_N, P^*(X \mid c_N))\}$, where N is the number of possible contexts. We assume $P^* > 0$ everywhere to account for errors and flexibility in natural language. Let $\{x_1, x_2, \cdots, x_M\}$ denote a set of M possible tokens in the language $\mathcal{L}$. The objective of a language model is to learn a model distribution $P_\theta(X \mid C)$ parameterized by θ to match the true data distribution $P^*(X \mid C)$.

Under the typical parameterization, a Softmax function operates on a context vector (or hidden

state) $\mathbf{h}_c$ and a word embedding $\mathbf{w}_x$ to define the categorical conditional distribution $P_\theta(x \mid c)$:

$$P_\theta(x \mid c) = \frac{\exp\left(\mathbf{h}_c^\top \mathbf{W}_x\right)}{\sum_{x'} \exp\left(\mathbf{h}_c^\top \mathbf{W}_{x'}\right)}$$

where $\mathbf{h}_c$ is a function of c, and $\mathbf{w}_x$ is a function of x. The dot product $\mathbf{h}_c^\top \mathbf{w}_x$ is called a *logit*. To help discuss the expressiveness of this Softmax-based parameterization, we define three matrices:

$$\mathbf{h}_\theta = \begin{bmatrix} \mathbf{h}_{c_1}^\top \\ \mathbf{h}_{c_2}^\top \\ \cdots \\ \mathbf{h}_{c_N}^\top \end{bmatrix}; \ \mathbf{w}_\theta = \begin{bmatrix} \mathbf{w}_{x_1}^\top \\ \mathbf{w}_{x_2}^\top \\ \cdots \\ \mathbf{w}_{x_M}^\top \end{bmatrix}; \ \mathbf{A} = \begin{bmatrix} \log P^*(x_1 \mid c_1), & \log P^*(x_2 \mid c_1) & \cdots & \log P^*(x_M \mid c_1) \\ \log P^*(x_1 \mid c_2), & \log P^*(x_2 \mid c_2) & \cdots & \log P^*(x_M \mid c_2) \\ \vdots & \vdots & \ddots & \vdots \\ \log P^*(x_1 \mid c_N), & \log P^*(x_2 \mid c_N) & \cdots & \log P^*(x_M \mid c_N) \end{bmatrix}$$

where $\mathbf{H}_\theta \in \mathbb{R}^{N \times d}$, $\mathbf{W}_\theta \in \mathbb{R}^{M \times d}$, $\mathbf{A} \in \mathbb{R}^{N \times M}$, and the rows of $\mathbf{H}_\theta$, $\mathbf{W}_\theta$, and $\mathbf{A}$ correspond to context vectors, word embeddings, and log probabilities of the *true data distribution* respectively. We further specify a set of matrices formed by adding a potentially different constant to each row of the matrix $\mathbf{A}$:

$$F(\mathbf{A}) = \{\mathbf{A} + \mathbf{\Lambda}\mathbf{J}_{N,M} \mid \mathbf{\Lambda} \text{ is an arbitray diagonal matrix of size } N \times N\}.$$

Effectively, this set of matrices correspond to the same conditional distribution. To see that, notice that for any $\mathbf{A}' \in F(\mathbf{A})$, the Softmax result of each row does not change, i.e.,

$$\text{Softmax}(\mathbf{A}'_i) = \text{Softmax}(\mathbf{A}_i), \quad \forall\, i = 1, \ldots, N.$$

Given the definitions above, for any $\mathbf{A}' \in F(A)$, it can be proved that

$$\mathbf{H}_\theta \mathbf{W}_\theta^\top = \mathbf{A}' \iff P_\theta(X \mid c) = P^*(X \mid c),\ \forall\, c \in \mathcal{L}. \tag{6.1}$$

Interestingly, the LHS of the Eqn. (6.1) is essentially a matrix factorization problem. In other words, for the standard Softmax parameterization to be able to recover the true conditional distribution, there must exist a solution θ^* to this matrix factorization problem. However, since $\mathbf{H}_\theta \in \mathbb{R}^{N \times d}$, $\mathbf{W}_\theta \in \mathbb{R}^{M \times d}$, it directly follows

$$\text{rank}(\mathbf{H}_\theta) \le d \quad \text{and} \quad \text{rank}(\mathbf{W}_\theta) \le d \implies \text{rank}(\mathbf{H}_\theta \mathbf{W}_\theta^\top) \le d.$$

In other words, the rank of the model parameterized logit matrix $\mathbf{H}_\theta \mathbf{W}_\theta^\top$ is strictly upper bounded by the hidden dimension of the word embeddings d.

Meanwhile, if can be proved that for any $\mathbf{A}_1 \neq \mathbf{A}_2 \in F(\mathbf{A})$, $|\text{rank}(\mathbf{A}_1) - \text{rank}(\mathbf{A}_2)| \le 1$. In other words, all matrices in $F(\mathbf{A})$ have similar ranks, with the maximum rank difference being 1.

Combining the two results above, we reach the following conclusion: If $d < \text{rank}(\mathbf{A}) - 1$, for any function family $\mathcal{U}$ and any model parameter θ, there exists a context c in $\mathcal{L}$ such that $P_\theta(X \mid c) \neq P^*(X \mid c)$. In other words, when the dimension d is too small, Softmax does not have the capacity to express the true data distribution.

For a natural language with very diverse context (i.e. a large N) and a large vocabulary size (i.e. a large M), we tend to believe $\mathbf{A} \in \mathbb{R}^{N \times M}$ will have a much higher rank than d, which is

only a few hundred. Given the hypothesis that natural language is high-rank, it is clear that the Softmax limits the expressiveness of the models. We refer to this phenomenon as the *Softmax Bottleneck*.

Motivated from the theoretical analysis above, we propose a high-rank language model called Mixture of Softmaxes (MoS) to alleviate the Softmax bottleneck. MoS formulates the conditional distribution as

$$P_\theta(x \mid c) = \sum_{k=1}^{K} \pi_{c,k} \frac{\exp\left(\mathbf{h}_{c,k}^\top \mathbf{w}_x\right)}{\sum_{x'} \exp\left(\mathbf{h}_{c,k}^\top \mathbf{w}_{x'}\right)}; \quad \text{s.t.} \sum_{k=1}^{K} \pi_{c,k} = 1$$

where $\pi_{c,k}$ is the *prior* or *mixture weight* of the k-th component, and $\mathbf{h}_{c,k}$ is the k-th context vector associated with context c. In other words, MoS computes K Softmax distributions and uses a weighted average of them as the next-token probability distribution.

MoS is very simple and easy to implement. More importantly, MoS is theoretically more (or at least equally) expressive compared to Softmax given the same dimension d. To see this, first notice that MoS with $K = 1$ reduces to Softmax. Moreover, MoS effectively approximates $\mathbf{A}$ by

$$\hat{\mathbf{A}}_{\text{MoS}} = \log \sum_{k=1}^{K} \mathbf{\Pi}_k \exp(\mathbf{H}_{\theta,k} \mathbf{W}_\theta^\top)$$

where $\mathbf{\Pi}_k$ is an $(N \times N)$ diagonal matrix with elements being the prior $\pi_{c,k}$. Because $\hat{\mathbf{A}}_{\text{MoS}}$ is a nonlinear function (*log_sum_exp*) of the context vectors and the word embeddings, $\hat{\mathbf{A}}_{\text{MoS}}$ can be arbitrarily high-rank. As a result, MoS does not suffer from the rank limitation, compared to Softmax.

We evaluate the proposed MoS on a wide range of language modeling benchmark datasets.

- Firstly, Table A.1 and Table A.2 show the results on the relatively smaller datasets, namely Penn Treebank (PTB) and Wikitext-2 (WT2). With a comparable number of parameters, MoS outperforms all baselines with or without dynamic evaluation, and substantially improves over the previous best methods by up to 3.6 points in perplexity, achieving the SOTA at the time of publication.
- Moreoever, the improvement on the large-scale dataset is even more significant. As shown in Table A.3, MoS outperforms Softmax by over 5.6 points in perplexity. It suggests the effectiveness of MoS is not limited to small datasets where many regularization techniques are used. Note that with limited computational resources, we didn't tune the hyper-parameters for MoS.

For more details and additional ablative study, please refer to the Appendix A.

Model	#Param	Validation	Test
Mikolov and Zweig [102] – RNN-LDA + KN-5 + cache	9M‡	-	92.0
Zaremba et al. [172] – LSTM	20M	86.2	82.7
Gal and Ghahramani [42] – Variational LSTM (MC)	20M	-	78.6
Kim et al. [69] – CharCNN	19M	-	78.9
Merity et al. [99] – Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. [49] – LSTM + continuous cache pointer†	-	-	72.1
Inan et al. [64] – Tied Variational LSTM + augmented loss	24M	75.7	73.2
Zilly et al. [180] – Variational RHN	23M	67.9	65.4
Zoph and Le [181] – NAS Cell	25M	-	64.0
Melis et al. [97] – 2-layer skip connection LSTM	24M	60.9	58.3
Merity et al. [100] – AWD-LSTM w/o finetune	24M	60.7	58.8
Merity et al. [100] – AWD-LSTM	24M	60.0	57.3
Ours – AWD-LSTM-MoS w/o finetune	22M	58.08	55.97
Ours – AWD-LSTM-MoS	22M	**56.54**	**54.44**
Merity et al. [100] – AWD-LSTM + continuous cache pointer†	24M	53.9	52.8
Krause et al. [78] – AWD-LSTM + dynamic evaluation†	24M	51.6	51.1
Ours – AWD-LSTM-MoS + dynamic evaluation†	22M	**48.33**	**47.69**

Table 6.1: Single model perplexity on validation and test sets on Penn Treebank. Baseline results are obtained from Merity et al. [100] and Krause et al. [78]. † indicates using dynamic evaluation.

Model	#Param	Validation	Test
Inan et al. [64] – Variational LSTM + augmented loss	28M	91.5	87.0
Grave et al. [49] – LSTM + continuous cache pointer†	-	-	68.9
Melis et al. [97] – 2-layer skip connection LSTM	24M	69.1	65.9
Merity et al. [100] – AWD-LSTM w/o finetune	33M	69.1	66.0
Merity et al. [100] – AWD-LSTM	33M	68.6	65.8
Ours – AWD-LSTM-MoS w/o finetune	35M	66.01	63.33
Ours – AWD-LSTM-MoS	35M	**63.88**	**61.45**
Merity et al. [100] – AWD-LSTM + continuous cache pointer †	33M	53.8	52.0
Krause et al. [78] – AWD-LSTM + dynamic evaluation†	33M	46.4	44.3
Ours – AWD-LSTM-MoS + dynamical evaluation†	35M	**42.41**	**40.68**

Table 6.2: Single model perplexity over WikiText-2. Baseline results are obtained from Merity et al. [100] and Krause et al. [78]. † indicates using dynamic evaluation.

Model	#Param	Train	Validation	Test
Softmax	119M	41.47	43.86	42.77
MoS	113M	**36.39**	**38.01**	**37.10**

Table 6.3: Comparison on 1B word dataset. Train perplexity is the average if the last 4,000 updates.

6.2 Enabling GANs to Perform Energy Estimation

As discussed in chapter 1, relying on the efficient minimax optimization procedure, GANs are able to generate images with significantly higher fidelity compared to other generative models. However, different from most tranditional generative models, GANs cannot yield a density estimation, making the evaluation and development of GANs challenging.

As a concrete example, it is well known that GANs often suffer from the problem of mode collapse. Specifically, when modeling data from multi-modal distributions, GANs often fail to generate samples from all modes of the distribution, and instead only focus on a small set of the modes. From the perspective of density estimation, this indicates such a generative model is actually flawed since it fails to assign a proper amount of probability mass to some real data. However, due to the implicit nature of GANs, we cannot perform density estimation to verify this problem and hence improve GANs on this end.

Faced with this fundamental weakness of GANs, drawing inspiration from energy-based models, we propose the calibrated energy-based GAN (CEGAN). The overall idea is very straightforward: instead of training the discriminator to distinguish real and fake (generated) samples using a binary classification objective, the discriminator of CEGAN outputs a real valued cost for each image, and is trained to assign a higher cost to fake samples and a lower cost to real samples. This cost can be understood as the "energy" in the Boltzmann distribution, where a higher energy corresponds to a lower probability. Conceptually, the discriminator is trained to assigned a higher probability (lower energy) to real samples and vice versa. Specifically, one can translation this idea into the following objective:

$$\min_{D} \max_{p_{\text{gen}}} \quad \mathbb{E}_{x \sim p_{\text{data}}(X)}[D(x)] - \mathbb{E}_{x \sim p_{\text{gen}}(X)}[D(x)], \tag{6.2}$$

where p_{data} is the unknown true data distribution, p_{gen} is the generator distribution and D is the discriminator that assigns each data point a real-valued cost.

Ideally, we hope the discriminator D can learn the energy function at convergence. However, the naive objective (6.2) cannot achieve this goal. In order to understand why, it is helpful to analyze the optimization dynamics near convergence in GANs first. When the generator distribution matches the data distribution, the training signal (gradient) w.r.t. the discriminator vanishes. At this point, assume the discriminator still retains density information, and views some samples as more real and others as less. This discriminator will produce a training signal (gradient) w.r.t. the generator, pushing the generator to generate samples that appear more real to the discriminator. Critically, this training signal is the sole driver of the generator's training. Hence, the generator distribution will diverge from the data distribution. In other words, as long as the discriminator retains relative density information, the generator distribution cannot stably match the data distribution. Thus, in order to keep the generator stationary as the data distribution, the discriminator must assign flat (exactly the same) density to all samples at the optimal.

From the analysis above, the fundamental difficulty is that the generator only receives a single training signal (gradient) from the discriminator, which it has to follow. To keep the generator stationary, this single training signal (gradient) must vanish, which requires a degenerate discriminator. In this work, we propose to tackle this single training signal constraint directly. Specifically, we introduce a novel adversarial learning formulation which incorporates an additional training signal to the generator, such that this additional signal can

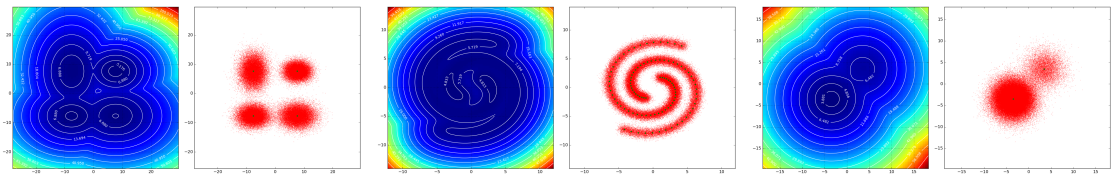


Figure 6.1: True energy functions and samples from synthetic distributions.

- balance (cancel out) the discriminator signal at the optimum, so that the generator can stay stationary even if the discriminator assigns non-flat density to samples
- cooperate with the discriminator signal to make sure the generator converges to the data distribution, and the discriminator retains the *correct* relative density information

Following this intuition, the proposed formulation can be written as the following calibrated minimax training objective,

$$\min_{D} \max_{p_{\text{gen}}} \quad \mathbb{E}_{x\sim p_{\text{data}}(X)}[D(x)] - \mathbb{E}_{x\sim p_{\text{gen}}(X)}[D(x)] + K(p_{\text{gen}}), \tag{6.3}$$

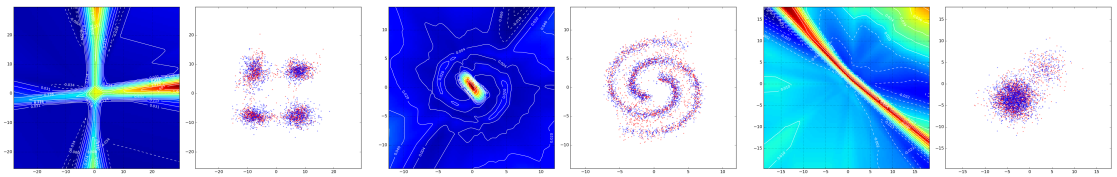
where $K(p_{\text{gen}})$ is some (functionally) differentiable, convex function of p_{gen}. Intuitively, $K(p_{\text{gen}})$ is a calibrating term introduced to provide a countervailing source of training signal for p_{gen} as we motivated above. With the objective in Eqn. (6.3), it can be proved that the discriminator D can recover the energy function by choosing the calibrating term K to be the negative entropy

$$K(p_{\text{gen}}) = -H(p_{\text{gen}}) \coloneqq \int p_{\text{gen}}(x) \log p_{\text{gen}}(x) \mathrm{d}x.$$

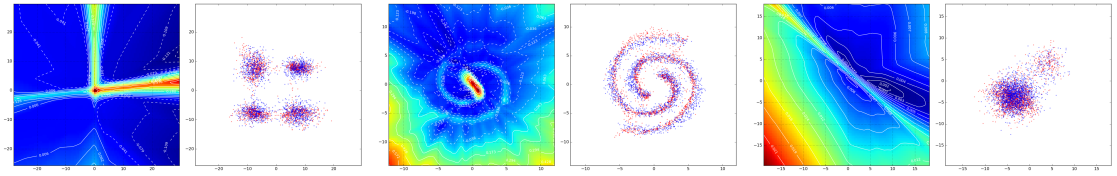
In order to show the proposed framework is able to capture the density information (in the form of energy), we perform a set of experiments on both synthetic data and real world data.

First, we consider three synthetic datasets in 2-dimensional space as illustrated in Figure B.1. Since the data lies in 2-dimensional space, we can easily visualize both the learned generator (by drawing samples) and the discriminator for direct comparison and evaluation. We compare the entropy regularized version of the proposed CEGAN with two baselines: the original GAN and the energy based GAN with no regularization (EBGAN). Experiment results are summarized in Figure 6.2. As we can see, all four models can generate perfect samples. However, for the discriminator, both GAN and EBGAN lead to degenerate solution, assigning flat energy inside the empirical data support. In comparison, CEGAN clearly captures the density information.

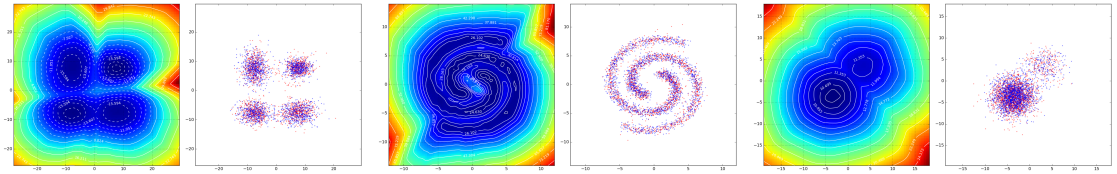
To show our framework can also generalize to high-dimension data, we show the learned energy function learns relative densities by inspecting the ranking of samples according to their assigned energies. We train on 28×28 images of a single handwritten digit from the NIST dataset. Again, we compare the ability of CEGAN with both EBGAN and GAN on ranking a set of 1,000 images, half of which are generated samples and the rest are real test images. Figures B.4 and B.5 show the top-100 and bottom-100 ranked images respectively for each model, after training them on digit 1. We also show in Figure B.7 the mean of all training samples, so we can get a sense of what is the most common style (highest density) of digit 1 in NIST. We can notice that all of the top-ranked images by CEGAN look similar to the mean sample. In addition, the lowest-ranked images are clearly different from the mean image, with either high (clockwise or



(a) Standard GAN



(b) Energy GAN without regularization (EBGAN)



(c) Entropy regularized Calibrated Energy GAN (CEGAN)

Figure 6.2: Learned energies and samples from proposed models and baseline models. Blue dots are generated samples, and red dots are real ones.

(a) CEGAN

(b) EBGAN

(c) GAN

Figure 6.3: 100 highest-ranked images out of 1000 generated and real (bounding box) samples.

(a) CEGAN

(b) EBGAN

(c) GAN

Figure 6.4: 100 lowest-ranked images out of 1000 generated and real (bounding box) samples.

Figure 6.5: mean digit

counter-clockwise) rotation degrees from the mean, or an extreme thickness level. We do not see such clear distinction in other models.

For the detailed theoretical derivation and additional empirical results, please refer to Appendix B.

6.3 Semi-supervised Learning with a "Bad" GAN

Among many potential downstream applications of generative mode, semi-supervised learning is a classic direction considered by various work in literature. Due to the relative short history of GANs, how to properly utilize GANs for semi-supervised learning remains an open problem. That said, since GANs are defined as data samplers, a natural idea is to employ GANs to perform (conditional) data augmentation, as explored by [**?**]. Interestingly, it turned out that for classification problems, there exists another more effective approach which shares the classifier and the discriminator [134]. Concretely, for a k-class classification problem, we create an additional class $k+1$, which indicates a sample is generated from the generator, i.e., "fake" data. Then, the training objective for the shared classifier/discriminator can be written as

$$\begin{aligned}\max_{\theta} \quad & \mathbb{E}_{x,y\sim\mathcal{D}_{\text{sup}}}[\log p_\theta(Y=y \mid x)] + \mathbb{E}_{x\sim\mathcal{D}_{\text{unsup}}}\Big[\log \overbrace{\sum_{i=1}^{k} p_\theta(Y=i \mid x)}^{p_\theta(Y\neq k+1|x)}\Big] \\ & + \mathbb{E}_{x\sim p_\phi(X)}[\log p_\theta(Y=k+1 \mid x)]\end{aligned} \tag{6.4}$$

where $p_\phi(X)$ is the generator distribution parameterized by ϕ, and $p_\theta(Y \mid X)$ is the classifier/discriminator distribution parameterized by θ. The three terms of the objective (6.4) have clear interpretations:

- For supervised data (1st term), simply perform the standard supervised learning;
- For an unlabeled sample (2nd term), classify it as "not fake", i.e., any of the first k classes;
- For a generated sample (3rd term), classify it as "fake", i.e., the $k+1$-th class.

Empirically, this approach achieves the state-of-the-art performance for GAN-based semi-supervised learning.

Despite the improved semi-supervised performance, the following questions still remain open. Firstly, it is not clear why training the model to distinguish real and fake samples should help the classification performance. In particular, when the generator is perfect, i.e., $p_\phi(X) = p_{\text{data}}(X)$, the generated samples are essentially the same as unlabeled real data. Then, asking the model to distinguish real and fake samples won't provide any additional information at all, not to mention improving the classification performance. Moreover, it is observed that when the generator is trained by a weaker objective such as feature matching, it generates worse images than usual but obtains a much better semi-supervised learning performance [134].

In this work, we provide both theoretical analysis and practical solutions towards addressing these questions. From a high level, our key conclusions can be summarized as follows:

- When trained by a weaker objective, the generator tends to generate *slightly* off-manifold samples that are close the true data manifold.
- Then, by additionally training the classifier to distinguishing between slightly off-manifold samples and on-manifold samples, the classifier tends to place the classification decision boundaries between the k classes in the off-manifold areas.
- Effectively, this leads to the phenomenon of low-density separation, which is a classic idea of semi-supervised learning.

To provide a more intuitive understanding, let's consider a one-dimension binary classification problem and compare the decision boundary of the classifier without or with the additional $k+1$ class objective as shown in Figure 6.6:

- When there is only supervised data (upper half), the decision boundary, i.e., where $l_{\text{red}} = l_{\text{green}}$ can be anywhere between the two supervised data points, potentially leading to the error showing in the figure;
- In contrast, when the k+1-class objective is employed (lower half), the generator will generate some "bad" samples in between the two separated manifolds. Denote the logit of the fake class with l_{fake}. Then, if the classifier is well trained based on the objective (6.4), the following three conditions should hold:
  1. On green data manifold, we have $l_{\text{fake}} \approx l_{\text{green}} \ll l_{\text{red}}$;
  2. On red data manifold, we have $l_{\text{fake}} \approx l_{\text{red}} \ll l_{\text{green}}$;
  3. In the area between the two manifolds, we have $l_{\text{red}} \approx l_{\text{green}} \ll l_{\text{fake}}$.

