Université de Montréal

Advances in Deep Learning with Limited Supervision and Computational Resources

par

Amjad Almahairi

Département de mathématiques et de statistique
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Discipline

décembre 2018

Université de Montréal

Faculté des études supérieures et postdoctorales

Cette thèse intitulée

Advances in Deep Learning with Limited Supervision and Computational Resources

présentée par

Amjad Almahairi

a été évaluée par un jury composé des personnes suivantes :

Yoshua Bengio
(président-rapporteur)

Aaron Courville
(directeur de recherche)

Guillaume Rabusseau
(membre du jury)

Lawrence Carin
(examinateur externe)

Mirjam Gollmitzer
(représentant du doyen de la FESP)

Résumé

Les réseaux de neurones profonds sont la pierre angulaire des systèmes à la fine pointe de la technologie pour une vaste gamme de tâches, comme la reconnaissance d'objets, la modélisation du langage et la traduction automatique. Mis à part le progrès important établi dans les architectures et les procédures de formation des réseaux de neurones profonds, deux facteurs ont été la clé du succès remarquable de l'apprentissage profond : la disponibilité de grandes quantités de données étiquetées et la puissance de calcul massive. Cette thèse par articles apporte plusieurs contributions à l'avancement de l'apprentissage profond, en particulier dans les problèmes avec très peu ou pas de données étiquetées, ou avec des ressources informatiques limitées.

Le premier article aborde la question de la rareté des données dans les systèmes de recommandation, en apprenant les représentations distribuées des produits à partir des commentaires d'évaluation de produits en langage naturel. Plus précisément, nous proposons un cadre d'apprentissage multitâches dans lequel nous utilisons des méthodes basées sur les réseaux de neurones pour apprendre les représentations de produits à partir de textes de critiques de produits et de données d'évaluation. Nous démontrons que la méthode proposée peut améliorer la généralisation dans les systèmes de recommandation et atteindre une performance de pointe sur l'ensemble de données Amazon Reviews.

Le deuxième article s'attaque aux défis computationnels qui existent dans l'entraînement des réseaux de neurones profonds à grande échelle. Nous proposons une nouvelle architecture de réseaux de neurones conditionnels permettant d'attribuer la capacité du réseau de façon adaptative, et donc des calculs, dans les différentes régions des entrées. Nous démontrons l'efficacité de notre modèle sur les tâches de reconnaissance visuelle où les objets d'intérêt sont localisés à la couche d'entrée, tout en maintenant une surcharge de calcul beaucoup plus faible que les architectures standards des réseaux de neurones.

Le troisième article contribue au domaine de l'apprentissage non supervisé, avec l'aide du paradigme des réseaux antagoniste génératifs. Nous introduisons un cadre fléxible pour l'entraînement des réseaux antagonistes génératifs, qui non seulement assure que le générateur estime la véritable distribution des données, mais permet également au discriminateur de conserver l'information sur la densité des données à l'optimum global. Nous validons notre

cadre empiriquement en montrant que le discriminateur est capable de récupérer l'énergie de la distribution des données et d'obtenir une qualité d'échantillons à la fine pointe de la technologie.

Enfin, dans le quatrième article, nous nous attaquons au problème de l'apprentissage non supervisé à travers différents domaines. Nous proposons un modèle qui permet d'apprendre des transformations plusieurs à plusieurs à travers deux domaines, et ce, à partir des données non appariées. Nous validons notre approche sur plusieurs ensembles de données se rapportant à l'imagerie, et nous montrons que notre méthode peut être appliquée efficacement dans des situations d'apprentissage semi-supervisé.

Mots-clés: réseaux de neurones, apprentissage automatique, apprentissage profond, apprentissage supervisé, apprentissage non supervisé, apprentissage non supervisé, calcul conditionnel, modèles génératifs probabilistes, réseau antagoniste génératif, synthèse d'images.

Abstract

Deep neural networks are the cornerstone of state-of-the-art systems for a wide range of tasks, including object recognition, language modelling and machine translation. In the last decade, research in the field of deep learning has led to numerous key advances in designing novel architectures and training algorithms for neural networks. However, most success stories in deep learning heavily relied on two main factors: the availability of large amounts of labelled data and massive computational resources. This thesis by articles makes several contributions to advancing deep learning, specifically in problems with limited or no labelled data, or with constrained computational resources.