  Here, we use $\approx$ to denote the two logits are of similar magnitude. Essentially, in order to correctly predict fake samples, the classifier must push the logits of the red and green classes to very low values in the off-manifold area between the two data manifolds. As a result, the point where $l_{\text{red}} = l_{\text{green}}$ is most likely lying in the off-manifold (a.k.a. low-density) area. Thus, it avoids the type of classification error when only using the supervised learning objective.

In summary, the success of this semi-supervised learning objective relies on the fact the generator can generate "bad" samples that are slightly off-manifold, which leads to the name bad GAN. Note that this is quite different from the idea of using GANs for data augmentation, which instead requires "good" samples.

Following this intuition, we perform rigorous theoretical analysis and derive the conditions that are needed for this low-density separation to provably succeed. Then, motivated by the analysis, we first propose a set of techniques to (1) ensure the generator generates "bad" (off-manifold) samples and (2) encourage the generated "bad" samples to be *diverse* so that all the gaps between manifolds can be filled as much as possible. In addition, we propose another entropy regularization term to encourage a condition derived from our theoretical analysis.

Following previous work, we evaluate the proposed methods on three widely used semi-supervised classification 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

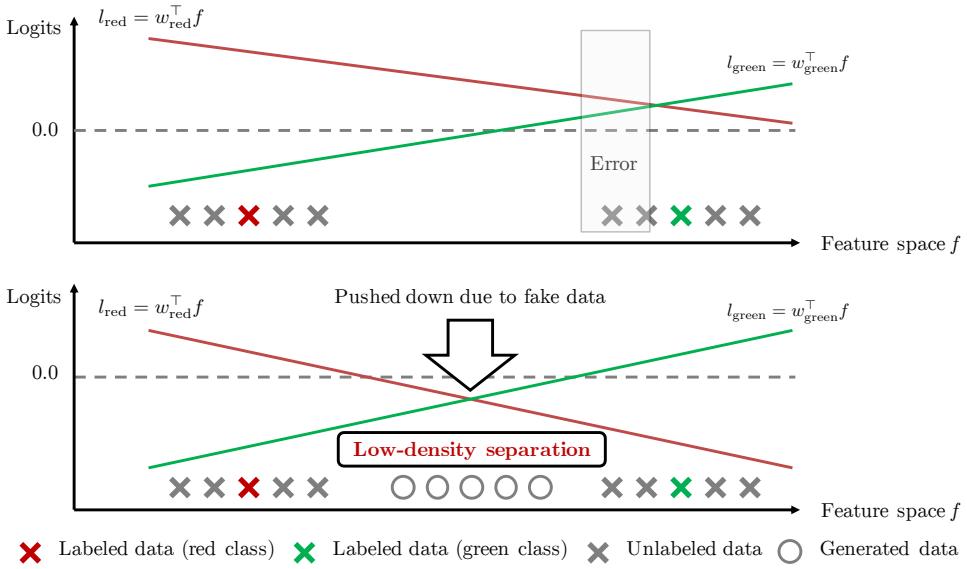


Figure 6.6: Illustration of the intuition why distinguishing real and fake examples can help the classification accuracy. l_{red} and l_{green} denote the logits of the red and green class respectively. Hence, the decision boundary is the point where the two curves of logit intersect.

Methods	MNIST (# errors)	SVHN (% errors)	CIFAR-10 (% errors)
CatGAN [144]	191 ± 10	-	19.58 ± 0.46
SDGM [94]	132 ± 7	16.61 ± 0.24	-
Ladder network [129]	106 ± 37	-	20.40 ± 0.47
ADGM [94]	96 ± 2	22.86	-
FM [134] *	93 ± 6.5	8.11 ± 1.3	18.63 ± 2.32
ALI [37]	-	7.42 ± 0.65	17.99 ± 1.62
VAT small [107] *	136	6.83	14.87
Our best model *	**79.5 ± 9.8**	**4.25 ± 0.03**	**14.41 ± 0.30**
Triple GAN [89] *‡	91 ± 58	5.77 ± 0.17	16.99 ± 0.36
Π model [83] †‡	-	5.43 ± 0.25	16.55 ± 0.29
VAT+EntMin+Large [107]†	-	4.28	13.15

Table 6.4: 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.

respectively during training, and use the standard data split for testing. We compare the the results of our best model with previous best methods on the benchmarks in Table C.1. Our proposed methods consistently improve the performance upon feature matching. At the time of publication, we achieve new SOTA results on all the datasets when only small discriminator architecture is considered. Our results are also SOTA on MNIST and SVHN among all single-model results, even when compared with methods using self-ensembling and large discriminator architectures.

For the theoretical details and empirical results, please refer to Appendix C.

Chapter 7

Conclusion

In this thesis, we investigate some directions to improve the architecture design or algorithm effectiveness of deep generative models, with significant gains in various practical applications. In summary, the contributions of this thesis can be categorized into three groups.

- The first group is centered around the relatively new topic of generative adversarial networks. Firstly, we propose a theoretical framework, i.e., calibrated energy-based GAN (CEGAN), to enable this implicit generative model to perform energy estimation at convergence, which tackles a fundamental weakness of GANs. In addition, we theoretically show how inferior generations from GANs can benefit semi-supervised classifier and connects this effectiveness to the classic low-density estimation principle in semi-supervised learning. Guided by the theoretical analysis, we further improve the semi-supervised learning algorithm with such "bad" GANs.
- The second group is concerned with language modeling, i.e. the density estimation problem for natural language text. Under the auto-regressive formulation, we identify the Softmax Bottleneck, i.e., a capacity limitation, of the most widely used output (prediction) distribution parameterized by the Softmax function. Then, we show how mixture of Softmaxes (MoS) can break this limitation and achieve substantial performance gains. In addition, to fully unleash the potential of self-attention models in capturing long-term dependency, we propose a novel model architecture Transformer-XL. With relative positional encoding, Transformer-XL allows the application of truncated back-propagation through time to training self-attention models, significantly advancing the state-of-the-art of language modeling.
- Lastly, we also make significant contribution to language pretraining (unsupervised representation learning for language). Specifically, we devise the permutation language modleing, which generalizes two previously most widely considered representation learning objectives, namely casual language modeling and the masked language modeling. As a result, the corresponding XLNet enjoys the advantages of both objectives, outperforming previously state-of-the-art pretrained models by a large margin. Moreover, to further exploit the effectiveness of language pretraining, we propose Funnel-Transformer, a more efficient self-attention architecture that compresses the hidden state sequence to a shorter length and hence reduces the computation cost. More importantly, this further allows one to trade

the sequential resolution of the hidden state sequence for a deep or wide model, leading to substantial gains under the same amount of computation as measured by the FLOPs.

Combining the recent development in deep learning with the observations of this thesis, we finally discuss several directions centered around deep generative modeling that are worth exploring in the future.

The relationship between density estimation and representation learning For deep generative modeling, this relation has always been one of the most fundamental questions. From the success of Transformer-XL, one relatively simple connection is that a model architecture that can works well in density estimation can also improve representation learning *if the model is used in similar ways*. While encouraging, this connection simply stays at the model architecture level, without tough the underlying objective or algorithm used. On one hand, the success of GPT series [16, 121, 122] clearly show how powerful standard density estimation can be in the domain of language. On the other hand, the fact that modeling conditional likelihood such as masked language modeling in BERT and partial permutation language modeling in XLNet works better for representation learning than standard density estimation in language modeling indicates that estimating the full density might not always the best choice for representation. A conceptually similar question also arises in the vision domain, where the image GPT [24] performs better than the BERT style conditional density estimation but falls behind a simple contrastive learning scheme SimCLR [25], which can be seen as estimating the density ratio. All these evidences tell us the best representation for a specific category of tasks may correspond to estimating certain statistical quantities other than the data density itself. Therefore, it is highly meaningful to establish a more formal or systematic connections between representation learning and various possible estimation statistic quantities.

More efficient algorithm to estimate various statistical quantities Following the discussion above, for different purposes, one may hope to estimate particular statistical quantities other than the density. There are often two practical complications to tackle. Firstly, as we discussed in Chapter 2, different deep generative models may or may not allow directly evaluating or approximating the desired quantity with different computational cost. Secondly, the model architecture available to us also pose additional constraints, especially on the computational efficiency. As a result, in order to estimate the desired quantity in the most *efficient* way, we need to employ a proper combination of objectives and architectures or even go beyond existing methods invent new components. This is particularly important for representation learning problems that still lie in the *under-fitting* regime and more computation can consistently lead to better performance. Therefore, we believe designing more efficient solution to estimating different desired statistical quantities would be highly rewarding.

Improving model architecture: capacity, efficiency and flexibility Finally, as one of the most critical driving forces of deep learning and deep generative modeling, we can never put too much emphasis on the model architecture. More specifically, we here emphasize three properties, namely the capacity, efficiency and flexibility. As deep generative modeling usually needs to model the full inter-dependency between high-dimension real-world data, the capacity is key to

the success to capturing the complicated high-order structures of the data. As we have seen in the last few year, the invention of Transformer has led to a revolution in the deep generative modeling for language. Compared to LSTM (the go-to model for the previous generation), Transformer has dramatically pushed down the perplexity for density estimation and enabled the success of large-scale language pretraining up to trillions of tokens. However, instead of only focusing on absolute capacity, we should care more about the capacity per computation unit (such as FLOP), i.e., the efficiency of the architecture. With representation learning starts to prevail, a more efficient model will not only allow one to better exploit the abundance of unlabeled data but also allows for practical deployment in products. At last but not the least, we believe the flexibility of architectures will become increasingly more rewarding in the future. On one hand, as more delicate objectives or algorithms are designed to tackle specific problems at discussed above, a flexible model is often required. A good example is that to implement the permutation language modeling objective used by XLNet, one can easily incorporates different factorization orders into a Transformer but not an LSTM. On the other hand, being flexible also allows one to easily migrate successful design across different problem or even data domains. Overall, we believe the pursuit of improved deep architecture will continue to be critical in the future.

Appendix A

Breaking the Softmax Bottleneck: A High-Rank RNN Language Model

A.1 Introduction

As a fundamental task in natural language processing, statistical language modeling has gone through significant development from traditional Ngram language models to neural language models in the last decade [12, 103, 108]. Despite the huge variety of models, as a density estimation problem, language modeling mostly relies on a universal auto-regressive factorization of the joint probability and then models each conditional factor using different approaches. Specifically, given a corpus of tokens $\mathbf{X} = (X_1, \ldots, X_T)$, the joint probability $P(\mathbf{X})$ factorizes as $P(\mathbf{X}) = \prod_t P(X_t \mid X_{<t}) = \prod_t P(X_t \mid C_t)$, where $C_t = X_{<t}$ is referred to as the *context* of the conditional probability hereafter.

Based on the factorization, recurrent neural networks (RNN) based language models achieve state-of-the-art results on various benchmarks [78, 97, 100]. A standard approach is to use a recurrent network to encode the context into a fixed size vector, which is then multiplied by the word embeddings [64, 119] using dot product to obtain the logits. The logits are consumed by the Softmax function to give a categorical probability distribution over the next token. In spite of the expressiveness of RNNs as universal approximators [135], an unclear question is whether the combination of dot product and Softmax is capable of modeling the conditional probability, which can vary dramatically with the change of the context.

In this work, we study the expressiveness of the aforementioned Softmax-based recurrent language models from a perspective of matrix factorization. We show that learning a Softmax-based recurrent language model with the standard formulation is essentially equivalent to solving a matrix factorization problem. More importantly, due to the fact that natural language is highly context-dependent, the matrix to be factorized can be high-rank. This further implies that standard Softmax-based language models with distributed (output) word embeddings do not have enough capacity to model natural language. We call this the *Softmax bottleneck*.

We propose a simple and effective method to address the Softmax bottleneck. Specifically, we introduce discrete latent variables into a recurrent language model, and formulate the next-token probability distribution as a *Mixture of Softmaxes* (MoS). Mixture of Softmaxes is more expressive

than Softmax and other surrogates considered in prior work. Moreover, we show that MoS learns matrices that have much larger normalized singular values and thus much higher rank than Softmax and other baselines on real-world datasets.

We evaluate our proposed approach on standard language modeling benchmarks. MoS substantially improves over the current state-of-the-art results on benchmarks, by up to 3.6 points in terms of perplexity, reaching perplexities 47.69 on Penn Treebank and 40.68 on WikiText-2. We further apply MoS to a dialog dataset and show improved performance over Softmax and other baselines.

Our contribution is two-fold. First, we identify the Softmax bottleneck by formulating language modeling as a matrix factorization problem. Second, we propose a simple and effective method that substantially improves over the current state-of-the-art results.

A.2 Language Modeling as Matrix Factorization

As discussed in Section A.1, with the autoregressive factorization, language modeling can be reduced to modeling the conditional distribution of the next token x given the context c. Though one might argue that a natural language allows an infinite number of contexts due to its compositionality [118], we proceed with our analysis by considering a finite set of possible contexts. The unboundedness of natural language does not affect our conclusions, which will be discussed later.

We consider a natural language as a finite set of pairs of a context and its conditional next-token distribution[1] $\mathcal{L} = \{(c_1, P^*(X|c_1)), \cdots, (c_N, P^*(X|c_N))\}$, where N is the number of possible contexts. We assume $P^* > 0$ everywhere to account for errors and flexibility in natural language. Let $\{x_1, x_2, \cdots, x_M\}$ denote a set of M possible tokens in the language $\mathcal{L}$. The objective of a language model is to learn a model distribution $P_\theta(X|C)$ parameterized by θ to match the true data distribution $P^*(X|C)$.

In this work, we study the expressiveness of the parametric model class $P_\theta(X|C)$. In other words, we are asking the following question: given a natural language $\mathcal{L}$, does there exist a parameter θ such that $P_\theta(X|c) = P^*(X|c)$ for all c in $\mathcal{L}$?

We start by looking at a Softmax-based model class since it is widely used.

A.2.1 Softmax

The majority of parametric language models use a Softmax function operating on a context vector (or hidden state) $\mathbf{h}_c$ and a word embedding $\mathbf{w}_x$ to define the conditional distribution $P_\theta(x|c)$. More specifically, the model distribution is usually written as

$$P_\theta(x|c) = \frac{\exp \mathbf{h}_c^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_c^\top \mathbf{w}_{x'}} \tag{A.1}$$

where $\mathbf{h}_c$ is a function of c, and $\mathbf{w}_x$ is a function of x. Both functions are parameterized by θ. Both the context vector $\mathbf{h}_c$ and the word embedding $\mathbf{w}_x$ have the same dimension d. The dot product $\mathbf{h}_c^\top \mathbf{w}_x$ is called a *logit*.

[1] We use capital letters for variables and small letters for constants.

To help discuss the expressiveness of Softmax, we define three matrices:

$$\mathbf{H}_\theta = \begin{bmatrix} \mathbf{h}_{c_1}^\top \\ \mathbf{h}_{c_2}^\top \\ \cdots \\ \mathbf{h}_{c_N}^\top \end{bmatrix}; \quad \mathbf{W}_\theta = \begin{bmatrix} \mathbf{w}_{x_1}^\top \\ \mathbf{w}_{x_2}^\top \\ \cdots \\ \mathbf{w}_{x_M}^\top \end{bmatrix}; \quad \mathbf{A} = \begin{bmatrix} \log P^*(x_1|c_1), & \log P^*(x_2|c_1) & \cdots & \log P^*(x_M|c_1) \\ \log P^*(x_1|c_2), & \log P^*(x_2|c_2) & \cdots & \log P^*(x_M|c_2) \\ \vdots & \vdots & \ddots & \vdots \\ \log P^*(x_1|c_N), & \log P^*(x_2|c_N) & \cdots & \log P^*(x_M|c_N) \end{bmatrix}$$

where $\mathbf{H}_\theta \in \mathbb{R}^{N\times d}$, $\mathbf{W}_\theta \in \mathbb{R}^{M\times d}$, $\mathbf{A} \in \mathbb{R}^{N\times M}$, and the rows of $\mathbf{H}_\theta$, $\mathbf{W}_\theta$, and $\mathbf{A}$ correspond to context vectors, word embeddings, and log probabilities of the true data distribution respectively. We use the subscript θ because $(\mathbf{H}_\theta, \mathbf{W}_\theta)$ is effectively a function indexed by the parameter θ, from the joint function family $\mathcal{U}$. Concretely, $\mathbf{H}_\theta$ is implemented as deep neural networks, such as a recurrent network, while $\mathbf{W}_\theta$ is instantiated as an embedding lookup.

We further specify a set of matrices formed by applying *row-wise shift* to $\mathbf{A}$

$$F(\mathbf{A}) = \{\mathbf{A} + \mathbf{\Lambda}\mathbf{J}_{N,M} | \mathbf{\Lambda} \text{ is diagonal and } \mathbf{\Lambda} \in \mathbb{R}^{N\times N}\},$$

where $\mathbf{J}_{N,M}$ is an all-ones matrix with size $N \times M$. Essentially, the row-wise shift operation adds an arbitrary real number to each row of $\mathbf{A}$. Thus, $F(\mathbf{A})$ is an infinite set. Notably, the set $F(\mathbf{A})$ has two important properties (see Section A.6 for the proof), which are key to our analysis.

Property 1. *For any matrix $\mathbf{A}'$, $\mathbf{A}' \in F(\mathbf{A})$ if and only if Softmax$(\mathbf{A}') = P^*$. In other words, $F(\mathbf{A})$ defines the set of **all** possible logits that correspond to the true data distribution.*

Property 2. *For any $\mathbf{A}_1 \neq \mathbf{A}_2 \in F(\mathbf{A})$, $|rank(\mathbf{A}_1) - rank(\mathbf{A}_2)| \leq 1$. In other words, all matrices in $F(\mathbf{A})$ have similar ranks, with the maximum rank difference being 1.*

Based on the Property 1 of $F(\mathbf{A})$, we immediately have the following Lemma.

Lemma 1. *Given a model parameter θ, $\mathbf{H}_\theta\mathbf{W}_\theta^\top \in F(\mathbf{A})$ if and only if $P_\theta(X|c) = P^*(X|c)$ for all c in $\mathcal{L}$.*

Now the expressiveness question becomes: does there exist a parameter θ and $\mathbf{A}' \in F(\mathbf{A})$ such that

$$\mathbf{H}_\theta\mathbf{W}_\theta^\top = \mathbf{A}'.$$

This is essentially a matrix factorization problem. We want the model to learn matrices $\mathbf{H}_\theta$ and $\mathbf{W}_\theta$ that are able to factorize some matrix $\mathbf{A}' \in F(\mathbf{A})$. First, note that for a valid factorization to exist, the rank of $\mathbf{H}_\theta\mathbf{W}_\theta^\top$ has to be at least as large as the rank of $\mathbf{A}'$. Further, since $\mathbf{H}_\theta \in \mathbb{R}^{N\times d}$ and $\mathbf{W}_\theta \in \mathbb{R}^{M\times d}$, the rank of $\mathbf{H}_\theta\mathbf{W}_\theta^\top$ is strictly upper bounded by the embedding size d. As a result, if $d \geq \text{rank}(\mathbf{A}')$, a universal approximator can theoretically recover $\mathbf{A}'$. However, if $d < \text{rank}(\mathbf{A}')$, no matter how expressive the function family $\mathcal{U}$ is, no $(\mathbf{H}_\theta, \mathbf{W}_\theta)$ can even theoretically recover $\mathbf{A}'$. We summarize the reasoning above as follows (see Section A.6 for the proof).

Proposition 1. *Given that the function family $\mathcal{U}$ is a universal approximator, there exists a parameter θ such that $P_\theta(X|c) = P^*(X|c)$ for all c in $\mathcal{L}$ if and only if $d \geq \min_{\mathbf{A}'\in F(\mathbf{A})} rank(\mathbf{A}')$.*

Combining Proposition 1 with the Property 2 of $F(\mathbf{A})$, we are now able to state the *Softmax Bottleneck* problem formally.

Corollary 1. ***(Softmax Bottleneck)** If $d < rank(\mathbf{A}) - 1$, for any function family $\mathcal{U}$ and any model parameter θ, there exists a context c in $\mathcal{L}$ such that $P_\theta(X|c) \neq P^*(X|c)$.*

The above corollary indicates that when the dimension d is too small, Softmax does not have the capacity to express the true data distribution. Clearly, this conclusion is not restricted to a finite language $\mathcal{L}$. When $\mathcal{L}$ is infinite, one can always take a finite subset and the Softmax bottleneck still exists. Next, we discuss why the Softmax bottleneck is an issue by presenting our hypothesis that $\mathbf{A}$ is high-rank for natural language.

A.2.2 Hypothesis: Natural Language is High-Rank

We hypothesize that for a natural language $\mathcal{L}$, the log probability matrix $\mathbf{A}$ is a high-rank matrix. It is difficult (if possible) to rigorously prove this hypothesis since we do not have access to the true data distribution of a natural language. However, it is suggested by the following intuitive reasoning and empirical observations:

- Natural language is highly context-dependent [102]. For example, the token "north" is likely to be followed by "korea" or "korean" in a news article on international politics, which however is unlikely in a textbook on U.S. domestic history. We hypothesize that such subtle context dependency should result in a high-rank matrix $\mathbf{A}$.
- If $\mathbf{A}$ is low-rank, it means humans only need a limited number (e.g. a few hundred) of bases, and all semantic meanings can be created by (potentially) negating and (weighted) averaging these bases. However, it is hard to find a natural concept in linguistics and cognitive science that corresponds to such bases, which questions the existence of such bases. For example, semantic meanings might not be those bases since a few hundred meanings may not be enough to cover everyday meanings, not to mention niche meanings in specialized domains.
- Empirically, our high-rank language model outperforms conventional low-rank language models on several benchmarks, as shown in Section A.3. We also provide evidences in Section A.3.3 to support our hypothesis that learning a high-rank language model is important.

Given the hypothesis that natural language is high-rank, it is clear that the Softmax bottleneck limits the expressiveness of the models. In practice, the embedding dimension d is usually set at the scale of 10^2, while the rank of $\mathbf{A}$ can possibly be as high as M (at the scale of 10^5), which is orders of magnitude larger than d. Softmax is effectively learning a low-rank approximation to $\mathbf{A}$, and our experiments suggest that such approximation loses the ability to model context dependency, both qualitatively and quantitatively (Cf. Section A.3).

A.2.3 Easy Fixes?

Identifying the Softmax bottleneck immediately suggests some possible "easy fixes". First, as considered by a lot of prior work, one can employ a non-parametric model, namely an Ngram model [73]. Ngram models are not constrained by any parametric forms so it can universally approximate any natural language, given enough parameters. Second, it is possible to increase the dimension d (e.g., to match M) so that the model can express a high-rank matrix $\mathbf{A}$.

However, these two methods increase the number of parameters dramatically, compared to using a low-dimensional Softmax. More specifically, an Ngram needs $(N \times M)$ parameters in order to express $\mathbf{A}$, where N is potentially unbounded. Similarly, a high-dimensional Softmax requires $(M \times M)$ parameters for the word embeddings. Increasing the number of model

parameters easily leads to overfitting. In past work, [73] used back-off to alleviate overfitting. Moreover, as deep learning models were tuned by extensive hyper-parameter search, increasing the dimension d beyond several hundred is not helpful[2] [78, 97, 100].

Clearly there is a tradeoff between expressiveness and generalization on language modeling. Naively increasing the expressiveness hurts generalization. Below, we introduce an alternative approach that increases the expressiveness without exploding the parametric space.

A.2.4 Mixture of Softmaxes: A High-Rank Language Model

We propose a high-rank language model called Mixture of Softmaxes (MoS) to alleviate the Softmax bottleneck issue. MoS formulates the conditional distribution as

$$P_\theta(x|c) = \sum_{k=1}^{K} \pi_{c,k} \frac{\exp \mathbf{h}_{c,k}^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_{c,k}^\top \mathbf{w}_{x'}}; \quad \text{s.t. } \sum_{k=1}^{K} \pi_{c,k} = 1$$

where $\pi_{c,k}$ is the *prior* or *mixture weight* of the k-th component, and $\mathbf{h}_{c,k}$ is the k-th context vector associated with context c. In other words, MoS computes K Softmax distributions and uses a weighted average of them as the next-token probability distribution. Similar to prior work on recurrent language modeling [78, 97, 100], we first apply a stack of recurrent layers on top of $\mathbf{X}$ to obtain a sequence of hidden states $(\mathbf{g}_1, \cdots, \mathbf{g}_T)$. The prior and the context vector for context c_t are parameterized as $\pi_{c_t,k} = \frac{\exp \mathbf{w}_{\pi,k}^\top \mathbf{g}_t}{\sum_{k'=1}^{K} \exp \mathbf{w}_{\pi,k'}^\top \mathbf{g}_t}$ and $\mathbf{h}_{c_t,k} = \tanh(\mathbf{W}_{h,k}\mathbf{g}_t)$ where $\mathbf{w}_{\pi,k}$ and $\mathbf{W}_{h,k}$ are model parameters.

Our method is simple and easy to implement, and has the following advantages:

- *Improved expressiveness* (compared to Softmax). MoS is theoretically more (or at least equally) expressive compared to Softmax given the same dimension d. This can be seen by the fact that MoS with $K = 1$ is reduced to Softmax. More importantly, MoS effectively approximates $\mathbf{A}$ by

  $$\hat{\mathbf{A}}_{\text{MoS}} = \log \sum_{k=1}^{K} \mathbf{\Pi}_k \exp(\mathbf{H}_{\theta,k} \mathbf{W}_\theta^\top)$$

  where $\mathbf{\Pi}_k$ is an $(N \times N)$ diagonal matrix with elements being the prior $\pi_{c,k}$. Because $\hat{\mathbf{A}}_{\text{MoS}}$ is a nonlinear function (*log_sum_exp*) of the context vectors and the word embeddings, $\hat{\mathbf{A}}_{\text{MoS}}$ can be arbitrarily high-rank. As a result, MoS does not suffer from the rank limitation, compared to Softmax.

- *Improved generalization* (compared to Ngram). Ngram models and high-dimensional Softmax (Cf. Section A.2.3) improve the expressiveness but do not generalize well. In contrast, MoS does not have a generalization issue due to the following reasons. First, MoS defines the following generative process: a discrete latent variable k is first sampled from $\{1, \cdots, K\}$, and then the next token is sampled based on the k-th Softmax component. By doing so we introduce an inductive bias that the next token is generated based on a latent discrete decision (e.g., a topic), which is often safe in language modeling [13]. Second, since $\hat{\mathbf{A}}_{\text{MoS}}$ is defined by a nonlinear function and not restricted by the rank bottleneck, in practice it is possible to

[2]This is also confirmed by our preliminary experiments.

reduce d to compensate for the increase of model parameters brought by the mixture structure. As a result, MoS has a similar model size compared to Softmax and thus is not prone to overfitting.

A.2.5 Mixture of Contexts: A Low-Rank Baseline

Another possible approach is to directly mix the context vectors (or logits) before taking the Softmax, rather than mixing the probabilities afterwards as in MoS. Specifically, the conditional distribution is parameterized as

$$P_\theta(x|c) = \frac{\exp\left(\sum_{k=1}^{K} \pi_{c,k}\mathbf{h}_{c,k}\right)^\top \mathbf{w}_x}{\sum_{x'} \exp\left(\sum_{k=1}^{K} \pi_{c,k}\mathbf{h}_{c,k}\right)^\top \mathbf{w}_{x'}} = \frac{\exp\left(\sum_{k=1}^{K} \pi_{c,k}\mathbf{h}_{c,k}^\top \mathbf{w}_x\right)}{\sum_{x'} \exp\left(\sum_{k=1}^{K} \pi_{c,k}\mathbf{h}_{c,k}^\top \mathbf{w}_{x'}\right)}, \tag{A.2}$$

where $\mathbf{h}_{c,k}$ and $\pi_{c,k}$ share the same parameterization as in MoS. Despite its superficial similarity to MoS, this model, which we refer to as mixture of contexts (MoC), actually suffers from the same rank limitation problem as Softmax. This can be easily seen by defining $\mathbf{h'}_c = \sum_{k=1}^{K} \pi_{c,k}\mathbf{h}_{c,k}$, which turns the MoC parameterization (A.2) into $P_\theta(x|c) = \frac{\exp \mathbf{h'}_c^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h'}_c^\top \mathbf{w}_{x'}}$. Note that this is equivalent to the Softmax parameterization (A.1). Thus, performing mixture in the feature space can only make the function family $\mathcal{U}$ more expressive, but does not change the fact that the rank of $\mathbf{H}_\theta \mathbf{W}_\theta^\top$ is upper bounded by the embedding dimension d. In our experiments, we implement MoC as a baseline and compare it experimentally to MoS.

A.3 Experiments

A.3.1 Main Results

We conduct a series of experiments with the following settings:

- Following previous work [78, 97, 100], we evaluate the proposed MoS model on two widely used language modeling datasets, namely Penn Treebank (PTB) [103] and WikiText-2 (WT2) [99] based on perplexity. For fair comparison, we closely follow the regularization and optimization techniques introduced by Merity et al. [100]. We heuristically and manually search hyper-parameters for MoS based on the validation performance while limiting the model size (see Section A.7.1 for our hyper-parameters).
- To investigate whether the effectiveness of MoS can be extended to even larger datasets, we conduct an additional language modeling experiment on the 1B Word dataset [22]. Specifically, we lower-case the text and choose the top 100K tokens as the vocabulary. A standard neural language model with 2 layers of LSTMs followed by a Softmax output layer is used as the baseline. Again, the network size of MoS is adjusted to ensure a comparable number of parameters. Notably, dropout was not used, since we found it not helpful to either model (see Section A.7.2 for more details).
- To show that the MoS is a generic structure that can be used to model other context-dependent distributions, we additionally conduct experiments in the dialog domain. We use the Switch-

Model	#Param	Validation	Test
Mikolov and Zweig [102] – RNN-LDA + KN-5 + cache	9M‡	-	92.0
Zaremba et al. [172] – LSTM	20M	86.2	82.7
Gal and Ghahramani [42] – Variational LSTM (MC)	20M	-	78.6
Kim et al. [69] – CharCNN	19M	-	78.9
Merity et al. [99] – Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. [49] – LSTM + continuous cache pointer†	-	-	72.1
Inan et al. [64] – Tied Variational LSTM + augmented loss	24M	75.7	73.2
Zilly et al. [180] – Variational RHN	23M	67.9	65.4
Zoph and Le [181] – NAS Cell	25M	-	64.0
Melis et al. [97] – 2-layer skip connection LSTM	24M	60.9	58.3
Merity et al. [100] – AWD-LSTM w/o finetune	24M	60.7	58.8
Merity et al. [100] – AWD-LSTM	24M	60.0	57.3
Ours – AWD-LSTM-MoS w/o finetune	22M	58.08	55.97
Ours – AWD-LSTM-MoS	22M	**56.54**	**54.44**
Merity et al. [100] – AWD-LSTM + continuous cache pointer†	24M	53.9	52.8
Krause et al. [78] – AWD-LSTM + dynamic evaluation†	24M	51.6	51.1
Ours – AWD-LSTM-MoS + dynamic evaluation†	22M	**48.33**	**47.69**

Table A.1: Single model perplexity on validation and test sets on Penn Treebank. Baseline results are obtained from Merity et al. [100] and Krause et al. [78]. † indicates using dynamic evaluation.

board dataset [45] preprocessed by Zhao et al. [176][3] to train a Seq2Seq [147] model with MoS added to the decoder RNN. Then, a Seq2Seq model using Softmax and another one augmented by MoC with comparable parameter sizes are used as baselines. For evaluation, we include both the perplexity and the precision/recall of Smoothed Sentence-level BLEU, as suggested by Zhao et al. [176]. When generating responses, we use beam search with beam size 10, restrict the maximum length to 30, and retain the top-5 responses.

The language modeling results on PTB and WT2 are presented in Table A.1 and Table A.2 respectively. With a comparable number of parameters, MoS outperforms all baselines with or without dynamic evaluation, and substantially improves over the current state of the art, by up to 3.6 points in perplexity.

The improvement on the large-scale dataset is even more significant. As shown in Table A.3, MoS outperforms Softmax by over 5.6 points in perplexity. It suggests the effectiveness of MoS is not limited to small datasets where many regularization techniques are used. Note that with limited computational resources, we didn't tune the hyper-parameters for MoS.

Further, the experimental results on Switchboard are summarized in Table A.4[4]. Clearly, on all metrics, MoS outperforms MoC and Softmax, showing its general effectiveness.

[3] `https://github.com/snakeztc/NeuralDialog-CVAE/tree/master/data`

[4] The numbers are not directly comparable to [176] since their Seq2Seq implementation and evaluation scripts are not publicly available.

Model	#Param	Validation	Test
Inan et al. [64] – Variational LSTM + augmented loss	28M	91.5	87.0
Grave et al. [49] – LSTM + continuous cache pointer†	-	-	68.9
Melis et al. [97] – 2-layer skip connection LSTM	24M	69.1	65.9
Merity et al. [100] – AWD-LSTM w/o finetune	33M	69.1	66.0
Merity et al. [100] – AWD-LSTM	33M	68.6	65.8
Ours – AWD-LSTM-MoS w/o finetune	35M	66.01	63.33
Ours – AWD-LSTM-MoS	35M	**63.88**	**61.45**
Merity et al. [100] – AWD-LSTM + continuous cache pointer †	33M	53.8	52.0
Krause et al. [78] – AWD-LSTM + dynamic evaluation†	33M	46.4	44.3
Ours – AWD-LSTM-MoS + dynamical evaluation†	35M	**42.41**	**40.68**

Table A.2: Single model perplexity over WikiText-2. Baseline results are obtained from Merity et al. [100] and Krause et al. [78]. † indicates using dynamic evaluation.

Model	#Param	Train	Validation	Test
Softmax	119M	41.47	43.86	42.77
MoS	113M	**36.39**	**38.01**	**37.10**

Table A.3: Perplexity comparison on 1B word dataset. Train perplexity is the average of the last 4,000 updates.

A.3.2 Ablation Study

To further verify the improvement shown above does come from the MoS structure rather than adding another hidden layer or finding a particular set of hyper-parameters, we conduct an ablation study on both PTB and WT2. Firstly, we compare MoS with an MoC architecture with the same number of layers, hidden sizes, and embedding sizes, which thus has the same number of parameters. In addition, we adopt the hyper-parameters used to obtain the best MoS model (denoted as MoS hyper-parameters), and train a baseline AWD-LSTM. To avoid distractive factors and save computational resources, all ablative experiments excluded the use of finetuing and dynamic evaluation.

The results are shown in Table A.5. Compared to the vanilla AWD-LSTM, though being more expressive, MoC performs only better on PTB, but worse on WT2. It suggests that simply adding another hidden layer or employing a mixture structure in the feature space does not guarantee a better performance. On the other hand, training AWD-LSTM using MoS hyper-parameters severely hurts the performance, which rules out hyper-parameters as the main source of improvement.

A.3.3 Verify the Role of Rank

While the study above verifies that MoS is the key to achieving the state-of-the-art performance, it is still not clear whether the superiority of MoS comes from its potential high rank, as suggested by our theoretical analysis in Section A.2. In the sequel, we take steps to verify this hypothesis.

- Firstly, we verify that MoS does induce a high-rank log-probability matrix empirically, while

Model	Perplexity	BLEU-1		BLEU-2		BLEU-3		BLEU-4	
		prec	recall	prec	recall	prec	recall	prec	recall
Seq2Seq-Softmax	34.657	0.249	0.188	0.193	0.151	0.168	0.133	0.141	0.111
Seq2Seq-MoC	33.291	0.259	0.198	0.202	0.159	0.176	0.140	0.148	0.117
Seq2Seq-MoS	**32.727**	**0.272**	**0.206**	**0.213**	**0.166**	**0.185**	**0.146**	**0.157**	**0.123**

Table A.4: Evaluation scores on Switchboard.

Model	PTB		WT2	
	Validation	**Test**	**Validation**	**Test**
AWD-LSTM-MoS	**58.08**	**55.97**	**66.01**	**63.33**
AWD-LSTM-MoC	59.82	57.55	68.76	65.98
AWD-LSTM (Merity et al. [100] hyper-parameters)	61.49	58.95	68.73	65.40
AWD-LSTM (MoS hyper-parameters)	78.86	74.86	72.73	69.18

Table A.5: Ablation study on Penn Treebank and WikiText-2 without finetuning or dynamical evaluation.

MoC and Softmax fail. On the validation or test set of PTB with tokens $\mathbf{X} = \{X_1, \ldots, X_T\}$, we compute the log probabilities $\{\log P(X_i \mid X_{<i}) \in \mathbb{R}^M\}_{t=1}^T$ for each token using all three models. Then, for each model, we stack all T log-probability vectors into a $T \times M$ matrix, resulting in $\hat{\mathbf{A}}_{\text{MoS}}$, $\hat{\mathbf{A}}_{\text{MoC}}$ and $\hat{\mathbf{A}}_{\text{Softmax}}$. Theoretically, the number of non-zero singular values of a matrix is equal to its rank. However, performing singular value decomposition of real valued matrices using numerical approaches often encounter roundoff errors. Hence, we adopt the expected roundoff error suggested by Press [120] when estimating the ranks of $\hat{\mathbf{A}}_{\text{MoS}}$, $\hat{\mathbf{A}}_{\text{MoC}}$ and $\hat{\mathbf{A}}_{\text{Softmax}}$.

The estimated ranks are shown in Table A.6. As predicted by our theoretical analysis, the matrix ranks induced by Softmax and MoC are both limited by the corresponding embedding sizes. By contrast, the matrix rank obtained from MoS does not suffer from this constraint, almost reaching full rank ($M = 10000$). In Section A.8.1, we give additional evidences for the higher rank of MoS.

- Secondly, we show that, before reaching full rank, increasing the number of mixture components in MoS also increases the rank of the log-probability matrix, which in turn leads to improved performance (lower perplexity). Specifically, on PTB, with other hyper-parameters fixed as used in section A.3.1, we vary the number of mixtures used in MoS and compare the corresponding empirical rank and test perplexity without finetuning. Table A.7 summarizes the results. This clear positive correlation between rank and performance strongly supports the our theoretical analysis in section A.2. Moreover, note that after reaching almost full rank (i.e., using 15 mixture components), further increasing the number of components degrades the performance due to overfitting (as we inspected the training and test perplexities).
- In addition, as performance improvement can often come from better regularization, we investigate whether MoS has a better, though unexpected, regularization effect compared to Softmax. We consider the 1B word dataset where overfitting is unlikely and no explicit regularization technique (e.g., dropout) is employed. As we can see from the left part of Table A.3, MoS and Softmax achieve a similar generalization gap, i.e., the performance gap between

Model	Validation	Test
Softmax	400	400
MoC	280	280
MoS	**9981**	**9981**

Table A.6: Rank comparison on PTB. To ensure comparable model sizes, the embedding sizes of Softmax, MoC and MoS are 400, 280, 280 respectively. The vocabulary size, i.e., M, is 10,000 for all models.

#Softmax	Rank	Perplexity
3	6467	58.62
5	8930	57.36
10	9973	56.33
15	9981	55.97
20	9981	56.17

Table A.7: Empirical rank and test perplexity on PTB with different number of Softmaxes.

the test set and the training set. It suggests both models have similar regularization effects. Meanwhile, MoS has a lower training perplexity compared to Softmax, indicating that the improvement of MoS results from improved expressiveness.

- The last evidence we provide is based on an *inverse* experiment. Empirically, we find that when Softmax does *not* suffer from a rank limitation, e.g., in character-level language modeling, using MoS will *not* improve the performance. Due to lack of space, we refer readers to Section A.8.2 for details.

A.3.4 Additional analysis

MoS computational time The expressiveness of MoS does come with a computational cost—computing a K-times larger Softmax. To give readers a concrete idea of the influence on training time, we perform detailed analysis in Section A.8.3. As we will see, computational wall time of MoS is actually *sub-linear* w.r.t. the number of Softmaxes K. In most settings, we observe a two to three times slowdown when using MoS with up to 15 mixture components.

Qualitative analysis Finally, we conduct a case study on PTB to see how MoS improves the next-token prediction in detail. Due to lack of space, we refer readers to Section A.8.4 for details. The key insight from the case study is that MoS is better at making context-dependent predictions. Specifically, given the same immediate preceding word, MoS will produce distinct next-step prediction based on long-term context in history. By contrast, the baseline often yields similar next-step prediction, independent of the long-term context.

A.4 Related work

In language modeling, Hutchinson et al. [62, 63] have previously considered the problem from a matrix rank perspective. However, their focus was to improve the generalization of Ngram language models via a sparse plus low-rank approximation. By contrast, as neural language models already generalize well, we focus on a high-rank neural language model that improves expressiveness without sacrificing generalization. Neubig and Dyer [110] proposed to mix Ngram and neural language models to unify and benefit from both. However, this mixture might not generalize well since an Ngram model, which has poor generalization, is included. Moreover,

the fact that the two components are separately trained can limit its expressiveness. Levy and Goldberg [87] also considered the matrix factorization perspective, but in the context of learning word embeddings.

In a general sense, Mixture of Softmaxes proposed in this work can be seen as a particular instantiation of the long-existing idea called Mixture of Experts (MoE) [65]. However, there are two core differences. Firstly, MoE has usually been instantiated as mixture of Gaussians to model data in continuous domains [10, 50, 65]. More importantly, the motivation of using the mixture structure is distinct. For Gaussian mixture models, the mixture structure is employed to allow for a parameterized multi-modal distribution. By contrast, Softmax by itself can parameterize a multi-modal distribution, and MoS is introduced to break the Softmax bottleneck as discussed in Section A.2.

There has been previous work [40, 138] proposing architectures that can be categorized as instantiations of MoC, since the mixture structure is employed in the feature space.[5] The target of Eigen et al. [40] is to create a more expressive feed-forward layer through the mixture structure. In comparison, Shazeer et al. [138] focuses on a sparse gating mechanism also on the feature level, which enables efficient conditional computation and allows the training of a very large neural architecture. In addition to having different motivations from our work, all these MoC variants suffer from the same rank limitation problem as discussed in Section A.2.

Finally, several previous works have tried to introduce latent variables into sequence modeling [9, 26, 27, 41, 43, 52]. Except for [27], these structures all define a continuous latent variable for each step of the RNN computation, and rely on the SGVB estimator [70] to optimize a variational lower bound of the log-likelihood. Since exact integration is infeasible, these models cannot estimate the likelihood (perplexity) exactly at test time. Moreover, for discrete data, the variational lower bound is usually too loose to yield a competitive approximation compared to standard auto-regressive models. As an exception, Chung et al. [27] utilizes Bernoulli latent variables to model the hierarchical structure in language, where the Bernoulli sampling is replaced by a thresholding operation at test time to give perplexity estimation.

A.5 Conclusions

Under the matrix factorization framework, the expressiveness of Softmax-based language models is limited by the dimension of the word embeddings, which is termed as the Softmax bottleneck. Our proposed MoS model improves the expressiveness over Softmax, and at the same time avoids overfitting compared to non-parametric models and naively increasing the word embedding dimensions. Our method improves the current state-of-the-art results on standard benchmarks by a large margin, which in turn justifies our theoretical reasoning: it is important to have a high-rank model for natural language.

[5] Although Shazeer et al. [138] name their architecture as MoE, it is not a standard MoE [65] and should be classified as MoC under our terminology.

A.6 Proofs

Proof of Property 1

Proof. For any $\mathbf{A}' \in F(\mathbf{A})$, let $P_{\mathbf{A}'}(X|C)$ denote the distribution defined by applying Softmax on the logits given by $\mathbf{A}'$. Consider row i column j, by definition any entry in $\mathbf{A}'$ can be expressed as $A'_{ij} = A_{ij} + \Lambda_{ii}$. It follows

$$P_{\mathbf{A}'}(x_j|c_i) = \frac{\exp A'_{ij}}{\sum_k \exp A'_{ik}} = \frac{\exp(A_{ij} + \Lambda_{ii})}{\sum_k \exp(A_{ik} + \Lambda_{ii})} = \frac{\exp A_{ij}}{\sum_k \exp A_{ik}} = P^*(x_j|c_i)$$

For any $\mathbf{A}'' \in \{\mathbf{A}'' \mid \text{Softmax}(\mathbf{A}'') = P^*\}$, for any i and j, we have

$$P_{\mathbf{A}''}(x_j|c_i) = P_{\mathbf{A}}(x_j|c_i)$$

It follows that for any i, j, and k,

$$\frac{P_{\mathbf{A}''}(x_j|c_i)}{P_{\mathbf{A}''}(x_k|c_i)} = \frac{\exp A''_{ij}}{\exp A''_{ik}} = \frac{\exp A_{ij}}{\exp A_{ik}} = \frac{P_{\mathbf{A}}(x_j|c_i)}{P_{\mathbf{A}}(x_k|c_i)}$$

As a result,

$$A''_{ij} - A_{ij} = A''_{ik} - A_{ik}$$

This means each row in $\mathbf{A}''$ can be obtained by adding a real number to the corresponding row in $\mathbf{A}$. Therefore, there exists a diagonal matrix $\mathbf{\Lambda} \in \mathbb{R}^{N \times N}$ such that

$$\mathbf{A}'' = \mathbf{A} + \mathbf{\Lambda}\mathbf{J}_{N,M}$$

It follows that $\mathbf{A}'' \in F(\mathbf{A})$. □

Proof of Property 2

Proof. For any $\mathbf{A}_1$ and $\mathbf{A}_2$ in $F(\mathbf{A})$, by definition we have $\mathbf{A}_1 = \mathbf{A} + \mathbf{\Lambda}_1\mathbf{J}_{N,M}$, and $\mathbf{A}_2 = \mathbf{A} + \mathbf{\Lambda}_2\mathbf{J}_{N,M}$ where $\mathbf{\Lambda}_1$ and $\mathbf{\Lambda}_2$ are two diagonal matrices. It can be rewritten as

$$\mathbf{A}_1 = \mathbf{A}_2 + (\mathbf{\Lambda}_1 - \mathbf{\Lambda}_2)\mathbf{J}_{N,M}$$

Let S be a maximum set of linearly independent rows in $\mathbf{A}_2$. Let $\mathbf{e}_N$ be an all-ones vector with dimension N. The i-th row vector $\mathbf{a}_{1,i}$ in $\mathbf{A}_1$ can be written as

$$\mathbf{a}_{1,i} = \mathbf{a}_{2,i} + (\Lambda_{1,ii} - \Lambda_{2,ii})\mathbf{e}_N$$

Because $\mathbf{a}_{2,i}$ is a linear combination of vectors in S, $\mathbf{a}_{1,i}$ is a linear combination of vectors in $S \cup \{\mathbf{e}_N\}$. It follows that

$$\text{rank}(\mathbf{A}_1) \leq \text{rank}(\mathbf{A}_2) + 1$$

Similarly, we can derive

$$\text{rank}(\mathbf{A}_2) \leq \text{rank}(\mathbf{A}_1) + 1$$

Therefore,

$$|\text{rank}(\mathbf{A}_1) - \text{rank}(\mathbf{A}_2)| \leq 1$$

□

Proof of Proposition 1

Proof. If there exists a parameter θ such that $P_\theta(X|c) = P^*(X|c)$ for all c in $\mathcal{L}$, by Lemma 1, we have $\mathbf{H}_\theta \mathbf{W}_\theta^\top \in F(\mathbf{A})$. As a result, there exists a matrix $\mathbf{A}' \in F(\mathbf{A})$ such that $\mathbf{H}_\theta \mathbf{W}_\theta^\top = \mathbf{A}'$. Because $\mathbf{H}_\theta$ and $\mathbf{W}_\theta$ are of dimensions $(N \times d)$ and $(M \times d)$ respectively, we have

$$d \geq \text{rank}(\mathbf{A}') \geq \min_{\mathbf{A}'' \in F(\mathbf{A})} \text{rank}(\mathbf{A}'')$$

If $d \geq \min_{\mathbf{A}'' \in F(\mathbf{A})} \text{rank}(\mathbf{A}'')$, there exist matrices $\mathbf{A}' \in F(\mathbf{A})$, $\mathbf{H}' \in \mathbb{R}^{N \times d}$ and $\mathbf{W}' \in \mathbb{R}^{M \times d}$, such that $\mathbf{A}'$ can be factorized as $\mathbf{A}' = \mathbf{H}'\mathbf{W}'^\top$. Because $\mathcal{U}$ is a universal approximator, there exists θ such that $\mathbf{H}_\theta = \mathbf{H}'$ and $\mathbf{W}_\theta = \mathbf{W}'$. By Lemma 1, $P_\theta(X|c) = P^*(X|c)$ for all c in $\mathcal{L}$. □

A.7 Experiment setting and Hyper-parameters

A.7.1 PTB and WT2

The hyper-parameters used for MoS in language modeling experiment is summarized below.