The first article addresses sparsity of labelled data that emerges in the application field of recommender systems. We propose a multi-task learning framework that leverages natural language reviews in improving recommendation. Specifically, we apply neural-network-based methods for learning representations of products from review text, while learning from rating data. We demonstrate that the proposed method can achieve state-of-the-art performance on the Amazon Reviews dataset.

The second article tackles computational challenges in training large-scale deep neural networks. We propose a conditional computation network architecture which can adaptively assign its capacity, and hence computations, across different regions of the input. We demonstrate the effectiveness of our model on visual recognition tasks where objects are spatially localized within the input, while maintaining much lower computational overhead than standard network architectures.

The third article contributes to the domain of unsupervised learning with the generative adversarial networks paradigm. We introduce a flexible adversarial training framework, in which not only the generator converges to the true data distribution, but also the discriminator recovers the relative density of the data at the optimum. We validate our framework empirically by showing that the discriminator is able to accurately estimate the true energy of data while obtaining state-of-the-art quality of samples.

Finally, in the fourth article, we address the problem of unsupervised domain translation. We propose a model which can learn flexible, many-to-many mappings across domains from unpaired data. We validate our approach on several image datasets, and we show that it

can be effectively applied in semi-supervised learning settings.

Keywords: neural networks, machine learning, deep learning, supervised learning, unsupervised learning, conditional computation, probabilistic generative models, generative adversarial network, image synthesis

Contents

List of tables

List of figures

Acknowledgements

Todo.

Chapter 1

Background

Computers have fundamentally changed every aspect of our lives. Despite being extremely powerful in performing computations, computers still need to be told what to do with a very meticulous and precise set of instructions — an *algorithm*.

The field of Computer Science is primarily concerned with the design and analysis of efficient algorithms. Algorithms are applied to automate a wide range of our every-day tasks, from using a calculator to reserving a flight ticket. Many tasks, however, cannot be easily described as an unambiguous set of instructions. Consider identifying a person's face in an image or understanding a voice command. Such tasks are the main quest of *Artificial Intelligence* (AI), and include problems such as allowing computers to see (computer vision), to understand language (natural language processing) and to recognize speech (speech recognition).

Machine Learning is based on a simple observation: while it is challenging to specify a precise algorithm for solving an AI task, it is usually much easier to identify a correct behavior and provide examples for it. The core idea of machine learning is to let the machine find an algorithm by itself through observing correct behaviour in *data*. This approach has proven to be very effective and has led to many serious breakthroughs in AI tasks in the last few decades.

In this chapter we will provide a brief overview of machine learning fundamentals, followed by a basic review of deep learning, which constitutes the general field of research of this thesis.

1.1. Machine Learning

Machine learning is the study of designing and analyzing algorithms that can adapt or "learn" from data. The main objective is *generalization*. The ability of learning algorithms to generalize to new situations is what distinguishes them from template matching algorithms or look-up tables.

Consider the task of recognizing handwritten digits as shown in Figure 1. Given an input image of size 28×28 gray-scale pixels, the goal is to recognize which of the possible 10 digits

the image represents. A traditional approach for solving this task is to manually design an algorithm which uses heuristics and hand-crafted rules to distinguish digits from their shapes. This turns out to be highly nontrivial due to the large number of possible variations in digit shapes.

A machine learning approach, on the other hand, would be to build an *adaptive algorithm* that learns from data how to solve the handwritten digit recognition task. In the next section, we will discuss basic elements of a machine learning algorithm. While there are many types of machine learning algorithms, we will focus on the digit recognition task as an example of *classification*, which is probably one of the most mature and widely used machine learning tasks. We will provide a general overview of other types of machine learning in section 1.1.2.

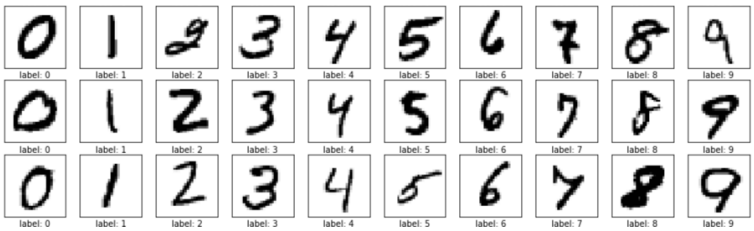


Fig. 1. Examples from MNIST handwritten digits.