Hyper-parameter	**PTB**	**WT2**
Learning rate	20	15
Batch size	12	15
Embedding size	280	300
RNN hidden sizes	[960, 960, 620]	[1150,1150,650]
Number of mixture components	15	15
Word-level V-dropout	0.10	0.10
Embedding V-dropout	0.55	0.40
Hidden state V-dropout	0.20	0.225
Recurrent weight dropout [157]	0.50	0.50
Context vector V-dropout	0.30	0.30

Table A.8: Hyper-parameters used for MoS. V-dropout abbreviates variational dropout [42]. See [100] for more detailed descriptions.

The hyper-parameters used for dynamic evaluation of MoS is summarized below.

Hyper-parameter	**PTB**	**WT2**
Batch size	100	100
learning rate (η)	0.002	0.002
ϵ	0.001	0.002
λ	0.075	0.02

Table A.9: Hyper-parameters used for dynamic evaluation of MoS. See [78] for more detailed descriptions.

A.7.2 1B Word Dataset

For training, we use all of the 100 training shards. For validation, we use two shards from the heldout set, namely `[heldout-00, heldout-10]`. For test, we use another three shards from the heldout set, namely `[heldout-20, heldout-30, heldout-40]`.

The hyper-parameters are listed below.

Hyper-parameter	Softmax	MoS-7
Learning rate	20	20
Batch size	60	60
BPTT langth	35	35
Embedding size	1024	900
RNN hidden sizes	[1024, 1024]	[1024,1024]
Dropout rate	0	0

Table A.10: Hyper-parameters used for Softmax and MoS in experiment on 1B word dataset.

A.8 Additional experiments

A.8.1 Higher empirical rank of MoS compared to MoC and Softmax

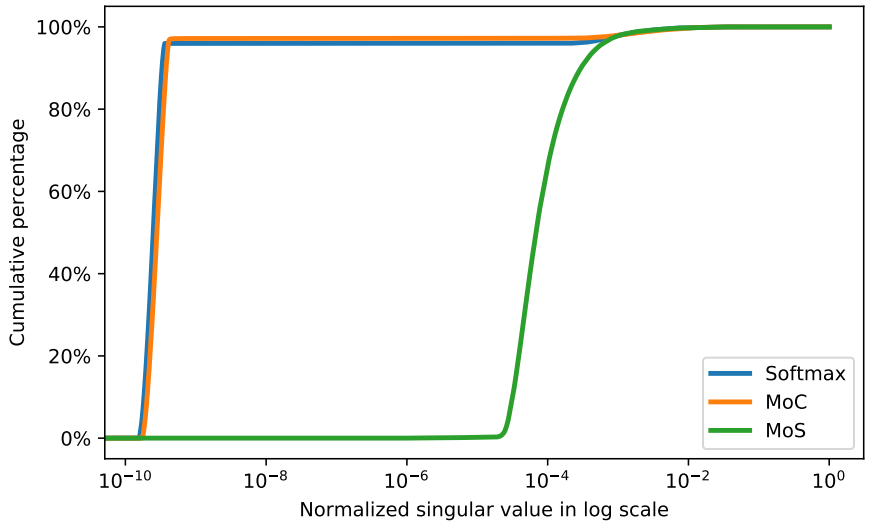


Figure A.1: Cumulative percentage of normalized singulars given a value in $[0, 1]$.

In section 4.3, we compute the rank of different models based on the non-zero singular values of the empirical log-likelihood matrix. Since there can be roundoff mistakes, a less error-prone approach is to directly study the distribution of singular values. Specifically, if more singular values have relatively larger magnitude, the rank of the matrix tends to be higher. Motivated from

this intuition, we visualize the distribution of the singular values. To account for the different magnitudes of singular values from different models, we first normalize all singular values to $[0, 1]$. Then, we plot the cumulative percentage of normalized singular values, i.e., percentage of normalized singular values below a threshold, in Figure A.1. As we can see, most of the singular values of Softmax and MoC concentrate on an area with very low values. In comparison, the concentration area of the MoS singular values is not only several orders larger, but also spans a much wider region. Intuitively, MoS utilizes the corresponding singular vectors to capture a larger and more diverse set of contexts.

Model	Validation	Test
Softmax	4.869	4.763
MoC	4.955	4.864
MoS	**5.400**	**5.284**

Table A.11: Empirical expected pairwise KLD on PTB.

What's more, another indicator of high rank is that the model can precisely capture the nuance of difference contexts. If a model can better capture the distinctions among contexts, we expect the next-step conditional distributions to be less similar to each on average. Based on this intuition, we use the expected pairwise Kullback–Leibler divergence (KLD), i.e., $\mathbb{E}_{c,c' \sim \mathcal{C}} \left[\mathrm{KLD}(P(X \mid c) \| P(X \mid c'))\right]$ where $\mathcal{C}$ denotes all possible contexts, as another metric to evaluate the ranks of the three models (MoS, MoC and Softmax). Practically, we sample c, c' from validation or test data of PTB to get the empirical estimations for the three models, which are shown in the right half of Table A.11. As we expected, MoS achieves higher expected pairwise KLD, indicating its superiority in covering more contexts of the next-token distribution.

A.8.2 An inverse experiment on character-level language modeling

Model	#Param	Train	Dev	Test
Softmax (hid1024, emb1024)	8.42M	1.35	1.41	1.49
MoS-7 (hid910, emb510)	8.45M	1.35	1.40	1.49
MoS-7 (hid750, emb750)	8.45M	1.38	1.42	1.50
MoS-10 (hid860, emb452)	8.43M	1.35	1.41	1.49
MoS-10 (hid683, emb683)	8.43M	1.38	1.42	1.50

Table A.12: BPC comparison on text8. "-n" indicates using n mixtures. "hid" and "emb" denote the hidden size and embedding size.

Here, we detail the inverse experiment, which shows that when Softmax does *not* suffer from a rank limitation, using MoS will *not* improve the performance. Notice that character-level language modeling (CharLM) is exactly such a problem, because the rank of the log-likelihood matrix is upper bounded by the vocabulary size, and CharLM usually has a very limited vocabulary (tens

of characters). In this case, with the embedding size being hundreds in practice, Softmax is no longer a bottleneck in this task. Hence, we expect MoS to yield similar performance to Softmax on CharLM.

We conduct experiments of CharLM using the text8 dataset [95], which consists of 100M characters including only alphabetical characters and spaces derived from Wikipedia. We follow Mikolov et al. [104] and use the first 90M characters for training, the next 5M for validation and the final 5M for testing. The standard evaluation metric bit-per-character (BPC) is employed. We employ a 1-layer 1024-unit LSTM followed by Softmax as the baseline. For MoS, we consider 7 or 10 mixtures and reduce the hidden and/or embedding size to match the baseline capacity. When decreasing the hidden and/or embedding size, we either keep both the same, or make the hidden size relatively larger. The results are summarized in Table A.12. Clearly, the Softmax and MoS obtain the same BPC on the test set and comparable BPC on the validation set, which well match our hypothesis. Since the only difference in word-level language modeling is the existence of the Softmax bottleneck, the distinct behavior of MoS again supports our hypothesis that it is solving the Softmax bottleneck problem.

A.8.3 MoS Computational Time

Model	PTB/bs	PTB/best-1	WT2/bs	WT2/best-1	WT2/best-3	1B/bs	1B/best-1	1B/best-3
Softmax	1x	1x	1x	1x	1x	1x	1x	1x
MoS-5	1.2x	–	1.3x	–	–	–	–	–
MoS-7	–	–	–	–	–	3.8x	5.7x	2.1x
MoS-10	1.6x	–	1.9x	–	–	–	–	–
MoS-15	1.9x	2.8x	2.5x	6.4x	2.9x	–	–	–

Table A.13: Training time slowdown compared to Softmax. MoS-K means using K mixture components. "bs" indicates Softmax and MoS use the same batch sizes on one GPU. "best-1" and "best-3" refer to the settings where Softmax and MoS obtain their own best perplexity, with 1 and 3 GPUs respectively.

We evaluate the additional computational cost introduced by MoS. We consider two sets of controlled experiments. In the first set, we compare the training time of MoS and Softmax using the same batch sizes. In the second set, we compare the training time of two methods using the hyper-parameter settings that achieve the best performance for each model (i.e., the settings in Tables A.1, A.2, and A.3). In both sets, we control two models to have comparable model sizes.

The results on the three datasets are shown in Table A.13. Thanks to the efficiency of matrix multiplication on GPU, the computational wall time of MoS is actually sub-linear w.r.t. the number of Softmaxes K. In most settings, we observe a two to three times slowdown when using MoS. Specifically, the "bs" setting measures the computational cost introduced by MoS given enough memory, which is 1.9x, 2.5x, and 3.8x slowdown on PTB, WT2, and 1B respectively. The "best-1" setting is usually slower compared to "bs", because a single batch does not fit into the memory of a single GPU using MoS, in which case we have to split one batch into multiple small ones, resulting in further slowdown. In this sense, the gap between "best-1" and "bs" measures the computational cost introduced due to the increase of memory consumed by MoS. The "best-3"

alleviates this issue by using three GPUs, which allows larger-batch training for MoS. In this case, we reduce the computational cost to 2.9x on WT2 and 2.1x on 1B with our best performing model.

Note that the computational cost is closely related to the batch size, which is interleaved with optimization. Though how batch sizes affect optimization remains an open question and might be task dependent, we believe the "best-1" and "best-3" settings well reflect the actual computational cost brought by MoS on language modeling tasks.

A.8.4 Qualitative Analysis

Since MoC shows a stronger performance than Softmax on PTB, the qualitative study focuses on the comparison between MoC and MoS. Concretely, given the same context (previous tokens), we search for prediction steps where MoS achieves lower negative log loss than MoC by a margin. We show some representative cases in Table A.14 with the following observations:

- Comparing the first two cases, given the same preceding word "N", MoS flexibly adjusts its top predictions based on the different topic quantities being discussed in the context. In comparison, MoC emits quite similar top choices regardless of the context, suggesting its inferiority in make context-dependent predictions.
- In the 3rd case, the context is about international politics, where country/region names are likely to appear. MoS captures this nuance well, and yields top choices that can be used to complete a country name given the immediate preceding word "south". Similarly, in the 4th case, MoS is able to include "ual", a core entity of discussion in the context, in its top predictions. In contrast, MoC gives rather generic predictions irrieselevant to the context in both cases.
- For the 5th and the 6th example, we see MoS is able to exploit less common words accurately according to the context, while MoC fails to yield such choices. This well matches our analysis that MoS has the capacity of modeling context-dependent language.

#1 Context	managed properly and with a long-term outlook these can become investment-grade quality properties <eos> canadian <unk> production totaled N metric tons in the week ended oct. N up N N from the preceding week 's total of N __?__				
MoS top-5	million 0.38	**tons 0.24**	billion 0.09	barrels 0.06	ounces 0.04
MoC top-5	billion 0.39	million 0.36	trillion 0.05	<eos> 0.04	N 0.03
Reference	canadian <unk> production totaled N metric tons in the week ended oct. N up N N from the preceding week 's total of N **tons** statistics canada a federal agency said <eos>				
#2 Context	the thriving <unk> street area offers <unk> of about $ N a square foot as do <unk> locations along lower fifth avenue <eos> by contrast <unk> in the best retail locations in boston san francisco and chicago rarely top $ N __?__				
MoS top-5	<eos> 0.36	**a 0.13**	to 0.07	for 0.07	and 0.06
MoC top-5	million 0.39	billion 0.36	<eos> 0.05	to 0.04	of 0.03
Reference	by contrast <unk> in the best retail locations in boston san francisco and chicago rarely top $ N **a** square foot <eos>				
#3 Context	as other <unk> governments particularly poland and the soviet union have recently discovered initial steps to open up society can create a momentum for radical change that becomes difficult if not impossible to control <eos> as the days go by the south __?__				
MoS top-5	africa 0.15	**african 0.15**	<eos> 0.14	korea 0.08	korean 0.05
MoC top-5	<eos> 0.38	and 0.08	of 0.06	or 0.05	<unk> 0.04
Reference	as the days go by the south **african** government will be ever more hard pressed to justify the continued <unk> of mr. <unk> as well as the continued banning of the anc and enforcement of the state of emergency <eos>				
#4 Context	shares of ual the parent of united airlines were extremely active all day friday reacting to news and rumors about the proposed $ N billion buy-out of the airline by an <unk> group <eos> wall street 's takeover-stock speculators or risk arbitragers had placed unusually large bets that a takeover would succeed and __?__				
MoS top-5	the 0.14	that 0.07	**ual 0.07**	<unk> 0.03	it 0.02
MoC top-5	the 0.10	<unk> 0.06	that 0.05	in 0.02	it 0.02
Reference	wall street 's takeover-stock speculators or risk arbitragers had placed unusually large bets that a takeover would succeed and **ual** stock would rise <eos>				
#5 Context	the government is watching closely to see if their presence in the <unk> leads to increased <unk> protests and violence if it does pretoria will use this as a reason to keep mr. <unk> behind bars <eos> pretoria has n't forgotten why they were all sentenced to life <unk> in the first place for sabotage and __?__				
MoS top-5	<unk> 0.47	violence 0.11	**conspiracy 0.03**	incest 0.03	civil 0.03
MoC top-5	<unk> 0.41	the 0.03	a 0.02	other 0.02	in 0.01
Reference	pretoria has n't forgotten why they were all sentenced to life <unk> in the first place for sabotage and **conspiracy** to <unk> the government <eos>				
#6 Context	china 's <unk> <unk> program has achieved some successes in <unk> runaway economic growth and stabilizing prices but has failed to eliminate serious defects in state planning and an <unk> drain on state budgets <eos> the official china daily said retail prices of <unk> foods have n't risen since last december but acknowledged that huge government __?__				
MoS top-5	**subsidies 0.15**	spending 0.08	officials 0.04	costs 0.04	<unk> 0.03
MoC top-5	officials 0.04	figures 0.03	efforts 0.03	<unk> 0.03	costs 0.03
Reference	the official china daily said retail prices of <unk> foods have n't risen since last december but acknowledged that huge government **subsidies** were a main factor in keeping prices down <eos>				

Table A.14: Compaison of next-token prediction on Penn Treebank test data. N stands for a number as

Appendix B

Calibrating Energy-based Generative Adversarial Networks

B.1 Introduction

Generative Adversarial Networks (GANs) [46] represent an important milestone on the path towards more effective generative models. GANs cast generative model training as a minimax game between a generative network (*generator*), which maps a random vector into the data space, and a discriminative network (*discriminator*), whose objective is to distinguish generated samples from real samples. Multiple researchers [121, 134, 175] have shown that the adversarial interaction with the discriminator can result in a generator that produces compelling samples. The empirical successes of the GAN framework were also supported by the theoretical analysis of Goodfellow et al., who showed that, under certain conditions, the distribution produced by the generator converges to the true data distribution, while the discriminator converges to a degenerate uniform solution.

While GANs have excelled as compelling sample generators, their use as general purpose probabilistic generative models has been limited by the difficulty in using them to provide density estimates or even unnormalized energy values for sample evaluation.

It is tempting to consider the GAN discriminator as a candidate for providing this sort of scoring function. Conceptually, it is a trainable sample evaluation mechanism that – owing to GAN training paradigm – could be closely calibrated to the distribution modeled by the generator. If the discriminator could retain fine-grained information of the relative quality of samples, measured for instance by probability density or unnormalized energy, it could be used as an evaluation metric. Such data-driven evaluators would be highly desirable for problems where it is difficult to define evaluation criteria that correlate well with human judgment. Indeed, the real-valued discriminator of the recently introduced energy-based GANs [175] might seem like an ideal candidate energy function. Unfortunately, as we will show, the degenerate fate of the GAN discriminator at the optimum equally afflicts the energy-based GAN of Zhao et al..

In this paper we consider the questions: (i) does there exists an adversarial framework that induces a non-degenerate discriminator, and (ii) if so, what form will the resulting discriminator take? We introduce a novel adversarial learning formulation, which leads to a non-degenerate

discriminator while ensuring the generator distribution matches the data distribution at the global optimum. We derive a general analytic form of the optimal discriminator, and discuss its properties and their relationship to the specific form of the training objective. We also discuss the connection between the proposed formulation and existing alternatives such as the approach of [68]. Finally, for a specific instantiation of the general formulation, we investigate two approximation techniques to optimize the training objective, and verify our results empirically.

B.2 Related Work

Following a similar motivation, the field of Inverse Reinforcement Learning (IRL) [111] has been exploring ways to recover the "intrinsic" reward function (analogous to the discriminator) from observed expert trajectories (real samples). Taking this idea one step further, apprenticeship learning or imitation learning [1, 179] aims at learning a policy (analogous to the generator) using the reward signals recovered by IRL. Notably, Ho and Ermon draw a connection between imitation learning and GAN by showing that the GAN formulation can be derived by imposing a specific regularization on the reward function. Also, under a special case of their formulation, Ho and Ermon provide a duality-based interpretation of the problem, which inspires our theoretical analysis. However, as the focus of [57] is only on the policy, the authors explicitly propose to bypass the intermediate IRL step, and thus provide no analysis of the learned reward function.

The GAN models most closely related to our proposed framework are energy-based GAN models of Zhao et al. [175] and Kim and Bengio [68]. In the next section, We show how one can derive both of these approaches from different assumptions regarding regularization of the generative model.

B.3 Alternative Formulation of Adversarial Training

B.3.1 Background

Before presenting the proposed formulation, we first state some basic assumptions required by the analysis, and introduce notations used throughout the paper.

Following the original work on GANs [46], our analysis focuses on the non-parametric case, where all models are assumed to have infinite capacities. While many of the non-parametric intuitions can directly transfer to the parametric case, we will point out cases where this transfer fails. We assume a finite data space throughout the analysis, to avoid technical machinery out of the scope of this paper. Our results, however, can be extended to continuous data spaces, and our experiments are indeed performed on continuous data.

Let $\mathcal{X}$ be the data space under consideration, and $\mathcal{P} = \{p \mid p(x) \geq 0, \forall x \in \mathcal{X}, \sum_{x \in \mathcal{X}} p(x) = 1\}$ be the set of all proper distributions defined on $\mathcal{X}$. Then, $p_{\text{data}} \in \mathcal{P} : \mathcal{X} \mapsto \mathbb{R}$ and $p_{\text{gen}} \in \mathcal{P} : \mathcal{X} \mapsto \mathbb{R}$ will denote the true data distribution and the generator distribution. $\mathbb{E}_{x \sim p} f(x)$ denotes the expectation of the quantity $f(x)$ w.r.t. x drawn from p. Finally, the term "discriminator" will refer to any structure that provides training signals to the generator based on some measure of difference between the generator distribution and the real data distribution, which which includes but is not limited to f-divergence.

B.3.2 Proposed Formulation

In order to understand the motivation of the proposed approach, it is helpful to analyze the optimization dynamics near convergence in GANs first.

When the generator distribution matches the data distribution, the training signal (gradient) w.r.t. the discriminator vanishes. At this point, assume the discriminator still retains density information, and views some samples as more real and others as less. This discriminator will produce a training signal (gradient) w.r.t. the generator, pushing the generator to generate samples that appear more real to the discriminator. Critically, this training signal is the sole driver of the generator's training. Hence, the generator distribution will diverge from the data distribution. In other words, as long as the discriminator retains relative density information, the generator distribution cannot stably match the data distribution. Thus, in order to keep the generator stationary as the data distribution, the discriminator must assign flat (exactly the same) density to all samples at the optimal.

From the analysis above, the fundamental difficulty is that the generator only receives a single training signal (gradient) from the discriminator, which it has to follow. To keep the generator stationary, this single training signal (gradient) must vanish, which requires a degenerate discriminator. In this work, we propose to tackle this single training signal constraint directly. Specifically, we introduce a novel adversarial learning formulation which incorporates an additional training signal to the generator, such that this additional signal can

- balance (cancel out) the discriminator signal at the optimum, so that the generator can stay stationary even if the discriminator assigns non-flat density to samples
- cooperate with the discriminator signal to make sure the generator converges to the data distribution, and the discriminator retains the *correct* relative density information

The proposed formulation can be written as the following minimax training objective,

$$\max_{c} \min_{p_{\text{gen}} \in \mathcal{P}} \quad \mathbb{E}_{x \sim p_{\text{gen}}}\Big[c(x)\Big] - \mathbb{E}_{x \sim p_{\text{data}}}\Big[c(x)\Big] + K(p_{\text{gen}}), \tag{B.1}$$

where $c(x) : \mathcal{X} \mapsto \mathbb{R}$ is the discriminator that assigns each data point an unbounded scalar cost, and $K(p_{\text{gen}}) : \mathcal{P} \mapsto \mathbb{R}$ is some (functionally) differentiable, convex function of p_{gen}. Compared to the original GAN, despite the similar minimax surface form, the proposed fomulation has two crucial distinctions.

Firstly, while the GAN discriminator tries to distinguish "fake" samples from real ones using binary classification, the proposed discriminator achieves that by assigning lower cost to real samples and higher cost to "fake" one. This distinction can be seen from the first two terms of Eqn. (B.1), where the discriminator $c(x)$ is trained to widen the expected cost gap between "fake" and real samples, while the generator is adversarially trained to minimize it. In addition to the different adversarial mechanism, a calibrating term $K(p_{\text{gen}})$ is introduced to provide a countervailing source of training signal for p_{gen} as we motivated above. For now, the form of $K(p_{\text{gen}})$ has not been specified. But as we will see later, its choice will directly decide the form of the optimal discriminator $c^*(x)$.

With the specific optimization objective, we next provide theoretical characterization of both the generator and the discriminator at the global optimum.

Define $L(p_{\text{gen}}, c) = \mathbb{E}_{x\sim p_{\text{gen}}}\Big[c(x)\Big] - \mathbb{E}_{x\sim p_{\text{data}}}\Big[c(x)\Big] + K(p_{\text{gen}})$, then $L(p_{\text{gen}}, c)$ is the Lagrange dual function of the following optimization problem

$$\begin{aligned} \min_{p_{\text{gen}}\in\mathcal{P}} \quad & K(p_{\text{gen}}) \\ \text{s.t.} \quad & p_{\text{gen}}(x) - p_{\text{data}}(x) = 0, \forall x \in \mathcal{X} \end{aligned} \tag{B.2}$$

where $c(x), \forall x$ appears in $L(p_{\text{gen}}, c)$ as the dual variables introduced for the equality constraints. This duality relationship has been observed previously in [57, equation (7)] under the adversarial imitation learning setting. However, in their case, the focus was fully on the generator side (induced policy), and no analysis was provided for the discriminator (reward function).