1.1.1. Elements of a Learning Algorithm

1.1.1.1. *Training Data*

The data used for learning is called *training data*. In digit classification, training data is a set of N input-output pairs:

$$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}),(\mathbf{x}^{(2)}, y^{(2)}), \ldots ,(\mathbf{x}^{(N)}, y^{(N)})\},$$

where each *training example* is composed of an input image $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and an output class $y^{(i)} \in \{0, \ldots , 9\}$.

Input and output domains, $\mathcal{X}$ and $\mathcal{Y}$, can take different forms depending on the learning task. Generally speaking, domains can be a finite set of elements, scalars in $\mathbb{R}$, vectors in $\mathbb{R}^d$, a finite sequence or a combination of all of them. Furthermore, in some learning tasks training examples are composed of only inputs without specific outputs. This type of learning is called *unsupervised learning*, in contrast to *supervised learning* as in our example of digit classification. Learning paradigms are discussed further in section 1.1.2.

Data generating distribution: A common assumption imposed on training examples is that they are *identically and independently distributed*, or i.i.d. for short. That is, we assume that each example is sampled independently from some distribution $P_{\mathcal{D}}$, which is called the *data generating distribution*. In digit classification, this distribution is defined over $\mathcal{X} \times \mathcal{Y}$, i.e., $P_{\mathcal{D}}(\mathbf{x},y)$. We generally have access to this distribution through a finite set of samples in the

training set $\mathcal{D}$. In practice, training data only comprise a tiny fraction of possible inputs, and this is why generalization is central in machine learning. The i.i.d. assumption is also crucial for guarantees on the generalization capability of most machine learning algorithms.

1.1.1.2. *Model*

A *model* specifies what a machine learning algorithm learns from the data. When designing a learning algorithm we generally specify a family of models $\mathcal{F}$, also known as the *hypothesis space*, which is the set of all possible models that the algorithm can learn. Learning from data reduces to searching for the "best" member of this family – according to a performance measure evaluated on data.

In digit classification, the model can be described as a function $f : \mathcal{X} \to \mathcal{Y}$ and is called a *classifier*. In an "optimal" classifier f^*, the predicted output $\hat{y} = f^*(\mathbf{x})$ matches the true output y for any $(\mathbf{x},y) \in \mathcal{D}$.

Parametric vs. non-parametric models: There are two main categories of models (or model families): *parametric* and *non-parametric*. Parametric models have a fixed-size set of parameters, which is independent of the amount of training data. Non-parametric models, on the other hand, have a variable number of parameters depending on observed data. [1]

A parametric model is typically denoted by a function f_θ, where θ are parameters that define a specific member of the hypothesis space, i.e., $f_\theta \in \mathcal{F}$. For example, linear regression is a parametric model in which weights comprise parameters. On the other hand, k-nearest neighbors (KNN) method is a canonical example of non-parametric methods. KNN stores all training examples, and predictions are made by finding the k most "similar" training examples (according to some similarity measure).

Model capacity: The notion of *model capacity* is central in machine learning. Informally, the capacity of a model (or family of models) is its ability to represent or "fit" training data. If we assume there is a true function $f^* : \mathcal{X} \to \mathcal{Y}$, which represents the true relationship between input and output domains, then the question of model capacity boils down to asking how "large" $\mathcal{F}$ is, which is important because we would like it to be large enough so that $f^* \in \mathcal{F}$. For example, if f^* is a linear function, then linear models have enough capacity to fit linear relationships in data, but they cannot fit more complex, non-linear relationships. Generally speaking, a parametric model's capacity grows as the number of its parameters increases. Neural networks with an arbitrary size are known to be *universal function approximators* (Hornik et al., 1989). The universal approximation theorem states that a single hidden layer neural network can approximate any continuous function to any desired accuracy, provided that it is given enough hidden units (i.e., enough parameters).

[1] Some authors consider a parametric model wrapped in a loop that increases the number of parameters as needed (based on data) as non-parametric.

1.1.1.3. *Loss Function*

In order to evaluate a model's performance, we define a loss function $\ell(f, \mathbf{x}, y)$, which measures the error that a model f makes on a specific example $(\mathbf{x}, y)$. An example of a loss function for digit classification is the zero-one loss:

$$\ell_{01}(f, \mathbf{x}, y) = \begin{cases} 0 & \text{if } f(\mathbf{x}) = y \\ 1 & \text{otherwise.} \end{cases} \tag{1.1.1}$$

Using a loss function, we can define the concept of *expected risk*:

$$\mathcal{L}(f, P_{\mathcal{D}}) = \mathbb{E}_{(\mathbf{x},y)\sim P_{\mathcal{D}}(\mathbf{x},y)} \left[\ell(f,\mathbf{x},y)\right]. \tag{1.1.2}$$

Expected risk allows us to describe the expected amount of error that a model f will incur on the data-generating distribution. This is very important because a model with low expected risk is expected to generalize well to a new data example $(\mathbf{x},y)$, as long as it is sampled from $P_{\mathcal{D}}(\mathbf{x},y)$. Unfortunately, expected loss cannot be computed exactly because we have no direct access to the true distribution $P_{\mathcal{D}}(\mathbf{x},y)$. However, we have access to a finite set of samples from this distribution in the training set $\mathcal{D}$, which can give us a Monte Carlo estimate of expected risk, also known as *empirical risk*:

$$\hat{\mathcal{L}}(f, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \ell(f,\mathbf{x}^{(i)},y^{(i)}). \tag{1.1.3}$$

This concept gives us a principled framework known as *empirical risk minimization* (Vapnik, 1992) for formulating machine learning as an optimization problem.

1.1.1.4. *Optimization Procedure*

A key component of a machine learning algorithm is the procedure used to search for best model within the hypothesis space, according to some objective. This can be formulated as the following optimization problem:

$$f_{\min} = \arg\min_{f\in\mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell(f,\mathbf{x}^{(i)},y^{(i)}). \tag{1.1.4}$$

In some instances, the solution to this optimization problem can be found in closed form. More generally, however, it is solved using iterative optimization algorithms. One important class of algorithms which is used frequently specifically in neural networks is gradient descent. We will discuss it further in section 1.2.2.

We refer to this optimization process as the *training phase* or *learning phase*, as opposed to using a trained model for making predictions, which is called the *testing phase* or *inference phase*.

Generalization, overfitting and underfitting: The question that remains is the following: is there any guarantee that the solution $f_{\min}$ achieves our main objective of generalization to future examples? The field of *statistical learning theory* investigates this question formally. In practice, we can measure generalization of a given model by evaluating its performance on a held-out dataset, which we call a *test set*. The error that a model makes on a test set is called the *test error*, and can give us an estimate of the model's performance on future examples. This is opposed to *training error* which is computed on examples used for training.

There are two main learning failures that happen often in practice. The first is *overfitting*, in which $f_{\min}$ is well tuned on training data $\mathcal{D}$ (low training error), but does not perform well on unseen examples (high test error). Overfitting typically occurs when we assume a family of models $\mathcal{F}$ with a very high capacity. that is, the hypothesis space is too large that some of its members can perfectly fit to noise in training data, without capturing patterns that help in generalizing to unseen examples. The other failure case is the opposite, and is called *underfitting*. This typically happens when we use a very restricted $\mathcal{F}$, such that none of its members can even fit the training set. In this case, both training error and testing error are high. Another form of underfitting can also happen due to *optimization failures*, which means that the optimization process fails to find $f_{\min}$.

Striking a balance between underfitting and overfitting is key to the success of learning, and can be achieved with several methods as we will discuss in the following sections.

1.1.1.5. *Model Selection*

So far, we assumed a fixed hypothesis space $\mathcal{F}$ from which we choose a best member (model) during training. But what if the capacity of this space is not suitable for our problem? The process of controlling the capacity of a model space is referred to as *model selection*. An effective method for model selection is *cross-validation*, in which we divide the available data into three sets: training, validation and test sets. We perform optimization using a training set to find $f_{\min}$, and use the error that the model makes on the validation set, which is referred to as the *validation error*, to measure generalization of $f_{\min}$.