In order to characterize c^*, we first expand the set constraint on p_{gen} into explicit equality and inequality constraints:

$$\begin{aligned} \min_{p_{\text{gen}}} \quad & K(p_{\text{gen}}) \\ \text{s.t.} \quad & p_{\text{gen}}(x) - p_{\text{data}}(x) = 0, \forall x \\ & -p_{\text{gen}}(x) \le 0, \forall x \\ & \sum_{x\in\mathcal{X}} p_{\text{gen}}(x) - 1 = 0. \end{aligned} \tag{B.3}$$

Notice that $K(p_{\text{gen}})$ is a convex function of $p_{\text{gen}}(x)$ by definition, and both the equality and inequality constraints are affine functions of $p_{\text{gen}}(x)$. Thus, problem (B.2) is a convex optimization problem. What's more, since (i) dom_K is open, and (ii) there exists a feasible solution $p_{\text{gen}} = p_{\text{data}}$ to (B.3), by the refined Slater's condition [14, page 226], we can further verify that strong duality holds for (B.3). With strong duality, a typical approach to characterizing the optimal solution is to apply the Karush-Kuhn-Tucker (KKT) conditions, which gives rise to this theorem:

Proposition 2. *By the KKT conditions of the convex problem* (B.3)*, at the global optimum, the optimal generator distribution p^*_{gen} matches the true data distribution p_{data}, and the optimal discriminator $c^*(x)$ has the following form:*

$$\begin{aligned} & c^*(x) = -\frac{\partial K(p_{gen})}{\partial p_{gen}(x)}\bigg|_{p_{gen}=p_{data}} - \lambda^* + \mu^*(x), \forall x \in \mathcal{X}, \\ & \textit{where} \quad \mu^*(x) = \begin{cases} 0, & p_{data}(x) > 0 \\ u_x, & p_{data}(x) = 0 \end{cases}, \\ & \qquad \lambda^* \in \mathbb{R}, \textit{ is an under-determined real number independent of } x, \\ & \qquad u_x \in \mathbb{R}_+, \textit{ is an under-determined non-negative real number.} \end{aligned} \tag{B.4}$$

The detailed proof of proposition 2 is provided in section B.7.1. From (B.4), we can see the exact form of the optimal discriminator depends on the term $K(p_{\text{gen}})$, or more specifically its gradient. But, before we instantiate $K(p_{\text{gen}})$ with specific choices and show the corresponding forms of $c^*(x)$, we first discuss some general properties of $c^*(x)$ that do not depend on the choice of K.

Weak Support Discriminator. As part of the optimal discriminator function, the term $\mu^*(x)$ plays the role of support discriminator. That is, it tries to distinguish the support of the data distribution, i.e. $\text{SUPP}(p_{\text{data}}) = \{x \in \mathcal{X} \mid p_{\text{data}}(x) > 0\}$, from its complement set

with zero-probability, i.e. $\text{SUPP}(p_{\text{data}})^{\complement} = \{x \in \mathcal{X} \mid p_{\text{data}}(x) = 0\}$. Specifically, for any $x \in \text{SUPP}(p_{\text{data}})$ and $x' \in \text{SUPP}(p_{\text{data}})^{\complement}$, it is guaranteed that $\mu^*(x) \leq \mu^*(x')$. However, because $\mu^*(\cdot)$ is under-determined, there is nothing preventing the inequality from degenerating into an equality. Therefore, we name it the *weak* support discriminator. But, in all cases, $\mu^*(\cdot)$ assigns zero cost to all data points within the support. As a result, it does not possess any fine-grained density information inside of the data support. It is worth pointing out that, in the parametric case, because of the smoothness and the generalization properties of the parametric model, the learned discriminator may generalize beyond the data support.

Global Bias. In (B.4), the term λ^* is a scalar value shared for all x. As a result, it does not affect the relative cost among data points, and only serves as a global bias for the discriminator function.

Having discussed general properties, we now consider some specific cases of the convex function K, and analyze the resulting optimal discriminator $c^*(x)$ in detail.

1. First, let us consider the case where K is the negative entropy of the generator distribution, i.e. $K(p_{\text{gen}}) = -H(p_{\text{gen}})$. Taking the derivative of the negative entropy w.r.t. $p_{\text{gen}}(x)$, we have

$$c^*_{\text{ent}}(x) = -\log p_{\text{data}}(x) - 1 - \lambda^* + \mu^*(x), \forall x \in \mathcal{X}, \tag{B.5}$$

   where $\mu^*(x)$ and λ^* have the same definitions as in (B.4).

   Up to a constant, this form of $c^*_{\text{ent}}(x)$ is exactly the energy function of the data distribution $p_{\text{data}}(x)$. This elegant result has deep connections to several existing formulations, which include max-entropy imitation learning [179] and the directed-generator-trained energy-based model [68]. The core difference is that these previous formulations are originally derived from maximum-likelihood estimation, and thus the minimax optimization is only implicit. In contrast, with an explicit minimax formulation we can develop a better understanding of the induced solution. For example, the global bias λ^* suggests that there exists more than one stable equilibrium the optimal discriminator can actually reach. Further, $\mu^*(x)$ can be understood as a support discriminator that poses extra cost on generator samples which fall in zero-probability regions of data space.

2. When $K(p_{\text{gen}}) = \frac{1}{2}\sum_{x\in\mathcal{X}} p_{\text{gen}}(x)^2 = \frac{1}{2}\|p_{\text{gen}}\|_2^2$, which can be understood as posing ℓ_2 regularization on p_{gen}, we have $\frac{\partial K(p_{\text{gen}})}{\partial p_{\text{gen}}(x)}\Big|_{p_{\text{gen}}=p_{\text{data}}} = p_{\text{data}}(x)$, and it follows

$$c^*_{\ell_2}(x) = -p_{\text{data}}(x) - \lambda^* + \mu^*(x), \forall x \in \mathcal{X}, \tag{B.6}$$

   with $\mu^*(x), \lambda^*$ similarly defined as in (B.4).

   Surprisingly, the result suggests that the optimal discriminator $c^*_{\ell_2}(x)$ directly recovers the negative probability $-p_{\text{data}}(x)$, shifted by a constant. Thus, similar to the entropy solution (B.5), it fully retains the relative density information of data points within the support.

   However, because of the under-determined term $\mu^*(x)$, we cannot recover the distribution density p_{data} exactly from either $c^*_{\ell_2}$ or c^*_{ent} if the data support is finite. Whether this ambiguity can be resolved is beyond the scope of this paper, but poses an interesting research problem.

3. Finally, let's consider consider a degenerate case, where $K(p_{\text{gen}})$ is a constant. That is, we don't provide any additional training signal for pgen at all. With $K(p_{\text{gen}}) = \text{const}$, we simply

have

$$c^*_{\text{cst}}(x) = -\lambda^* + \mu^*(x), \forall x \in \mathcal{X}, \tag{B.7}$$

whose discriminative power is fully controlled by the weak support discriminator $\mu^*(x)$. Thus, it follows that $c^*_{\text{cst}}(x)$ won't be able to discriminate data points within the support of p_{data}, and its power to distinguish data from $\text{SUPP}(p_{\text{data}})$ and $\text{SUPP}(p_{\text{data}})^{\complement}$ is weak. This closely matches the intuitive argument in the beginning of this section.

Note that when $K(p_{\text{gen}})$ is a constant, the objective function (B.1) simplifies to:

$$\max_{c} \min_{p_{\text{gen}} \in \mathcal{P}} \quad \mathbb{E}_{x \sim p_{\text{gen}}}\Big[c(x)\Big] - \mathbb{E}_{x \sim p_{\text{data}}}\Big[c(x)\Big], \tag{B.8}$$

which is very similar to the EBGAN objective [175, equation (2) and (4)]. As we show in section B.7.2, compared to the objective in (B.8), the EBGAN objective puts extra constraints on the allowed discriminator function. In spite of that, the EBGAN objective suffers from the single-training-signal problem and does not guarantee that the discriminator will recover the real energy function (see section B.7.2 for detailed analysis).

As we finish the theoretical analysis of the proposed formulation, we want to point out that simply adding the same term $K(p_{\text{gen}})$ to the original GAN formulation will not lead to both a generator that matches the data distribution, and a discriminator that retains the density information (see section B.7.3 for detailed analysis).

B.4 Parametric Instantiation with Entropy Approximation

While the discussion in previous sections focused on the non-parametric case, in practice we are limited to a finite amount of data, and the actual problem involves high dimensional continuous spaces. Thus, we resort to parametric representations for both the generator and the discriminator. In order to train the generator using standard back-propagation, we do not parametrize the generator distribution directly. Instead, we parametrize a directed generator network that transforms random noise $z \sim p_z(z)$ to samples from a continuous data space $\mathbb{R}^n$. Consequently, we don't have analytical access to the generator distribution, which is defined implicitly by the generator network's noise→data mapping. However, the regularization term $K(p_{\text{gen}})$ in the training objective (B.1) requires the generator distribution. Faced with this problem, we focus on the max-entropy formulation, and exploit two different approximations of the regularization term $K(p_{\text{gen}}) = -H(p_{\text{gen}})$.

B.4.1 Nearest-Neighbor Entropy Gradient Approximation

The first proposed solution is built upon an intuitive interpretation of the entropy gradient. Firstly, since we construct p_{gen} by applying a deterministic, differentiable transform g_θ to samples z from a fixed distribution p_z, we can write the gradient of $H(p_{\text{gen}})$ with respect to the generator parameters θ as follows:

$$-\nabla_\theta H(p_{\text{gen}}) = \mathbb{E}_{z \sim p_z}\left[\nabla_\theta \log p_{\text{gen}}(g_\theta(z))\right] = \mathbb{E}_{z \sim p_z}\left[\frac{\partial g_\theta(z)}{\partial \theta}\frac{\partial \log p_{\text{gen}}(g_\theta(z))}{\partial g_\theta(z)}\right], \tag{B.9}$$

where the first equality relies on the "reparametrization trick". Equation B.9 implies that, if we can compute the gradient of the generator log-density $\log p_{\text{gen}}(x)$ w.r.t. any $x = g_\theta(z)$, then we can directly construct the Monte-Carlo estimation of the entropy gradient $\nabla_\theta H(p_{\text{gen}})$ using samples from the generator.

Intuitively, for any generated data $x = g_\theta(z)$, the term $\frac{\partial \log p_{\text{gen}}(x)}{\partial x}$ essentially describes the direction of *local change* in the sample space that will increase the log-density. Motivated by this intuition, we propose to form a local Gaussian approximation p^i_{gen} of p_{gen} around each point x_i in a batch of samples $\{x_1, ..., x_n\}$ from the generator, and then compute the gradient $\frac{\partial \log p_{\text{gen}}(x_i)}{\partial x_i}$ based on the Gaussian approximation. Specifically, each local Gaussian approximation p^i_{gen} is formed by finding the k nearest neighbors of x_i in the batch $\{x_1, ..., x_n\}$, and then placing an isotropic Gaussian distribution at their mean (i.e. maximimum likelihood). Based on the isotropic Gaussian approximation, the resulting gradient has the following form

$$\frac{\partial \log p_{\text{gen}}(x_i)}{\partial x_i} \approx \mu_i - x_i, \quad \text{where } \mu_i = \frac{1}{k} \sum_{x' \in \text{KNN}(x_i)} x' \text{ is the mean of the Gaussian} \tag{B.10}$$

Finally, note the scale of this gradient approximation may not be reliable. To fix this problem, we normalize the approximated gradient into unit norm, and use a single hyper-parameter to model the scale for all x, leading to the following entropy gradient approximation

$$-\nabla_\theta H(p_{\text{gen}}) \approx \alpha \frac{1}{k} \sum_{x_i = g_\theta(z_i)} \frac{\mu_i - x_i}{\|\mu_i - x_i\|_2} \tag{B.11}$$

where α is the hyper-parameter and μ_i is defined as in equation (B.10).

An obvious weakness of this approximation is that it relies on Euclidean distance to find the k nearest neighbors. However, Euclidean distance is usually not the proper metric to use when the effective dimension is very high. As the problem is highly challenging, we leave it for future work.

B.4.2 Variational Lower bound on the Entropy

Another approach we consider relies on defining and maximizing a variational lower bound on the entropy $H(p_{\text{gen}}(x))$ of the generator distribution. We can define the joint distribution over observed data and the noise variables as $p_{\text{gen}}(x, z) = p_{\text{gen}}(x \mid z) p_{\text{gen}}(z)$, where simply $p_{\text{gen}}(z) = p_z(z)$ is a fixed prior. Using the joint, we can also define the marginal $p_{\text{gen}}(x)$ and the posterior $p_{\text{gen}}(z \mid x)$. We can also write the mutual information between the observed data and noise variables as:

$$\begin{aligned} I(p_{\text{gen}}(x); p_{\text{gen}}(z)) &= H(p_{\text{gen}}(x)) - H(p_{\text{gen}}(x \mid z)) \\ &= H(p_{\text{gen}}(z)) - H(p_{\text{gen}}(z \mid x)), \end{aligned} \tag{B.12}$$

where $H(p_{\text{gen}}(. \mid .))$ denotes the conditional entropy. By reorganizing terms in this definition, we can write the entropy $H(p_{\text{gen}}(x))$ as:

$$H(p_{\text{gen}}(x)) = H(p_{\text{gen}}(z)) - H(p_{\text{gen}}(z \mid x)) + H(p_{\text{gen}}(x \mid z)) \tag{B.13}$$

We can think of $p_{\text{gen}}(x \mid z)$ as a peaked Gaussian with a fixed, diagonal covariance, and hence its conditional entropy is constant and can be dropped. Furthermore, $H(p_{\text{gen}}(z))$ is also assumed to be fixed a priori. Hence, we can maximize $H(p_{\text{gen}}(x))$ by minimizing the conditional entropy:

$$H(p_{\text{gen}}(z \mid x)) = \mathbb{E}_{x \sim p_{\text{gen}}(x)} \left[\mathbb{E}_{z \sim p_{\text{gen}}(z|x)} \left[-\log p_{\text{gen}}(z \mid x) \right] \right] \tag{B.14}$$

Optimizing this term is still problematic, because (i) we do not have access to the posterior $p_{\text{gen}}(z \mid x)$, and (ii) we cannot sample from it. Therefore, we instead minimize a variational upper bound defined by an approximate posterior $q_{\text{gen}}(z \mid x)$:

$$\begin{aligned} H(p_{\text{gen}}(z \mid x)) &= \mathbb{E}_{x \sim p_{\text{gen}}(x)} \left[\mathbb{E}_{z \sim p_{\text{gen}}(z|x)} \left[-\log q_{\text{gen}}(z \mid x) \right] - \mathbf{KL}(p_{\text{gen}}(z \mid x) \| q_{\text{gen}}(z \mid x)) \right] \\ &\leq \mathbb{E}_{x \sim p_{\text{gen}}(x)} \left[\mathbb{E}_{z \sim p_{\text{gen}}(z|x)} \left[-\log q_{\text{gen}}(z \mid x) \right] \right] \\ &= \mathcal{U}(q_{\text{gen}}). \end{aligned} \tag{B.15}$$

We can also rewrite the variational upper bound as:

$$\mathcal{U}(q_{\text{gen}}) = \mathbb{E}_{x,z \sim p_{\text{gen}}(x,z)} \left[-\log q_{\text{gen}}(z \mid x) \right] = \mathbb{E}_{z \sim p_{\text{gen}}(z)} \left[\mathbb{E}_{x \sim p_{\text{gen}}(x|z)} \left[-\log q_{\text{gen}}(z \mid x) \right] \right], \tag{B.16}$$

which can be optimized efficiently with standard back-propagation and Monte Carlo integration of the relevant expectations based on independent samples drawn from the joint $p_{\text{gen}}(x, z)$. By minimizing this upper bound on the conditional entropy $H(p_{\text{gen}}(z \mid x))$, we are effectively maximizing a variational lower bound on the entropy $H(p_{\text{gen}}(x))$.

B.5 Experiments

In this section, we verify our theoretical results empirically on several synthetic and real datasets. In particular, we evaluate whether the discriminator obtained from the entropy-regularized adversarial training can capture the density information (in the form of energy), while making sure the generator distribution matches the data distribution. For convenience, we refer to the obtained models as EGAN-Ent. Our experimental setting follows closely recommendations from [121], except in Sec. B.5.1 where we use fully-connected models (see section B.8.1 for details). [1]

B.5.1 Synthetic low-dimensional data

First, we consider three synthetic datasets in 2-dimensional space, which are drawn from the following distributions: (i) Mixture of 4 Gaussians with equal mixture weights, (ii) Mixture of 200 Gaussians arranged as two spirals (100 components each spiral), and (iii) Mixture of 2 Gaussians with highly biased mixture weights, $P(c_1) = 0.9, P(c_2) = 0.1$. We visualize the ground-truth energy of these distributions along with 100K training samples in Figure B.1. Since the data lies in 2-dimensional space, we can easily visualize both the learned generator (by drawing samples) and the discriminator for direct comparison and evaluation. We evaluate here our EGAN-Ent model using both approximations: the nearest-neighbor based approximation (EGAN-Ent-NN)

[1] For more details, please refer to `https://github.com/zihangdai/cegan_iclr2017`.

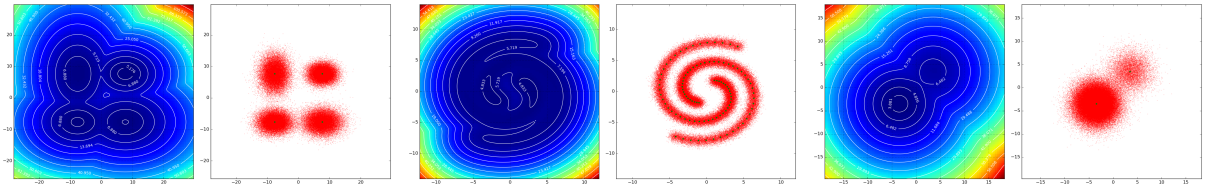
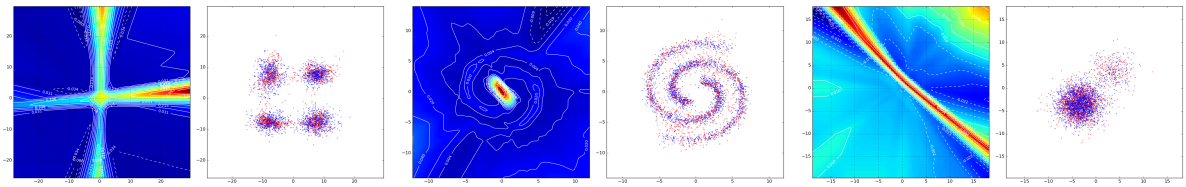
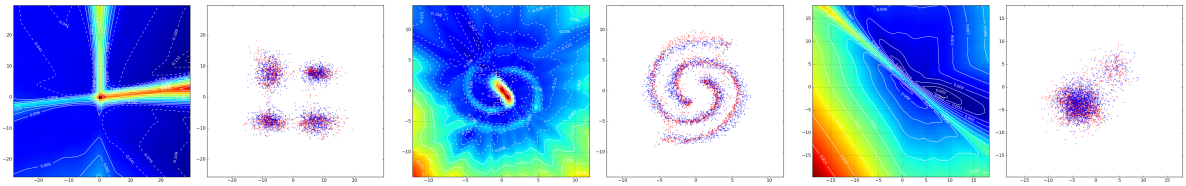


Figure B.1: True energy functions and samples from synthetic distributions. Green dots in the sample plots indicate the mean of each Gaussian component.



(a) Standard GAN



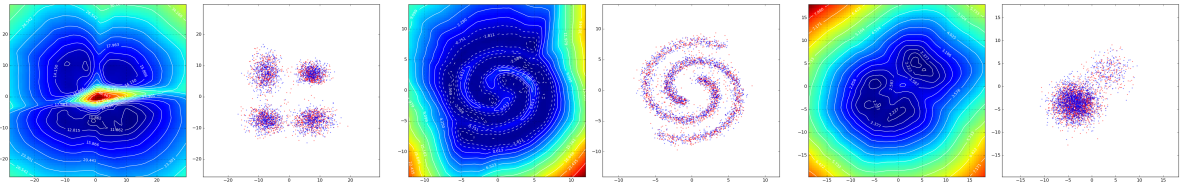
(b) Energy GAN without regularization (EGAN-Const)

Figure B.2: Learned energies and samples from baseline models whose discriminator cannot retain density information at the optimal. In the sample plots, blue dots indicate generated samples, and red dots indicate real ones.

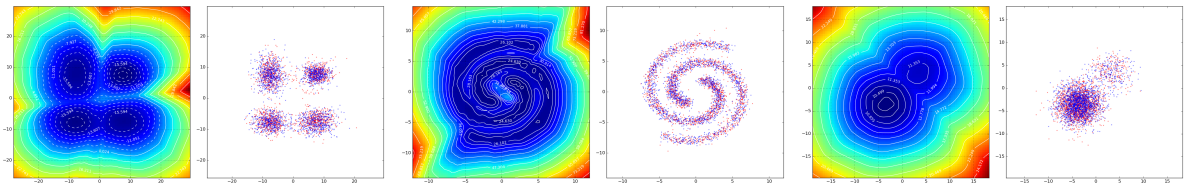
and the variational-inference based approximation (EGAN-Ent-VI), and compare them with two baselines: the original GAN and the energy based GAN with no regularization (EGAN-Const).

Experiment results are summarized in Figure B.2 for baseline models, and Figure B.3 for the proposed models. As we can see, all four models can generate perfect samples. However, for the discriminator, both GAN and EGAN-Const lead to degenerate solution, assigning flat energy inside the empirical data support. In comparison, EGAN-Ent-VI and EGAN-Ent-NN clearly capture the density information, though to different degrees. Specifically, on the equally weighted Gaussian mixture and the two-spiral mixture datasets, EGAN-Ent-NN tends to give more accurate and fine-grained solutions compared to EGAN-Ent-VI. However, on the biased weighted Gaussian mixture dataset, EGAN-Ent-VI actually fails to captures the correct mixture weights of the two modes, incorrectly assigning lower energy to the mode with lower probability (smaller weight). In contrast, EGAN-Ent-NN perfectly captures the bias in mixture weight, and obtains a contour very close to the ground truth.

To better quantify these differences, we present detailed comparison based on KL divergence in section B.8.2. What's more, the performance difference between EGAN-Ent-VI and EGAN-Ent-NN on biased Gaussian mixture reveals the limitations of the variational inference based approximation, i.e. providing inaccurate gradients. Due to space consideratiosn, we refer interested readers to the section B.8.3 for a detailed discussion.



(a) Entropy regularized Energy GAN with variational inference approximation (EGAN-Ent-VI)



(b) Entropy regularized Energy GAN with nearest neighbor approximation (EGAN-Ent-NN)

Figure B.3: Learned energies and samples from proposed models whose discriminator can retain density information at the optimal. Blue dots are generated samples, and red dots are real ones.

B.5.2 Ranking NIST digits

In this experiment, we verify that the results in synthetic datasets can translate into data with higher dimensions. While visualizing the learned energy function is not feasible in high-dimensional space, we can verify whether the learned energy function learns relative densities by inspecting the ranking of samples according to their assigned energies. We train on 28×28 images of a single handwritten digit from the NIST dataset. [2] We compare the ability of EGAN-Ent-NN with both EGAN-Const and GAN on ranking a set of 1,000 images, half of which are generated samples and the rest are real test images. Figures B.4 and B.5 show the top-100 and bottom-100 ranked images respectively for each model, after training them on digit 1. We also show in Figure B.7 the mean of all training samples, so we can get a sense of what is the most common style (highest density) of digit 1 in NIST. We can notice that all of the top-ranked images by EGAN-Ent-NN look similar to the mean sample. In addition, the lowest-ranked images are clearly different from the mean image, with either high (clockwise or counter-clockwise) rotation degrees from the mean, or an extreme thickness level. We do not see such clear distinction in other models. We provide in the section B.8.4 the ranking of the full set of images.

B.5.3 Sample quality on natural image datasets

In this last set of experiments, we evaluate the visual quality of samples generated by our model in two datasets of natural images, namely CIFAR-10 and CelebA. We employ here the variational-based approximation for entropy regularization, which can scale well to high-dimensional data. Figure B.6 shows samples generated by EGAN-Ent-VI. We can see that despite the noisy gradients provided by the variational approximation, our model is able to generate high-quality samples.

We futher validate the quality of our model's samples on CIFAR-10 using the *Inception score*

[2] `https://www.nist.gov/srd/nist-special-database-19`, which is an extended version of MNIST with an average of over 74K examples per digit.

(a) EGAN-Ent-NN

(b) EGAN-Const

(c) GAN

Figure B.4: 100 highest-ranked images out of 1000 generated and reals (bounding box) samples.

(a) EGAN-Ent-NN

(b) EGAN-Const

(c) GAN

Figure B.5: 100 lowest-ranked images out of 1000 generated and reals (bounding box) samples.

Model	Our model	Improved GAN†	EGAN-Const
Score $\pm$ std.	7.07 $\pm$ .10	6.86 $\pm$ .06	6.7447 $\pm$ 0.09

Table B.1: Inception scores on CIFAR-10. † As reported in [134] without using labeled data.

proposed by [134] [3]. Table B.1 shows the scores of our EGAN-Ent-VI, the best GAN model from [134] which uses only unlabeled data, and an EGAN-Const model which has the same architecture as our model. We notice that even without employing suggested techniques in [134], energy-based models perform quite similarly to the GAN model. Furthermore, the fact that our model scores higher than EGAN-Const highlights the importance of entropy regularization in obtaining good quality samples.

B.6 Conclusion

In this paper we have addressed a fundamental limitation in adversarial learning approaches, which is their inability of providing sensible energy estimates for samples. We proposed a novel adversarial learning formulation which results in a discriminator function that recovers the true data energy. We provided a rigorous characterization of the learned discriminator in the non-parametric setting, and proposed two methods for instantiating it in the typical parametric setting. Our experimental results verify our theoretical analysis about the discriminator properties, and

[3] Using the evaluation script released in `https://github.com/openai/improved-gan/`

(a) CIFAR-10 (b) CelebA

Figure B.6: Samples generated from our model.

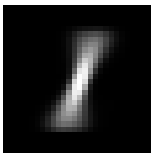


Figure B.7: mean digit

show that we can also obtain samples of state-of-the-art quality.

B.7 Supplementary materials for Section B.3

B.7.1 Optimal discriminator form under the proposed formulation

Proof of proposition 2. Refining the Lagrange $L(p_{\text{gen}}, c)$ by introducing additional dual variables for the probability constraints (the second and third), the new Lagrange function has the form

$$L(p_{\text{gen}}, c, \mu, \lambda) = K(p_{\text{gen}}) + \sum_{x \in \mathcal{X}} c(x)\Big(p_{\text{gen}}(x) - p_{\text{data}}(x)\Big) - \sum_{x \in \mathcal{X}} \mu(x) p_{\text{gen}}(x) + \lambda\Big(\sum_{x \in \mathcal{X}} p_{\text{gen}}(x) - 1\Big) \tag{B.17}$$

where $c(x) \in \mathbb{R}, \forall x$, $\mu(x) \in \mathbb{R}_+, \forall x$, and $\lambda \in \mathbb{R}$ are the dual variables. The KKT conditions for the optimal primal and dual variables are as follows

$$\begin{aligned}
\left.\frac{\partial K(p_{\text{gen}})}{\partial p_{\text{gen}}(x)}\right|_{p_{\text{gen}}=p_{\text{data}}} + c^*(x) - \mu^*(x) + \lambda^* = 0, \quad \forall x \qquad & \text{(stationarity)} \\
\mu^*(x) p^*_{\text{gen}}(x) = 0, \quad \forall x \qquad & \text{(complement slackness)} \\
\mu^*(x) \geq 0, \quad \forall x \qquad & \text{(dual feasibility)} \\
p^*_{\text{gen}}(x) \geq 0, \quad p^*_{\text{gen}}(x) = p_{\text{data}}(x), \quad \forall x \qquad & \text{(primal feasibility)} \\
\sum_{x \in \mathcal{X}} p_{\text{gen}}(x) = 1 \qquad & \text{(primal feasibility)}
\end{aligned} \tag{B.18}$$

Rearranging the conditions above, we get $p^*_{\text{gen}}(x) = p_{\text{data}}(x), \forall x \in \mathcal{X}$ as well as equation (B.4), which concludes the proof. □

B.7.2 Optimal conditions of EBGAN

In [175], the training objectives of the generator and the discriminator cannot be written as a single minimax optimization problem since the margin structure is only applied to the objective of the discriminator. In addition, the discriminator is designed to produce the mean squared reconstruction error of an auto-encoder structure. This restricted the range of the discriminator output to be non-negative, which is equivalent to posing a set constraint on the discriminator under the non-parametric setting.

Thus, to characterize the optimal generator and discriminator, we adapt the same analyzing logic used in the proof sketch of the original GAN [46]. Specifically, given a specific generator distribution p_{gen}, the optimal discriminator function given the generator distribution $c^*(x; p_{\text{gen}})$ can be derived by examining the objective of the discriminator. Then, the conditional optimal discriminator function is substituted into the training objective of p_{gen}, simplifying the "adversarial" training as a minimizing problem only w.r.t. p_{gen}, which can be well analyzed.

Firstly, given any generator distribution p_{gen}, the EBGAN training objective for the discriminator can be written as the following form

$$\begin{aligned}
c^*(x; p_{\text{gen}}) &= \operatorname*{argmax}_{c \in \mathcal{C}} \quad -\mathbb{E}_{p_{\text{gen}}} \max(0, m - c(x)) - \mathbb{E}_{p_{\text{data}}} c(x) \\
&= \operatorname*{argmax}_{c \in \mathcal{C}} \quad \mathbb{E}_{p_{\text{gen}}} \min(0, c(x) - m) - \mathbb{E}_{p_{\text{data}}} c(x)
\end{aligned} \tag{B.19}$$

where $\mathcal{C} = \{c : c(x) \geq 0, \forall x \in \mathcal{X}\}$ is the set of allowed non-negative discriminator functions. Note this set constraint comes from the fact the mean squared reconstruction error as discussed above.

Since the problem (B.19) is independent w.r.t. each x, the optimal solution can be easily derived as

$$c^*(x; p_{\text{gen}}) = \begin{cases} 0, & p_{\text{gen}}(x) < p_{\text{data}}(x) \\ m, & p_{\text{gen}}(x) > p_{\text{data}}(x) \\ \alpha_x, & p_{\text{gen}}(x) = p_{\text{data}}(x) > 0 \\ \beta_x, & p_{\text{gen}}(x) = p_{\text{data}}(x) = 0 \end{cases} \tag{B.20}$$

where $\alpha_x \in [0, m]$ is an under-determined number, a $\beta_x \in [0, \infty)$ is another under-determined non-negative real number, and the subscripts in m, α_x, β_x reflect that fact that these under-determined values can be distinct for different x.

This way, the overall training objective can be cast into a minimization problem w.r.t. p_{gen},

$$\begin{aligned} p^*_{\text{gen}} &= \operatorname*{argmin}_{p_{\text{gen}} \in \mathcal{P}} \ \mathbb{E}_{x \sim p_{\text{gen}}} c^*(x; p_{\text{gen}}) - \mathbb{E}_{x \sim p_{\text{data}}} c^*(x; p_{\text{gen}}) \\ &= \operatorname*{argmin}_{p_{\text{gen}} \in \mathcal{P}} \ \sum_{x \in \mathcal{X}} \Big[p_{\text{gen}}(x) - p_{\text{data}}(x) \Big] c^*(x; p_{\text{gen}}) \end{aligned} \tag{B.21}$$

where the second term of the first line is implicitly defined as the problem is an adversarial game between p_{gen} and c.

Proposition 3. *The global optimal of the EBGAN training objective is achieved if and only if $p_{gen} = p_{data}$. At that point, $c^*(x)$ is fully under-determined.*

Proof. The proof is established by showing contradiction.

Firstly, assume the optimal $p^*_{\text{gen}} \neq p_{\text{data}}$. Thus, there must exist a non-equal set $\mathcal{X}_{\neq} = \{x \mid p_{\text{data}}(x) \neq p^*_{\text{gen}}(x)\}$, which can be further splitted into two subsets, the greater-than set $\mathcal{X}_{>} = \{x \mid p^*_{\text{gen}}(x) > p_{\text{data}}(x)\}$, and the less-than set $\mathcal{X}_{<} = \{x \mid p^*_{\text{gen}}(x) < p_{\text{data}}(x)\}$. Similarly, we define the equal set $\mathcal{X}_{=} = \{x : p^*_{\text{gen}}(x) = p_{\text{data}}(x)\}$. Obviously, $\mathcal{X}_{>} \bigcup \mathcal{X}_{<} \bigcup \mathcal{X}_{=} = \mathcal{X}$.

Let $L(p_{\text{gen}}) = \sum_{x \in \mathcal{X}} \Big[p_{\text{gen}}(x) - p_{\text{data}}(x) \Big] c^*(x; p_{\text{gen}})$, substituting the results from equation (B.20) into (B.21), the $L(p_{\text{gen}})^*$ can be written as

$$\begin{aligned} L(p^*_{\text{gen}}) &= \sum_{x \in \mathcal{X}_{<} \bigcup \mathcal{X}_{<} \bigcup \mathcal{X}_{=}} \Big[p^*_{\text{gen}}(x) - p_{\text{data}}(x) \Big] c^*(x; p^*_{\text{gen}}) \\ &= \sum_{x \in \mathcal{X}_{<}} \Big[p^*_{\text{gen}}(x) - p_{\text{data}}(x) \Big] c^*(x; p^*_{\text{gen}}) + \sum_{x \in \mathcal{X}_{>}} \Big[p^*_{\text{gen}}(x) - p_{\text{data}}(x) \Big] c^*(x; p^*_{\text{gen}}) \\ &= m \sum_{x \in \mathcal{X}_{>}} p^*_{\text{gen}}(x) - p_{\text{data}}(x) \\ &> 0 \end{aligned} \tag{B.22}$$

However, when $p'_{\text{gen}} = p_{\text{data}}$, we have

$$L(p'_{\text{gen}}) = 0 < L(p^*_{\text{gen}}) \tag{B.23}$$

which contradicts the optimal (miminum) assumption of p^*_{gen}. Hence, the contradiction concludes that at the global optimal, $p^*_{\text{gen}} = p_{\text{data}}$. By equation (B.20), it directly follows that $c^*(x; p^*_{\text{gen}}) = \alpha_x$, which completes the proof. □

B.7.3 Analysis of adding additional training signal to GAN formulation

To show that simply adding the same training signal to GAN will not lead to the same result, it is more convenient to directly work with the formulation of f-GAN [112, equation (6)] family, which include the original GAN formulation as a special case.

Specifically, the general f-GAN formulation takes the following form

$$\max_{c} \min_{p_{\text{gen}} \in \mathcal{P}} \quad \mathbb{E}_{x \sim p_{\text{gen}}}\Big[f^{\star}(c(x))\Big] - \mathbb{E}_{x \sim p_{\text{data}}}\Big[c(x)\Big], \tag{B.24}$$

where the $f^{\star}(\cdot)$ denotes the convex conjugate [14] of the f-divergence function. The optimal condition of the discriminator can be found by taking the variation w.r.t. c, which gives the optimal discriminator

$$c^{*}(x) = f'(\frac{p_{\text{data}}(x)}{p_{\text{gen}}(x)}) \tag{B.25}$$

where $f'(\cdot)$ is the first-order derivative of $f(\cdot)$. Note that, even when we add an extra term $L(p_{\text{gen}})$ to equation (B.24), since the term $K(p_{\text{gen}})$ is a constant w.r.t. the discriminator, it does not change the result given by equation (B.25) about the optimal discriminator. As a consequence, for the optimal discriminator to retain the density information, it effectively means $p_{\text{gen}} \neq p_{\text{data}}$. Hence, there will be a contradiction if both $c^{*}(x)$ retains the density information, and the generator matches the data distribution.

Intuitively, this problem roots in the fact that f-divergence is quite "rigid" in the sense that given the $p_{\text{gen}}(x)$ it only allows one fixed point for the discriminator. In comparison, the divergence used in our proposed formulation, which is the expected cost gap, is much more flexible. By the expected cost gap itself, i.e. without the $K(p_{\text{gen}})$ term, the optimal discriminator is actually under-determined.

B.8 Supplementary Materials for section B.5

B.8.1 Experiment setting

Here, we specify the neural architectures used for experiements presented in Section B.5.

Firstly, for the Egan-Ent-VI model, we parameterize the approximate posterior distribution $q_{\text{gen}}(z \mid x)$ with a diagonal Gaussian distribution, whose mean and covariance matrix are the output of a trainable inference network, i.e.

$$\begin{aligned} q_{\text{gen}}(z \mid x) &= \mathcal{N}(\mu, \mathbf{I}\sigma^2) \\ \mu, \log \sigma &= f^{\text{infer}}(x) \end{aligned} \tag{B.26}$$

where f^{infer} denotes the inference network, and $\mathbf{I}$ is the identity matrix. Note that the Inference Network only appears in the Egan-Ent-VI model.

For experiments with the synthetic datasets, the following fully-connected feed forward neural networks are employed

- Generator: `FC(4,128)-BN-ReLU-FC(128,128)-BN-ReLU-FC(128,2)`
- Discriminator: `FC(2,128)-ReLU-FC(128,128)-ReLU-FC(128,1)`
- Inference Net: `FC(2,128)-ReLU-FC(128,128)-ReLU-FC(128,4*2)`

where `FC` and `BN` denote fully-connected layer and batch normalization layer respectively. Note that since the input noise to the generator has dimension `4`, the Inference Net output has

dimension `4*2`, where the first 4 elements correspond the inferred mean, and the last 4 elements correspond to the inferred diagonal covariance matrix in log scale.

For the handwritten digit experiment, we closely follow the DCGAN [121] architecture with the following configuration

- Generator: `FC(10,512*7*7)-BN-ReLU-DC(512,256;4c2s)-BN-ReLU-DC(256,128;4c2s)-BN-ReLU-DC(128,1;3c1s)-Sigmoid`
- Discriminator: `CV(1,64;3c1s)-BN-LRec-CV(64,128;4c2s)-BN-LRec-CV(128,256;4c2s)-BN-LRec-FC(256*7*7,1)`
- Inference Net: `CV(1,64;3c1s)-BN-LRec-CV(64,128;4c2s)-BN-LRec-CV(128,256;4c2s)-BN-LRec-FC(256*7*7,10*2)`

Here, `LRec` is the leaky rectified non-linearity recommended by Radford et al. [121]. In addition, `CV(128,256,4c2s)` denotes a convolutional layer with 128 input channels, 256 output channels, and kernel size 4 with stride 2. Similarly, `DC(256,128,4c2s)` denotes a corresponding transposed convolutional operation. Compared to the original DCGAN architecture, the discriminator under our formulation does not have the last sigmoid layer which squashes a scalar value into a probability in [0, 1].

For celebA experiment with 64×64 color images, we use the following architecture

- Generator: `FC(10,512*4*4)-BN-ReLU-DC(512,256;4c2s)-BN-ReLU-DC(256,128;4c2s)-BN-ReLU-DC(256,128;4c2s)-BN-ReLU-DC(128,3;4c2s)-Tanh`
- Discriminator: `CV(3,64;4c2s)-BN-LRec-CV(64,128;4c2s)-BN-LRec-CV(128,256;4c2s)-BN-LRec-CV(256,256;4c2s)-BN-LRec-FC(256*4*4,1)`
- Inference Net: `CV(3,64;4c2s)-BN-LRec-CV(64,128;4c2s)-BN-LRec-CV(128,256;4c2s)-BN-LRec-CV(256,256;4c2s)-BN-LRec-FC(256*4*4,10*2)`

For Cifar10 experiment, where the image size is 32×32, similar architecture is used

- Generator: `FC(10,512*4*4)-BN-ReLU-DC(512,256;4c2s)-BN-ReLU-DC(256,128;3c1s)-BN-ReLU-DC(256,128;4c2s)-BN-ReLU-DC(128,3;4c2s)-Tanh`
- Discriminator: `CV(3,64;3c1s)-BN-LRec-CV(64,128;4c2s)-BN-LRec-CV(128,256;4c2s)-BN-LRec-CV(256,256;4c2s)-BN-LRec-FC(256*4*4,1)`
- Inference Net: `CV(3,64;3c1s)-BN-LRec-CV(64,128;4c2s)-BN-LRec-CV(128,256;4c2s)-BN-LRec-CV(256,256;4c2s)-BN-LRec-FC(256*4*4,10*2)`

Given the chosen architectures, we follow Radford et al. [121] and use Adam as the optimization algorithm. For more detailed hyper-parameters, please refer to the code.

B.8.2 Quantitative comparison of different models

In order to quantify the quality of recovered distributions, we compute the pairwise KL divergence of the following four distributions:

- The real data distribution with analytic form, denoted as p_{data}
- The empirical data distribution approximated from the 100K training data, denoted as p_{emp}
- The generator distribution approximated from 100K generated data, denoted as p_{gen}
- The discriminator distribution re-normalized from the learned energy, denoted as p_{disc}

	Gaussian Mixture: $\mathrm{KL}(p_{\text{data}}\|p_{\text{emp}}) = 0.0291$, $\mathrm{KL}(p_{\text{emp}}\|p_{\text{data}}) = 0.0159$									
KL Divergence	$p_{\text{gen}}\|p_{\text{emp}}$	$p_{\text{emp}}\|p_{\text{gen}}$	$p_{\text{gen}}\|p_{\text{data}}$	$p_{\text{data}}\|p_{\text{gen}}$	$p_{\text{disc}}\|p_{\text{emp}}$	$p_{\text{emp}}\|p_{\text{disc}}$	$p_{\text{disc}}\|p_{\text{data}}$	$p_{\text{data}}\|p_{\text{disc}}$	$p_{\text{gen}}\|p_{\text{disc}}$	$p_{\text{disc}}\|p_{\text{gen}}$
GAN	0.3034	0.5024	0.2498	0.4807	6.7587	2.0648	6.2020	2.0553	2.4596	7.0895
EGAN-Const	0.2711	0.4888	0.2239	0.4735	6.7916	2.1243	6.2159	2.1149	2.5062	7.0553
EGAN-Ent-VI	0.1422	0.1367	0.0896	0.1214	0.8866	0.6532	0.7215	0.6442	0.7711	1.0638
EGAN-Ent-NN	**0.1131**	**0.1006**	**0.0621**	**0.0862**	**0.0993**	**0.1356**	**0.0901**	**0.1187**	**0.1905**	**0.1208**
	Biased Gaussian Mixture: $\mathrm{KL}(p_{\text{data}}\|p_{\text{emp}}) = 0.0273$, $\mathrm{KL}(p_{\text{emp}}\|p_{\text{data}}) = 0.0144$									
KL Divergence	$p_{\text{gen}}\|p_{\text{emp}}$	$p_{\text{emp}}\|p_{\text{gen}}$	$p_{\text{gen}}\|p_{\text{data}}$	$p_{\text{data}}\|p_{\text{gen}}$	$p_{\text{disc}}\|p_{\text{emp}}$	$p_{\text{emp}}\|p_{\text{disc}}$	$p_{\text{disc}}\|p_{\text{data}}$	$p_{\text{data}}\|p_{\text{disc}}$	$p_{\text{gen}}\|p_{\text{disc}}$	$p_{\text{disc}}\|p_{\text{gen}}$
GAN	0.0788	0.0705	0.0413	0.0547	7.1539	2.5230	6.4927	2.5018	2.5205	7.1140
EGAN-Const	0.1545	0.1649	0.1211	0.1519	7.1568	2.5269	6.4969	2.5057	2.5860	7.1995
EGAN-Ent-VI	**0.0576**	0.0668	**0.0303**	0.0518	3.9151	1.3574	2.9894	1.3365	1.4052	4.0632
EGAN-Ent-NN	0.0784	**0.0574**	0.0334	**0.0422**	**0.8505**	**0.3480**	**0.5199**	**0.3299**	**0.3250**	**0.7835**
	Two-spiral Gaussian Mixture: $\mathrm{KL}(p_{\text{data}}\|p_{\text{emp}}) = 0.3892$, $\mathrm{KL}(p_{\text{emp}}\|p_{\text{data}}) = 1.2349$									
KL Divergence	$p_{\text{gen}}\|p_{\text{emp}}$	$p_{\text{emp}}\|p_{\text{gen}}$	$p_{\text{gen}}\|p_{\text{data}}$	$p_{\text{data}}\|p_{\text{gen}}$	$p_{\text{disc}}\|p_{\text{emp}}$	$p_{\text{emp}}\|p_{\text{disc}}$	$p_{\text{disc}}\|p_{\text{data}}$	$p_{\text{data}}\|p_{\text{disc}}$	$p_{\text{gen}}\|p_{\text{disc}}$	$p_{\text{disc}}\|p_{\text{gen}}$
GAN	0.5297	0.2701	0.3758	0.7240	6.3507	1.7180	4.3818	1.0866	1.6519	5.7694
EGAN-Const	0.7473	1.0325	0.7152	1.6703	5.9930	1.5732	3.9749	0.9703	1.8380	6.0471
EGAN-Ent-VI	0.2014	0.1260	**0.4283**	**0.8399**	1.1099	0.3508	**0.3061**	**0.4037**	0.4324	0.9917
EGAN-Ent-NN	**0.1246**	**0.1147**	0.4475	1.2435	**0.1036**	**0.0857**	0.4086	0.7917	**0.1365**	**0.1686**

Table B.2: Pairwise KL divergence between distributions. Bold face indicate the lowest divergence within group.

Since the synthetic datasets are two dimensional, we approximate both the empirical data distribution and the generator distribution using the simple histogram estimation. Specifically, we divide the canvas into a 100-by-100 grid, and assign each sample into its nearest grid cell based on euclidean distance. Then, we normalize the number of samples in each cell into a proper distribution. When recovering the discriminator distribution from the learned energy, we assume that $\mu^*(x) = 0$ (i.e. infinite data support), and discretize the distribution into the same grid cells

$$p_{\text{disc}}(x) = \frac{\exp(-c^*(x))}{\sum_{x' \in \text{Grid}} \exp(-c^*(x'))}, \forall x \in \text{Grid}$$

Based on these approximation, Table B.2 summarizes the results. For all measures related to the discriminator distribution, EGAN-Ent-VI and EGAN-Ent-NN significantly outperform the other two baseline models, which matches our visual assessment in Figure B.2 and B.3. Meanwhile, the generator distributions learned from our proposed framework also achieve relatively lower divergence to both the empirical data distribution and the true data distribution.

B.8.3 Comparison of the entropy (gradient) approximation methods

In order to understand the performance difference between EGAN-Ent-VI and EGAN-Ent-NN, we analyze the quality of the entropy gradient approximation during training. To do that, we visualize some detailed training information in Figure B.8.

As we can see in figure B.8a, the viarational entropy gradient approximation w.r.t. samples is not accurate:

- It is inaccurate in terms of gradient direction. Ideally, the direction of the entropy gradient should be pointing from the center of its closest mode towards the surroundings, with

the direction orthogonal to the implicit contour in Fig. (1,2). However, the direction of gradients in the Fig. (2,3) does not match this.