It is common to repeat this process multiple times in order to "tune" the model space $\mathcal{F}$ based on the validation error. More generally, we use this process to tune any parameter of the learning algorithm that is fixed during training. These parameters are called *hyper-parameters*, and are distinct from parameters of the model which are adapted according to training data. Finally, we report the test error of selected model $f_{\min}$, which ensures that the test set is not involved in any step of the learning algorithm.

1.1.1.6. *Regularization*

Another form for striking a balance between underfitting and overfitting can be achieved by choosing a flexible enough hypothesis space, but controlling its capacity through *regularization*. Regularization can be used to bias the optimization criterion such that models with better generalization are preferred. In the empirical risk minimization framework, regularization is characterized by adding an extra term to the optimization criterion:

$$f_{\min} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell(f, \mathbf{x}^{(i)}, y^{(i)}) + \Omega(f). \tag{1.1.5}$$

The function Ω is data-independent and introduces what is known as *inductive bias* to the learning algorithm. Inductive biases refer to any set of assumptions that help the induction or generalization of the learning algorithm. Some commonly used inductive biases include: smoothness, simplicity, similar examples have similar classes. We will discuss in section 1.2.3 some regularization methods used in the context of neural networks. In addition, one of the contributions of this thesis presented in Chapter 3 introduces a specific technique for improving generalization of recommender systems by leveraging additional data sources.

1.1.2. Paradigms of Machine Learning

Machine learning methods can be divided into three main paradigms: *supervised learning*, *unsupervised learning* and *reinforcement learning*.

1.1.2.1. *Supervised Learning*

The goal in supervised learning is to recover an input-output mapping from examples of the desired mapping. Therefore, training data is composed of a set of input-output pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}); i = 1 \dots N\}$.

Supervised learning can be further classified according to the nature of target y into two main categories: (i) *Classification*, which deals with predicting one of a fixed number of discrete values, or *labels*, such as predicting an object identity from an image. (ii) *Regression* deals with predicting real-valued targets, or more generally any ordered set of numerical values, such as a stock price or weather temperature. Another popular form of supervised learning is *structured output prediction*, which deals with more complex targets, such as a structured collection of labels (e.g., image segmentation, where each pixel is classified into one of a fixed set of categories) or a sequence of real values (e.g., time-series of future stock prices).

Supervised learning can be formalized as learning a function $f : \mathcal{X} \to \mathcal{Y}$, which can be modelled by a variety of methods, such as decision trees, support vector machines or neural networks. A probabilistic interpretation of supervised learning casts the problem as

estimating a conditional distribution $P(y \mid \mathbf{x})$. This is arguably a more general view which can account for noise in labels or multimodal mappings.

One class of probabilistic algorithms estimates the conditional distribution directly, and this is called *discriminative* algorithms, while another class, called *generative* algorithms, takes an indirect path by modelling a full joint distribution $P(y,\mathbf{x})$. This is typically done by estimating a prior over targets $P(y)$ and the conditional $P(\mathbf{x} \mid y)$, and then applying Bayes rule to get the posterior $P(y \mid \mathbf{x})$. Explicit modeling of the prior and the conditional can be sometimes useful, especially when we have very good estimates of the prior (e.g., expert knowledge or using extra data resources). In addition, generative algorithms allows sampling new data from the model, but they can be computationally more demanding than discriminative ones.

1.1.2.2. *Unsupervised learning*

While supervised learning is a very general framework that can handle a wide range of practical problems, it is limited to the availability of *labelled data*, i.e., input-output examples pairs. Would it be possible to somehow leverage the massive amounts of *unlabelled data* around us (images, text, sound... etc.)?

The goal of unsupervised learning is to discover latent structure in data, without requiring direct labels. Training data in unsupervised learning problems is hence composed of a set of input examples $\mathcal{D} = \{\mathbf{x}^{(i)}; i = 1 \dots N\}$.

A prominent form of unsupervised learning is *density estimation*, which tries to uncover the true data generating distribution $P_{\mathcal{D}}(\mathbf{x})$ from training examples. Structure in data can be captured in the form of *latent variables*. More precisely, we can decompose the probability of a specific observation $\mathbf{x}$ as $P(\mathbf{x}) = \sum_{\mathbf{z}} P(\mathbf{x}|\mathbf{z})P(\mathbf{z})$. Latent variables allow us to explain *factors of variations* in observed data more precisely, and often assume a simple prior $P(\mathbf{z})$. This kind of information can be very helpful for dealing efficiently with limited amounts of labelled data, and is the basis for a hybrid form of learning, called *semi-supervised learning*.

Other forms of unsupervised learning include *clustering*, where the goal is to find groups of similar examples, which can be very useful for automatic labelling of data. *Dimensionality reduction* also focuses on projecting high-dimensional data into the few most meaningful dimensions. One important application of dimensionality reduction is data visualization.

1.1.2.3. *Reinforcement learning*

Reinforcement learning deals with tasks where the learning algorithm, or *agent* is allowed to interact with a dynamic environment. The agent takes *observations* of the environment, internalizes them as a *state*, performs *actions*, which can affect the environment, and can possibly receive a *reward* for the actions it made. The goal of the agent is to learn a *policy*

or a model of the behavior which maximizes the long-term expected sum of future rewards, or *return*.

Contrary to supervised learning where the algorithm is given examples of correct behavior, a reinforcement learning agent learns policies through indirect, and possibly very scarce, feedback signals.

We focus in this thesis mainly on supervised and unsupervised learning tasks, but many advances in these paradigms can be applied for reinforcement learning, as evidenced in the recent wave of deep reinforcement learning.

1.2. Deep Learning

Machine learning can, in principle, facilitate the automation of a wide range of difficult tasks. However, in practice, the success of traditional learning algorithms relies heavily on extracting good *features* or *representations* of data, as they tend to be not extremely effective when applied directly on raw data. Extracting useful features of data, also known as *feature engineering*, is generally a tedious task that requires a lot of domain expertise. The goal of *representation learning* is to take the automation process a step further by learning features or representations automatically from data. This is typically performed as part of solving a learning task, be it a supervised, unsupervised, or reinforcement learning one.

The field of *deep learning* is concerned with learning deep representations, i.e., hierarchical representations composed of multiple layers of abstraction. This is generally motivated by the nature of data itself. A lot of the data we observe can be explained in a hierarchical form: in language, words make up sentences, which make up paragraphs of a full document. In vision, pixels form edges, and edges form simple shapes, which in turn form more complex shapes in a natural image. On the other hand, neuroscience evidence suggests that the brain processes signals in multiple stages (Bengio, 2009).

While in deep learning there are generally very little assumptions about the learned representation, a cornerstone concept is learning *distributed representations* (Hinton et al., 1986) of data, as opposed to *local representations*. The basic idea is to distribute information about data observations across several dimensions of the feature space, as opposed to assigning a specific dimension for each symbolic concept. As a simple example, we can think of the binary representation of a set of N integers as a distributed representation ($\log_2 N$ space), while a one-hot vector representation is a local representation (N space). A more concrete example is representing words in a vocabulary as vectors in $\mathbb{R}^d$, also called *word embeddings*, where each dimension contributes to the word meaning (distributed representation), as opposed to a one-hot vectors where dimensions are independent of each other (local representation).

A classic example of deep learning models is a feedforward neural network, also called multi-layer perceptron (MLP). MLPs apply multiple layers of nonlinear transformations on

the input to produce the desired output. Each consecutive transformation can be seen as providing a distributed data representation with a higher level of abstraction. What makes these representations appealing is that they are *learned*, i.e., optimized according to some training criterion, as opposed to being hand-crafted and fixed a priori. This, in principle, allows a system to learn powerful data representations with very little prior assumptions.

In the remainder of this section we will review some fundamentals of the field of deep learning that are specifically relevant to the work presented in this thesis. A more comprehensive treatment of deep learning can be found in (Goodfellow et al., 2016).

1.2.1. Feedforward Neural Networks

The basic building block of a feedforward or a fully-connected neural network is an affine transformation applied to an input vector $\mathbf{x} \in \mathbb{R}^n$ followed by a simple non-linearity ϕ:

$$\mathbf{h} = \phi\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right), \tag{1.2.1}$$

where $\mathbf{W} \in \mathbb{R}^{m \times n}$ is a *weight matrix* and $\mathbf{b} \in \mathbb{R}^m$ is a *bias vector*. The function ϕ is applied element-wise, and typical choices for it include:

- *hyperbolic tangent*: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- *logistic or sigmoid*: $\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$
- *rectified linear*: $\text{ReLU}(z) = \max(0,z)$

This building block composes one *hidden layer* in a neural network, and a network is generally composed of multiple hidden layers followed by a final *output layer*. The choice of output layer is very much task dependent. Generally speaking, simple linear models (either for classification or regression) are used as output layers, since the last hidden layer can be viewed as providing a highly nonlinear representation of the input.