- It is inaccurate in magnitude. As we can see, the entropy approximation gradient (Fig. (2,3)) has much larger norm than the discriminator gradient (Fig. (2,2)). As a result, the total gradient (Fig. (2,4)) is fully dominated by the entropy approximation gradient. Thus, it usually takes much longer for the generator to learn to generate rare samples, and the training also proceeds much slower compared to the nearest neighbor based approximation.

In comparison, the nearest neighbor based gradient approximation is much more accurate as shown in B.8b. As a result, it leads to more accurate energy contour, as well as faster training. What's more, from Figure B.8b Fig. (2,4), we can see the entropy gradient does have the cancel-out effect on the discriminator gradient, which again matches our theory.

B.8.4 Ranking NIST Digits

Figure B.9 shows the ranking of all 1000 generated and real images (from the test set) for three models: EGAN-Ent-NN, EGAN-Const, and GAN. We can clearly notice that in EGAN-Ent-NN the top-ranked digits look very similar to the mean digit. From the upper-left corner to the lower-right corner, the transition trend is: the rotation degree increases, and the digits become increasingly thick or thin compared to the mean. In addition, samples in the last few rows do diverge away from the mean image: either highly diagonal to the right or left, or have different shape: very thin or thick, or typewriter script. Other models are not able to achieve a similar clear distinction for high versus low probability images. Finally, we consistently observe the same trend in modeling other digits, which are not shown in this paper due to space constraint.

B.8.5 Classifier performance as a proxy measure

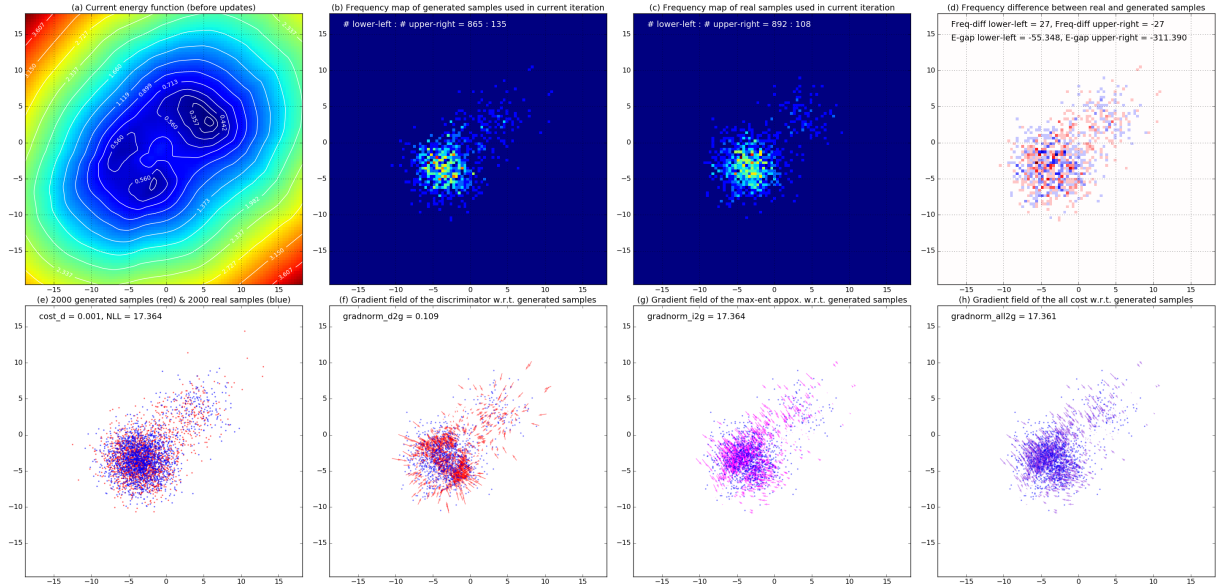
As mentioned in Section B.5, evaluating the proposed formulation quantitatively on high-dimensional data is extremely challenging. Here, in order to provide more quantitative intuitions on the learned discriminator at convergence, we adopt a proxy measure. Specifically, we take the last-layer activation of the converged discriminator network as **fixed** pretrained feature, and build a linear classifier upon it. Hypothetically, if the discriminator does not degenerate, the extracted last-layer feature should maintain more information about the data points, especially compared to features from degenerated discriminators. Following this idea, we first train EGAN-Ent-NN, EGAN-Const, and GAN on the MNIST till convergence, and then extract the last-layer activation from their discriminator networks as fixed feature input. Based on fixed feature, a randomly initialized linear classifier is trained to do classification on MNIST. Based on 10 runs (with different initialization) of each of the three models, the test classification performance is summarized in Table B.3. For comparison purpose, we also include a baseline where the input features are extracted from a discriminator network with random weights.

Based on the proxy measure, EGAN-Ent-NN seems to maintain more information of data, which suggests that the discriminator from our proposed formulation is more informative. Despite the positive result, it is important to point out that maintaining information about categories does not necessarily mean maintaining information about the energy (density). Thus, this proxy

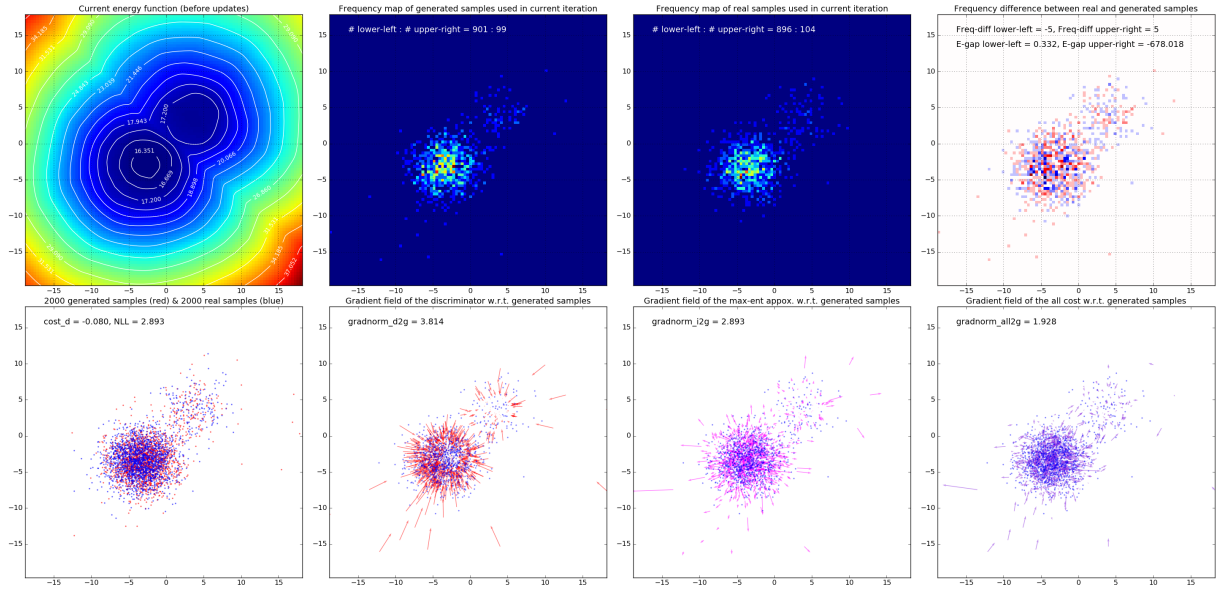
Test error (%)	EGAN-Ent-NN	EGAN-Const	GAN	Random
Min	**1.160**	1.280	1.220	3.260
Mean	**1.190**	1.338	1.259	3.409
Std.	0.024	0.044	0.032	0.124

Table B.3: Test performance of linear classifiers based on last-layer discriminator features.

measure should be understood cautiously.

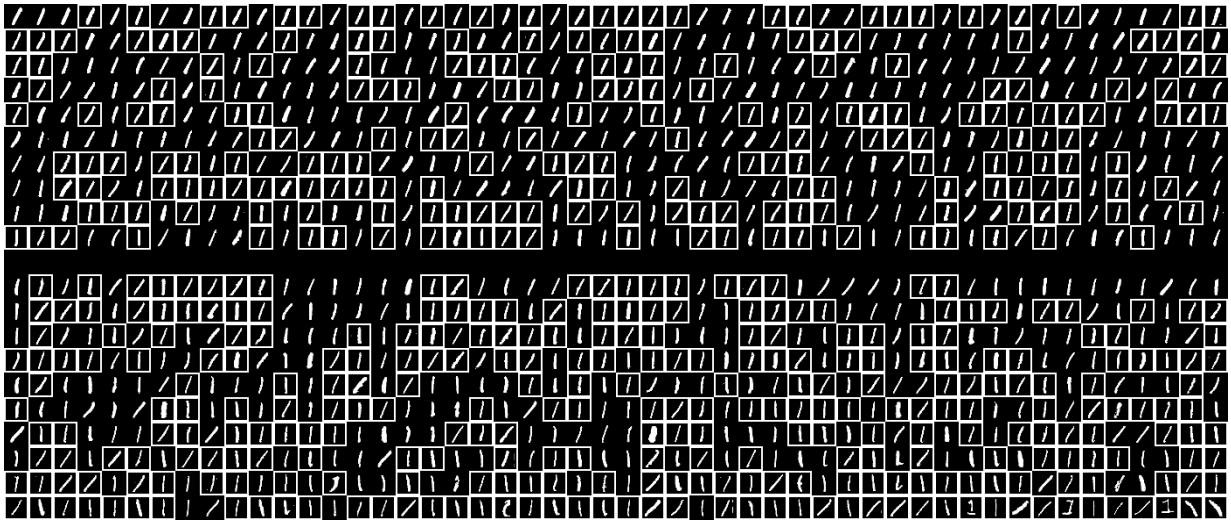


(a) Training details under variational inference entropy approximation

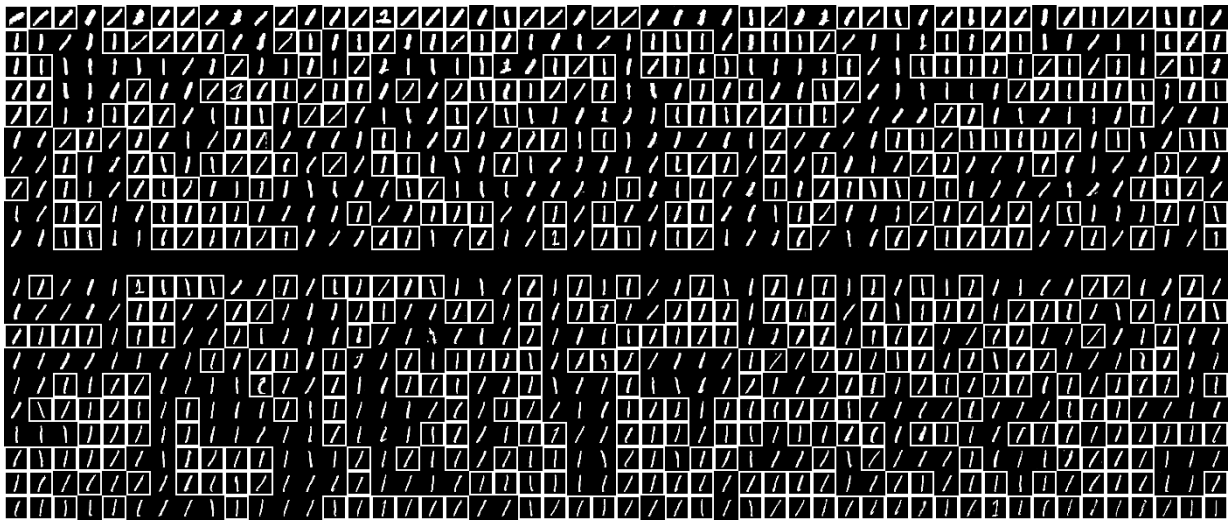


(b) Training details under nearest neighbor entropy approximation

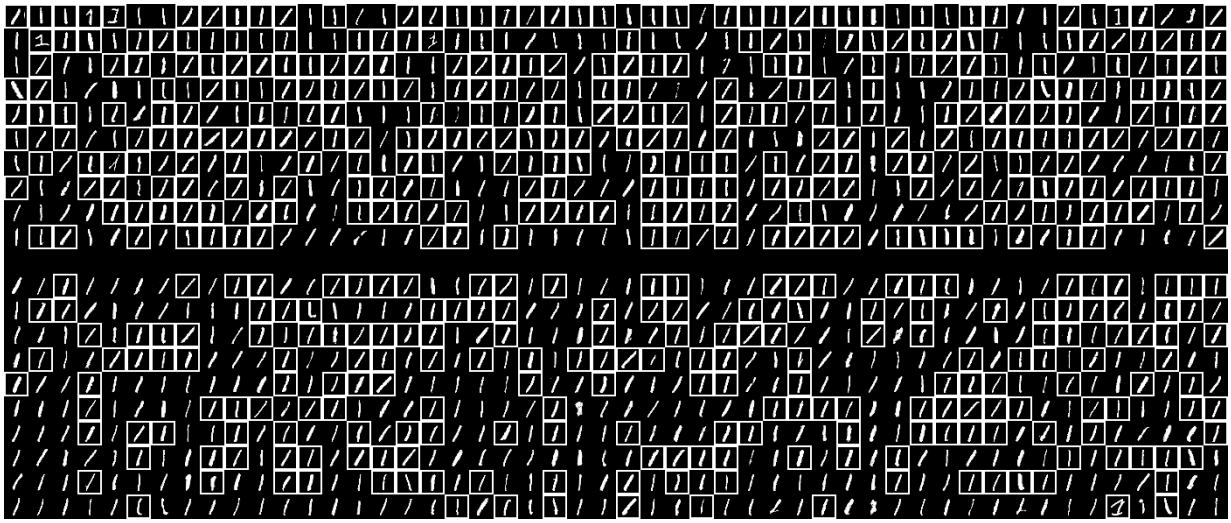
Figure B.8: For convenience, we will use Fig. (i,j) to refer to the subplot in row i, column j. Fig. (1,1): current energy plot. Fig. (1,2): frequency map of generated samples in the current batch. Fig. (1,3): frequency map of real samples in the current batch. Fig-(1,4): frequency difference between real and generated samples. Fig. (2,1) comparison between more generated from current model and real sample. Fig. (2,2): the discriminator gradient w.r.t. each training sample. Fig. (2,3): the entropy gradient w.r.t. each training samples. Fig. (2,4): all gradient (discriminator + entropy) w.r.t. each training sample.



(a) EGAN-Ent-NN



(b) EGAN-Const



(c) GAN

Figure B.9: 1000 generated and test images (bounding box) ranked according their assigned energies.

Appendix C

Good Semi-supervised Learning that Requires a Bad GAN

C.1 Introduction

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 [11, 177] were extended to deep neural networks [72, 161, 168], which involves applying convolutional neural networks [86] and feature learning techniques to graphs so that the underlying manifold structure can be exploited. [129] 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 [71, 94] by maximizing the variational lower bound of the unlabeled data log-likelihood.

Recently, generative adversarial networks (GANs) [46] 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 [134] 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 [38, 150].

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, [134] 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 [150], 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.

Empirically, our approach substantially improves over vanilla feature matching GANs, and obtains new state-of-the-art results on MNIST, SVHN, and CIFAR-10 when all methods are compared under the same discriminator architecture. Our results on MNIST and SVHN also represent state-of-the-art amongst all single-model results.

C.2 Related Work

Besides the adversarial feature matching approach [134], several previous works have incorporated the idea of adversarial training in semi-supervised learning. Notably, [144] 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, [105, 107] 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, [170] 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 [89] 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, [71] 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. [94] 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 [129] take advantage of the local (layerwise) denoising auto-encoding criterion, and create a more informative unsupervised signal through lateral connection.

C.3 Theoretical Analysis

Given a labeled set $\mathcal{L} = \{(x, y)\}$, 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 [134]:

$$\max_D \mathbb{E}_{x,y\sim\mathcal{L}} \log P_D(y|x, y \leq K) + \mathbb{E}_{x\sim p} \log P_D(y \leq K|x) + \mathbb{E}_{x\sim p_G} \log P_D(K + 1|x), \quad \text{(C.1)}$$

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 $f(x)$ be a nonlinear vector-valued function, and w_k be the weight vector for class k. As a standard setting in previous work [38, 134], the discriminator D is defined as $P_D(k|x) = \frac{\exp(w_k^\top f(x))}{\sum_{k'=1}^{K+1} \exp(w_{k'}^\top f(x))}$. Since this is a form of over-parameterization, w_{K+1} is fixed as a zero vector [134]. We next discuss the choices of different possible G's.

C.3.1 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 4. *If $p_G = p$, and D has infinite capacity, then for any optimal solution $D = (w, f)$ of the following supervised objective,*

$$\max_D \mathbb{E}_{x,y\sim\mathcal{L}} \log P_D(y|x, y \leq K), \quad \text{(C.2)}$$

there exists $D^* = (w^*, f^*)$ such that D^* maximizes Eq. (C.1) and that for all x, $P_D(y|x, y \leq K) = P_{D^*}(y|x, y \leq K)$.

The proof is provided in the supplementary material. Proposition 4 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.

C.3.2 Complement Generator

The function f maps data points in the input space to the feature space. Let $p_k(f)$ be the density of the data points of class k in the feature space. Given a threshold ϵ_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., $\forall i \neq j, \operatorname{argmax}_f p_i(f) \cap \operatorname{argmax}_f p_j(f) = \emptyset$, this assumption can always be satisfied by tuning the thresholds ϵ_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 $\mathcal{B}$. If the support F_G of a generator G in the feature space is a relative complement set in $\mathcal{B}$, i.e., $F_G = \mathcal{B} - \cup_{k=1}^K F_k$, we call G a complement generator. The reason why we utilize a bounded $\mathcal{B}$ to define the complement is presented in the supplementary material. Note that the definition of complement generator implies that G is a function of f. By treating G as function of f, theoretically D can optimize the original objective function in Eq. (C.1).

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

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

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., $\log \sum_k \exp w_k^\top f(x) > 0$ for $x \in \mathcal{U}$. However, condition (3) goes beyond this and assumes $\max_k w_k^\top f(x) > 0$. We discuss this issue in detail in the supplementary material and argue that these assumptions are reasonable. Moreover, in Section C.5, our approach addresses this issue

explicitly by adding a conditional entropy term to the discriminator objective to enforce condition (3).

Lemma 2. *Suppose for all k, the L2-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 $f'_G \in \mathcal{G}$ such that $\|f_G - f'_G\|_2 \leq \epsilon$. With the conditions in Assumption 1, for all $k \leq K$, we have $w_k^\top f_G < C\epsilon$.*

Corollary 2. *When unlimited generated data samples are available, with the conditions in Lemma 2, we have $\lim_{|\mathcal{G}|\to\infty} w_k^\top f_G \leq 0$.*

See the supplementary material for the proof.

Proposition 5. *Given the conditions in Corollary 2, 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, $\mathcal{B}$ is a convex set and $F_G = \mathcal{B} - \cup_k F_k$, 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 2, 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 5 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 ϵ_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 [11, 177] which also leverage the low-density boundary assumption.

With our theoretical analysis, we can now answer the questions raised in Section C.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 5). Second, good semi-supervised learning indeed requires a bad generator because a perfect generator is not able to improve the generalization performance (Proposition 4).

C.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) [134] 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. C.1.

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

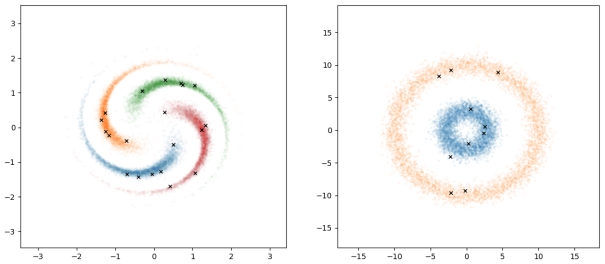


Figure C.1: Labeled and unlabeled data are denoted by cross and point respectively, and different colors indicate classes.

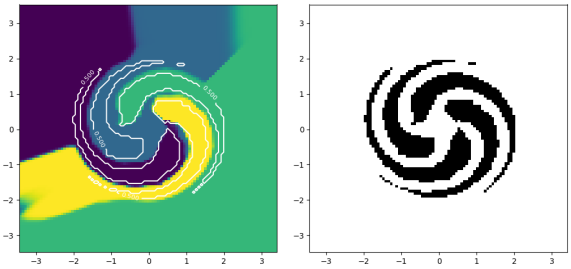


Figure C.2: Left: Classification decision boundary, where the white line indicates true-fake boundary; Right: True-Fake decision boundary

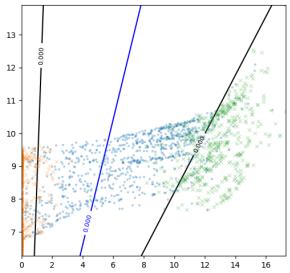


Figure C.3: Feature space at convergence

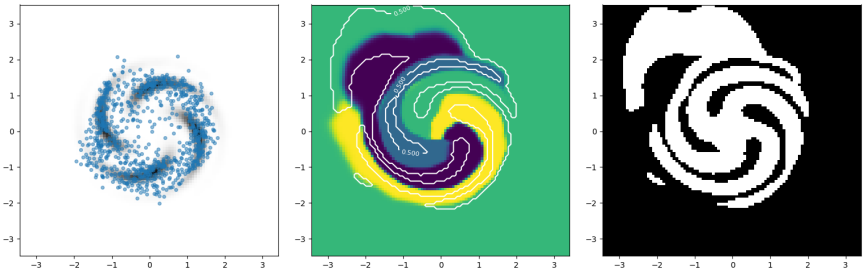


Figure C.4: Left: Blue points are generated data, and the black shadow indicates unlabeled data. Middle and right can be interpreted as above.

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. C.2. 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. C.3. 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. C.4. 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. C.3).
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 along as the feature mean matches. Actually, we do see such collapsing phenomenon in high-dimensional experiments when FM is used (see Fig. C.5a and Fig. C.5c) 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.

C.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 C.3.2. Our approach aims to explicitly address the above drawbacks.

Following prior work [46, 134], we employ a GAN-like implicit generator. We first sample a latent variable z from a uniform distribution $\mathcal{U}(0,1)$ for each dimension, and then apply a deep convolutional network to transform z to a sample x.

C.5.1 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., $\mathcal{H}(p_G(x))$, using a technique derived from an information theoretical perspective and relies on variational inference (VI). Specially, let $\mathcal{Z}$ be the latent variable space, and $\mathcal{X}$ be the input space. We introduce an additional encoder, $q : \mathcal{X} \mapsto \mathcal{Z}$, to define a variational upper bound of the negative entropy [33], $-\mathcal{H}(p_G(x)) \leq -\mathbb{E}_{x,z\sim p_G} \log q(z|x) = L_{\text{VI}}$. Hence, minimizing the upper bound L_{VI} effectively increases the generator entropy. In our implementation, we formulate q as a diagonal Gaussian with bounded variance, i.e. $q(z|x) = \mathcal{N}(\mu(x), \sigma^2(x))$, with $0 < \sigma(x) < \theta$, where $\mu(\cdot)$ and $\sigma(\cdot)$ are neural networks, and θ 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, $L_{\text{PT}} = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j\neq i} \left(\frac{f(x_i)^\top f(x_j)}{\|f(x_i)\|\|f(x_j)\|} \right)^2$, 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.

C.5.2 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 C.5.1, 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 ϵ, we minimize the following term as part of our objective:

$$\mathbb{E}_{x\sim p_G} \log p(x) \mathbb{I}[p(x) > \epsilon] \tag{C.3}$$

where $\mathbb{I}[\cdot]$ is an indicator function. Using a threshold ϵ, 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++ [**?**] model. The PixelCNN++ model is used to estimate the density $p(x)$ in Eq. (C.3). The model is pretrained on the training set, and fixed during semi-supervised training.