A typical choice for the output layer in binary classification tasks is the logistic regression classifier:

$$o = \text{sigmoid}(\mathbf{V}\mathbf{h} + c), \tag{1.2.2}$$

where $\mathbf{V}$, c, o is the weight vector, bias scalar and output scalar, respectively. Since the output $o \in [0, 1]$, we can interpret it as a probability of a class, i.e., $P(y = 1 \mid \mathbf{x})$. It is customary to give a probabilistic interpretation to the output of neural networks. In this example, the network is used to model or *parametrize* a (conditional) Bernoulli distribution.

An example of the computation done in a simple MLP with two hidden layers (using tanh nonlinearities) and a logistic regression output layer is the following:

$$\mathbf{h}^1 = \tanh\left(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1\right) \tag{1.2.3}$$

$$\mathbf{h}^2 = \tanh\left(\mathbf{W}^2\mathbf{h}^1 + \mathbf{b}^2\right) \tag{1.2.4}$$

$$o = \text{sigmoid}\left(\mathbf{V}\mathbf{h}^2 + c\right), \tag{1.2.5}$$

where $\theta = \{\mathbf{W}^1,\mathbf{W}^2,\mathbf{V},\mathbf{b}^1,\mathbf{b}^2,c\}$ are the *parameters* of the network. More precisely, a neural network architecture defines a family of parametric functions $\mathcal{F} = \{f_\theta \mid \theta \in \Theta\}$, where Θ is a parameter space and θ is a specific configuration for the parameters. Optimization, or *training* a neural network, amounts to finding the configuration of the parameters that minimizes the training criterion. As we discussed earlier, the training criterion is defined in terms of the loss function and the choice of regularization. One typical choice for the loss function for binary classification is the cross-entropy loss function:

$$\ell(f_\theta(\mathbf{x}),y) = -y \log f_\theta(\mathbf{x}) - (1-y) \log \left(1 - f_\theta(\mathbf{x})\right). \tag{1.2.6}$$

Notably, this is equivalent to the negative log-likelihood of a Bernoulli with $p = f_\theta(\mathbf{x})$. Minimizing the negative log-likelihood function (or, equivalently, maximizing log-likelihood) is one of the most common learning methods known as *maximum likelihood*.

We can define the training loss for a binary classification network as follows:

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} -y^{(i)} \log f_\theta(\mathbf{x}^{(i)}) - (1-y^{(i)}) \log \Big(1 - f_\theta(\mathbf{x}^{(i)})\Big). \tag{1.2.7}$$

For multi-class classification tasks, the sigmoid function is replaced with the softmax function to provide probabilities for each class. The softmax function takes an arbitrary vector, $\mathbf{a}$, and returns a *stochastic vector* $\mathbf{p} = \text{softmax}(\mathbf{a})$, i.e., $p_i \in [0,1]$ for all i and $\sum_i p_i = 1$. Each element p_i is computed as:

$$p_i = \frac{e^{a_i}}{\sum_j e^{a_j}}. \tag{1.2.8}$$

The output vector $\mathbf{p}$ of the softmax function can be interpreted as the parameters of a Multinomial($\mathbf{p}$,n) distribution (with number of trials $n = 1$), also known as the Multinoulli distribution.

In regression tasks, one can use a simple linear regression output layer:

$$o = \mathbf{V}\mathbf{h} + c, \tag{1.2.9}$$

and a standard loss function in this case is the mean-squared error (MSE):

$$\ell(f_\theta(\mathbf{x}),y) = \left(y - f_\theta(\mathbf{x})\right)^2. \tag{1.2.10}$$

A probabilistic interpretation of this network is given by interpreting the network's output o as the mean of a Gaussian distribution. Minimizing MSE here corresponds to minimizing negative log-likelihood of $\mathcal{N}(o,\sigma^2)$, where σ^2 is a fixed variance (e.g., $\sigma = 1$).