C.5.3 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:

$$\min_G \quad -\mathcal{H}(p_G) + \mathbb{E}_{x\sim p_G} \log p(x) \mathbb{I}[p(x) > \epsilon] + \|\mathbb{E}_{x\sim p_G} f(x) - \mathbb{E}_{x\sim \mathcal{U}} f(x)\|^2. \tag{C.4}$$

This objective is closely related to the idea of complement generator discussed in Section C.3. To see that, let's first define a target complement distribution in the input space as follows

$$p^*(x) = \begin{cases} \frac{1}{Z}\frac{1}{p(x)} & \text{if } p(x) > \epsilon \text{ and } x \in \mathcal{B}_x \\ C & \text{if } p(x) \leq \epsilon \text{ and } x \in \mathcal{B}_x, \end{cases}$$

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

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

The form of the KLD immediately reveals the aforementioned connection. Firstly, the KLD shares two exactly the same terms with the generator objective (C.4). Secondly, while $p^*(x)$ is only defined in $\mathcal{B}_x$, there is not such a hard constraint on $p_G(x)$. However, the feature matching term in Eq. (C.4) can be seen as softly enforcing this constraint by bringing generated samples "close" to the true data (Cf. Section C.4). Moreover, because the identity function $\mathbb{I}[\cdot]$ 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 (C.4) 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.

C.5.4 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., $\max_{k=1}^K w_k^\top f(x) > 0$. However, the objective function of the discriminator in [134] 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,

$$\begin{aligned} \max_D \quad & \mathbb{E}_{x,y\sim\mathcal{L}} \log p_D(y|x, y \leq K) + \mathbb{E}_{x\sim\mathcal{U}} \log p_D(y \leq K|x) + \\ & \mathbb{E}_{x\sim p_G} \log p_D(K+1|x) + \mathbb{E}_{x\sim\mathcal{U}} \sum_{k=1}^K p_D(k|x) \log p_D(k|x). \end{aligned} \tag{C.5}$$

By optimizing Eq. (C.5), 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 [107, 144] 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. (C.4) and Eq. (C.5) 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.

Methods	MNIST (# errors)	SVHN (% errors)	CIFAR-10 (% errors)
CatGAN [144]	191 ± 10	-	19.58 ± 0.46
SDGM [94]	132 ± 7	16.61 ± 0.24	-
Ladder network [129]	106 ± 37	-	20.40 ± 0.47
ADGM [94]	96 ± 2	22.86	-
FM [134] *	93 ± 6.5	8.11 ± 1.3	18.63 ± 2.32
ALI [37]	-	7.42 ± 0.65	17.99 ± 1.62
VAT small [107] *	136	6.83	14.87
Our best model *	$\mathbf{79.5 \pm 9.8}$	$\mathbf{4.25 \pm 0.03}$	$\mathbf{14.41 \pm 0.30}$
Triple GAN [89] *‡	91 ± 58	5.77 ± 0.17	16.99 ± 0.36
Π model [83] †‡	-	5.43 ± 0.25	16.55 ± 0.29
VAT+EntMin+Large [107]†	-	4.28	13.15

Table C.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.

(a) FM on SVHN (b) Ours on SVHN (c) FM on CIFAR (d) Ours on CIFAR

Figure C.5: Comparing images generated by FM and our model. FM generates collapsed samples, while our model generates diverse "bad" samples.

C.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 ϵ in Eq. (C.4). We add instance noise to the input of the discriminator [4, 141], and use spatial dropout [148] to obtain faster convergence. Except for these two modifications, we use the same neural network architecture as in [134]. For fair comparison, we also report the performance of our FM implementation with the aforementioned differences.

C.6.1 Main Results

We compare the the results of our best model with state-of-the-art methods on the benchmarks in Table C.1. Our proposed methods consistently improve the performance upon feature match-

Setting	Error	Setting	Error
MNIST FM	85.0 ± 11.7	CIFAR FM	16.14
MNIST FM+VI	86.5 ± 10.6	CIFAR FM+VI	14.41
MNIST FM+LD	79.5 ± 9.8	CIFAR FM+VI+Ent	15.82
MNIST FM+LD+Ent	89.2 ± 10.5		

Setting	Error	Setting	Max log-p
SVHN FM	6.83	MNIST FM	-297
SVHN FM+VI	5.29	MNIST FM+LD	-659
SVHN FM+PT	4.63	SVHN FM+PT+Ent	-5809
SVHN FM+PT+Ent	4.25	SVHN FM+PT+LD+Ent	-5919
SVHN FM+PT+LD+Ent	4.19	SVHN 10-quant	-5622

Setting ϵ as q-th centile	$q = 2$	$q = 10$	$q = 20$	$q = 100$
Error on MNIST	77.7 ± 6.1	79.5 ± 9.8	80.1 ± 9.6	85.0 ± 11.7

Table C.2: Ablation study. *FM* is feature matching. *LD* is the low-density enforcement term in Eq. (C.3). *VI* and *PT* are two entropy maximization methods described in Section C.5.1. *Ent* means the conditional entropy term in Eq. (C.5). *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.

ing. 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 single-model results, even when compared with methods using self-ensembling and large discriminator architectures. Finally, note that because our method is actually orthogonal to VAT [107], combining VAT with our presented approach should yield further performance improvement in practice.

C.6.2 Ablation Study

We report the results of ablation study in Table C.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 C.5.1) improve the performance on SVHN and CIFAR by up to 2.2 points in terms of error rate. Moreover, as shown in Fig C.5, 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 [134] 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 C.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. C.5a). Quantitatively, SVHN samples are assigned very low log probabilities (Cf. Table C.2). As expected, LD has a negligible effect on the performance for SVHN. Moreover, the "max log-p" column in Table C.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 ϵ in Eq. (C.4). As shown at the bottom of Table C.2, reducing ϵ clearly leads to better performance, which further justifies our analysis in Sections C.4 and C.3 that off-manifold samples are favorable.

C.6.3 Generated Samples

We compare the generated samples of FM and our approach in Fig. C.5. The FM images in Fig. C.5c are extracted from previous work [134]. While collapsing is widely observed in FM samples, our model generates diverse "bad" images, which is consistent with our analysis.

C.7 Conclusions

In this work, we present a semi-supervised learning framework that uses generated data to boost task performance. Under this framework, we characterize the properties of various generators and theoretically prove that a complementary (i.e. bad) generator improves generalization. Empirically our proposed method improves the performance of image classification on several benchmark datasets.

C.8 Appendix

C.8.1 Proof of Proposition 4

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

$$\exp(w_k^{*\top} f^*(x)) = \frac{\exp(w_k^\top f(x))}{\sum_{k'} \exp(w_{k'}^\top f(x))} \tag{C.6}$$

For all x,

$$P_{D^*}(y|x, y \leq K) = \frac{\exp(w_k^{*\top} f^*(x))}{\sum_{k'} \exp(w_{k'}^{*\top} f^*(x))} = \frac{\exp(w_k^\top f(x))}{\sum_{k'} \exp(w_{k'}^\top f(x))} = P_D(y|x, y \leq K)$$

Let L_D be the supervised objective in Eq. (C.1). Since $p = p_G$, the objective in Eq. (C.1) can be written as

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

Given Eq. (C.6), we have

$$P_{D^*}(K+1|x) = \frac{1}{1 + \sum_k \exp w_k^{*\top} f^*(x)} = \frac{1}{2}$$

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. □

C.8.2 On the Feature Space Bound Assumption

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

The bounded $\mathcal{B}$ 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 3. *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 $\mathbb{R}^d - \cup_{k=1}^K F_k$, where d is the feature space dimension, then by Assumption 1, we have $S = F_G$. Since F_G is the complement set of $\cup_{k=1}^K F_k$ and F_k's are disjoint, F_G is a non-convex set, if $K \geq 2$. However, by Lemma 3, F_G is convex, leading to contradiction. We therefore define the complement generator using a bound $\mathcal{B}$.

C.8.3 The Reasonableness of Assumption 1

Here, we justify the proposed Assumption 1.

Classification correctness on $\mathcal{L}$ For (1), it assumes the correctness of classification on labeled data $\mathcal{L}$. This only requires the transformation $f(x)$ 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 $|\mathcal{L}|$ is quite limited, this assumption is usually reasonable.

True-Fake correctness on $\mathcal{G}$ 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 $w_{K+1}^\top f = 0$, and the assumption thus reduces to $w_{K+1}^\top f(x) > \max_{k=1}^K w_k^\top f(x)$. For this part to hold, again we essentially require a transformation $f(x)$ with high enough capacity to distinguish true and fake data, which is a standard assumption made in GAN literature.

Strong true-fake belief on $\mathcal{U}$ 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 $\max_{k=1}^K w_k^\top f(x) > 0 = w_{K+1}^\top f(x)$ is a *sufficient* (but not necessary) condition for x being classified as a true data point. Instead, the actual necessary condition is that $\log \sum_{k=1}^K \exp(w_k^\top f(x)) \geq w_{K+1}^\top f(x) = 0$. Thus, it means the condition (3) might be violated.
- However, using the relationship $\log \sum_{k=1}^K \exp(w_k^\top f(x)) \leq \log K \max_{k=1}^K \exp(w_k^\top f(x))$, to guarantee the necessary condition $\log \sum_{k=1}^K \exp(w_k^\top f(x)) \geq 0$, we must have

$$\begin{aligned} & \log K \max_{k=1}^K \exp(w_k^\top f(x)) \geq 0 \\ \implies \quad & \max_{k=1}^K w_k^\top f(x) \geq \log 1/K \end{aligned}$$

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

$$\log 1/K \leq \max_{k=1}^K w_k^\top f(x) \leq 0$$

  Note that this is a very small interval for the logit $w_k^\top f(x)$, 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 5 always hold for regions inside $\mathcal{U}$ where $\max_{k=1}^K w_k^\top f(x) > 0$. 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 $\mathcal{G}$, we calculate the ratio after each epoch

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

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

$$\frac{\sum_{x \sim p_G} \mathbb{I}[\max_i w_i^T f(x) < 0]}{|\mathcal{T}|}$$

The plot is presented in Fig. C.6. 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.

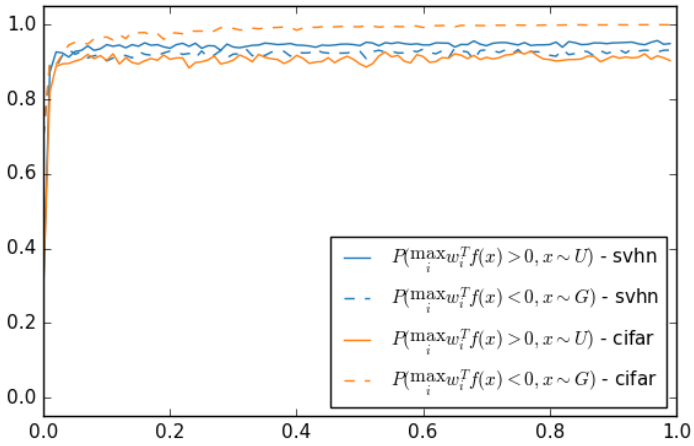


Figure C.6: Percentage of the test samples that satisfy the assumption under our best model.

C.8.4 Proof of Lemma 2

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

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

□

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004. 82

[2] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. *arXiv preprint arXiv:1808.04444*, 2018. 4, 11, 12, 16, 17

[3] Anonymous. Bam! born-again multi-task networks for natural language understanding. anonymous preprint under review, 2018. 30

[4] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *NIPS 2016 Workshop on Adversarial Training. In review for ICLR*, volume 2016, 2017. 112

[5] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*, 2018. 15, 16, 17, 18

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 10

[7] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018. 16

[8] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 10

[9] Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014. 73

[10] Loris Bazzani, Hugo Larochelle, and Lorenzo Torresani. Recurrent mixture density network for spatiotemporal visual attention. *arXiv preprint arXiv:1603.08199*, 2016. 73

[11] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR*, 7:2399–2434, 2006. 103, 107

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

49, 63

[13] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003. 67

[14] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 84, 95

[15] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 3, 8

[16] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020. 60

[17] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Accurate and conservative estimates of mrf log-likelihood using reverse annealing. In *Artificial Intelligence and Statistics*, pages 102–110, 2015. 7

[18] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015. 7

[19] Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language gans falling short. *arXiv preprint arXiv:1811.02549*, 2018. 8

[20] Jamie Callan, Mark Hoy, Changkuk Yoo, and Le Zhao. Clueweb09 data set, 2009. 27

[21] Nicolas Carion, F. Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander M Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *ArXiv*, abs/2005.12872, 2020. 44

[22] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013. 15, 17, 68

[23] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. Adabert: Task-adaptive bert compression with differentiable neural architecture search. *arXiv preprint arXiv:2001.04246*, 2020. 33

[24] Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. 60

[25] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020. 60

[26] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015. 73

[27] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016. 16, 73

[28] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020. 33, 35, 39, 42, 43

[29] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016. 16

[30] Common Crawl. Common crawl. *URl: http://http://commoncrawl. org*, 2019. 27

[31] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087, 2015. 9, 19

[32] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 126–134. ACM, 2018. 30

[33] Zihang Dai, Amjad Almahairi, Philip Bachman, Eduard Hovy, and Aaron Courville. Calibrating energy-based generative adversarial networks. *arXiv preprint arXiv:1702.01691*, 2017. 110

[34] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019. 20, 24, 25, 39, 44

[35] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. *arXiv preprint arXiv:1612.08083*, 2016. 16, 17

[36] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 2, 4, 9, 10, 12, 19, 20, 25, 27, 28, 30, 33, 34, 38, 44

[37] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016. 58, 112

[38] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016. 103, 105

[39] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*, 2018. 3

[40] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013. 73

[41] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems*, pages 2199–2207, 2016. 73

[42] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016. xii, 52, 69, 75

[43] Zhe Gan, Chunyuan Li, Ricardo Henao, David E Carlson, and Lawrence Carin. Deep temporal sigmoid belief networks for sequence modeling. In *Advances in Neural Information Processing Systems*, pages 2467–2475, 2015. 73

[44] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015. 8

[45] John J Godfrey and Edward Holliman. Switchboard-1 release 2. *Linguistic Data Consortium, Philadelphia*, 1997. 69

[46] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2, 8, 81, 82, 93, 103, 109

[47] Saurabh Goyal, Anamitra Roy Choudhary, Venkatesan Chakaravarthy, Saurabh ManishRaje, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference for classification tasks. *arXiv preprint arXiv:2001.08950*, 2020. 43

[48] Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. Efficient softmax approximation for gpus. *arXiv preprint arXiv:1609.04309*, 2016. 15

[49] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016. 16, 52, 69, 70

[50] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. 3, 73

[51] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. 13

[52] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015. 73

[53] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 55–64. ACM, 2016. 30

[54] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 16

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[56] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012. 7

[57] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016. 82, 84

[58] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[59] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001. 3, 10

[60] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 328–339, 2018. 9, 29

[61] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, and Douglas Eck. An improved relative self-attention mechanism for transformer with application to music generation. *arXiv preprint arXiv:1809.04281*, 2018. 14

[62] Brian Hutchinson, Mari Ostendorf, and Maryam Fazel. Low rank language models for small training sets. *IEEE Signal Processing Letters*, 18(9):489–492, 2011. 72

[63] Brian Hutchinson, Mari Ostendorf, and Maryam Fazel. A sparse plus low rank maximum entropy language model. In *INTERSPEECH*, pages 1676–1679, 2012. 72

[64] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016. 17, 49, 52, 63, 69, 70

[65] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991. 73

[66] Rie Johnson and Tong Zhang. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 562–570, 2017. 29

[67] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016. 8, 17, 49

[68] Taesup Kim and Yoshua Bengio. Deep directed generative models with energy-based probability estimation. *arXiv preprint arXiv:1606.03439*, 2016. 82, 85

[69] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016. 52, 69

[70] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 2, 7, 73

[71] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014. 103, 104

[72] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 103

[73] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995. 66, 67

[74] Bryon Knol. cmix v13. `http://www.byronknoll.com/cmix.html`, 2017. 16

[75] Vid Kocijan, Ana-Maria Cretu, Oana-Maria Camburu, Yordan Yordanov, and Thomas Lukasiewicz. A surprisingly robust trick for winograd schema challenge. *arXiv preprint arXiv:1905.06290*, 2019. 29

[76] Lingpeng Kong, Cyprien de Masson d'Autume, Wang Ling, Lei Yu, Zihang Dai, and Dani Yogatama. A mutual information maximization perspective of language representation learning. *arXiv preprint arXiv:1910.08350*, 2019. 33

[77] Ben Krause, Liang Lu, Iain Murray, and Steve Renals. Multiplicative lstm for sequence modelling. *arXiv preprint arXiv:1609.07959*, 2016. 16

[78] Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. Dynamic evaluation of neural sequence models. *arXiv preprint arXiv:1709.07432*, 2017. xii, xiii, 52, 63, 67, 68, 69, 70, 75

[79] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for lstm networks. *arXiv preprint arXiv:1703.10722*, 2017. 17

[80] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018. 27

[81] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017. 27

[82] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017. 39

[83] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016. 58, 112

[84] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019. 33, 43

[85] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011. 8

[86] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 103

[87] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014. 73

[88] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019. 33

[89] Chongxuan Li, Kun Xu, Jun Zhu, and Bo Zhang. Triple generative adversarial nets. *arXiv preprint arXiv:1703.02291*, 2017. 58, 104, 112

[90] Rui Lin, Shujie Liu, Muyun Yang, Mu Li, Ming Zhou, and Sheng Li. Hierarchical recurrent neural network for document modeling. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 899–907, 2015. 43

[91] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 17

[92] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*, 2019. 29, 30, 33

[93] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. 33, 42, 43

[94] Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016. 58, 103, 105, 112

[95] Matt Mahoney. Large text compression benchmark, 2011. 15, 78

[96] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305, 2017. 19

[97] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017. 49, 52, 63, 67, 68, 69, 70

[98] Gábor Melis, Charles Blundell, Tomáš Kočiskỳ, Karl Moritz Hermann, Chris Dyer, and Phil Blunsom. Pushing the bounds of dropout. *arXiv preprint arXiv:1805.09208*, 2018. 17

[99] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016. 15, 52, 68, 69

[100] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017. xii, 17, 18, 49, 52, 63, 67, 68, 69, 70, 71, 75

[101] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*, 2018. 16

[102] Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. *SLT*, 12:234–239, 2012. 15, 52, 66, 69

[103] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010. xiii, 13, 49, 63, 68, 80

[104] Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012. 78

[105] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*, 2015. 104

[106] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. *arXiv preprint arXiv:1605.07725*, 2016. 29

[107] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *arXiv preprint arXiv:1704.03976*, 2017. 58, 104, 111, 112, 113

[108] Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pages 641–648. ACM, 2007. 49, 63

[109] Asier Mujika, Florian Meier, and Angelika Steger. Fast-slow recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 5915–5924, 2017. 16

[110] Graham Neubig and Chris Dyer. Generalizing and hybridizing count-based and neural language models. *arXiv preprint arXiv:1606.00499*, 2016. 72

[111] A. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000. 82

[112] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. *arXiv preprint arXiv:1606.00709*, 2016. 94

[113] Xiaoman Pan, Kai Sun, Dian Yu, Heng Ji, and Dong Yu. Improving question answering with external knowledge. *arXiv preprint arXiv:1902.00993*, 2019. 28

[114] Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword fifth edition, linguistic data consortium. *Technical report, Technical Report. Linguistic Data Consortium, Philadelphia, Tech. Rep.*, 2011. 27

[115] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, and Alexander Ku. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018. 10

[116] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018. 2, 4, 9, 10, 12, 19, 20, 27, 33

[117] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 17

[118] Steven Pinker. The language instinct, 1994. 64

[119] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *EACL*, 2017. 49, 63

[120] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007. 71

[121] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 60, 81, 88, 96

[122] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018. 2, 4, 9, 10, 19, 20, 26, 28, 60

[123] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 3

[124] Jack W Rae, Chris Dyer, Peter Dayan, and Timothy P Lillicrap. Fast parametric learning with activation memorization. *arXiv preprint arXiv:1803.10049*, 2018. 16

[125] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019. 33

[126] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016. 28

[127] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018. 27, 28

[128] Qiu Ran, Peng Li, Weiwei Hu, and Jie Zhou. Option comparison network for multiple-choice reading comprehension. *arXiv preprint arXiv:1903.03033*, 2019. 28

[129] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015. 58, 103, 105, 112

[130] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017. 30

[131] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015. 2

[132] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014. 2, 7

[133] Devendra Singh Sachan, Manzil Zaheer, and Ruslan Salakhutdinov. Revisiting lstm networks for semi-supervised text classification via mixed objective function. 2018. 29

[134] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016. xiii, 56, 58, 81, 91, 103, 104, 105, 107, 109, 111, 112, 113, 114

[135] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In *International Conference on Artificial Neural Networks*, pages 632–640. Springer, 2006. 49, 63

[136] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018. 14, 15

[137] Noam Shazeer, Joris Pelemans, and Ciprian Chelba. Skip-gram language modeling using sparse non-negative matrix probability estimation. *arXiv preprint arXiv:1412.1454*, 2014. 17

[138] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-

of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017. 17, 73

[139] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems*, pages 10434–10443, 2018. 17

[140] David R So, Chen Liang, and Quoc V Le. The evolved transformer. *arXiv preprint arXiv:1901.11117*, 2019. 33

[141] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. Amortised map inference for image super-resolution. *arXiv preprint arXiv:1610.04490*, 2016. 112

[142] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*, 2019. 33

[143] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *arXiv preprint arXiv:2004.09297*, 2020. 33, 42, 43

[144] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015. 58, 104, 111, 112

[145] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015. 9

[146] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020. 33

[147] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 9, 69

[148] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015. 112

[149] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830*, 2016. 29

[150] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Adversarial generator-encoder networks. *arXiv preprint arXiv:1704.02304*, 2017. 103, 104

[151] Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, pages 2175–2183, 2013. 8

[152] Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning*, pages 467–475, 2014. 8

[153] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016. 21

[154] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016. 9

[155] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016. 9

[156] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. 2, 3, 10, 14, 15, 24, 33, 34

[157] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1058–1066, 2013. 75

[158] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018. 39

[159] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR. 29

[160] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2015. 9

[161] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012. 103

[162] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014. 13

[163] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019. 33

[164] Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. *arXiv preprint arXiv:2004.11886*, 2020. 33

[165] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*, 2019. 29

[166] Chenyan Xiong, Jamie Callan, and Tie-Yan Liu. Word-entity duet representations for document ranking. In *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*, pages 763–772. ACM, 2017. 31

[167] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*, pages 55–64. ACM, 2017. 31

[168] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016. 103

[169] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017. 17, 26

[170] Zhilin Yang, Junjie Hu, Ruslan Salakhutdinov, and William W Cohen. Semi-supervised qa with generative domain-adaptive nets. *arXiv preprint arXiv:1702.02206*, 2017. 104

[171] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764, 2019. 33, 39, 42, 43

[172] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014. 8, 49, 52, 69

[173] Shuailiang Zhang, Hai Zhao, Yuwei Wu, Zhuosheng Zhang, Xi Zhou, and Xiang Zhou. Dual co-matching network for multi-choice reading comprehension. *arXiv preprint arXiv:1901.09381*, 2019. xi, 28, 31

[174] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015. 29, 39

[175] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016. 81, 82, 86, 93, 110

[176] Tiancheng Zhao, Ran Zhao, and Maxine Eskenazi. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. *arXiv preprint arXiv:1703.10960*, 2017. 69

[177] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003. 103, 107

[178] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015. 27

[179] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. 82, 85

[180] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016. 16, 17, 52, 69

[181] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv*

preprint arXiv:1611.01578, 2016. 17, 52, 69