1.2.2. Optimization in Neural Networks

Optimization in deep neural networks is hard because the training loss $\mathcal{L}(\theta, \mathcal{D})$ is a highly non-linear and non-convex function of the parameters. Nevertheless, the training loss

is differentiable with respect to parameters θ, which makes first-order optimization methods, such as *gradient descent*, an appealing choice for deep neural networks.

Gradient descent is an iterative algorithm. Let θ^t be the value of parameters at some learning step t, we start with some initial values θ^0 and update the parameters at each learning step t with the following update rule:

$$\theta^t \leftarrow \theta^{t-1} - \alpha \frac{\partial \mathcal{L}(\theta^{t-1}, \mathcal{D})}{\partial \theta^{t-1}}, \tag{1.2.11}$$

where α is a scalar known as the *learning rate* that controls the amount of change in the parameters each learning step. Training steps are repeated until we reach a satisfactory learning performance.

Computing the gradient of the training loss $\frac{\partial \mathcal{L}(\theta, \mathcal{D})}{\partial \theta}$ can be done efficiently using the *back-propagation* algorithm (Rumelhart et al., 1986). Back-propagation is a dynamic programming algorithm that applies the chain rule of calculus for finding derivatives in the most efficient order: starting from the output, it computes gradients by going backwards to each hidden layer until it reaches the input. More specifically, for a given hidden layer:

$$\begin{aligned}
\mathbf{h}^{(i)} &= \phi\left(\mathbf{a}^{(i)}\right) \\
\mathbf{a}^{(i)} &= \mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)},
\end{aligned}$$

we can compute the gradients of the loss (denoted as $\mathcal{L}$ to avoid notation clutter):

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(i)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(i)}} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{a}^{(i)}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(i-1)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(i)}} \frac{\partial \mathbf{a}^{(i)}}{\partial \mathbf{h}^{(i-1)}}.
\end{aligned}$$

This gives us a recursive definition for computing gradients for each hidden layer. We can compute gradients with respect to parameters in each hidden layer as follows:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(i)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(i)}} \frac{\partial \mathbf{a}^{(i)}}{\partial \mathbf{W}^{(i)}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(i)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(i)}} \frac{\partial \mathbf{a}^{(i)}}{\partial \mathbf{b}^{(i)}}.
\end{aligned}$$

Many software packages support computing gradients based on back-propagation, including Theano (Al-Rfou et al., 2016), Pytorch (Paszke et al., 2017) and Tensorflow (Abadi et al., 2016).

In principle, each gradient computation, and correspondingly each update of the parameters, requires computing gradients over all training examples. This is called batch gradient descent (BGD). In practice, however, BGD can be extremely inefficient, because each update requires scanning through the whole training set, which is generally very large. A more efficient approach is to compute an estimate of the gradient of the training loss for a single or

few training examples (i.e., a *mini-batch*), which is referred to as stochastic gradient descent (SGD). SGD is generally much more efficient in practice, especially with modern hardware which allows for parallel computation of gradients over a mini-batch.

One challenge in gradient based learning is how much signal can be back-propagated as we increase the number of layers in the network. This issue is known as *vanishing gradients* (Bengio et al., 1994), and has been one of the main hurdles for training deep neural networks. A lot of the recent success in deep learning can be attributed to important advances for mitigating this issue including:

- Introducing activation functions, namely rectified linear units and their variants (Glorot et al., 2011; Xu et al., 2015a; Clevert et al., 2016), which allow gradients to flow more smoothly compared to other activations, such as tanh and sigmoid.
- Better parameter initialization techniques (Glorot and Bengio, 2010; Saxe et al., 2013), which allow for better gradient flow especially in the beginning of training.
- Adaptive learning rates (Sutskever et al., 2013; Schaul et al., 2013; Kingma and Ba, 2014), which help in automatically adjusting learning rate and avoiding too large values (possible divergence) or too small values (very slow convergence).
- Normalization techniques for network activations, such as *batch normalization* (Ioffe and Szegedy, 2015), *weight normalization* (Salimans and Kingma, 2016), and *layer normalization* (Ba et al., 2016). The main idea is to normalize the distribution of activations across layers (e.g., zero mean and unit variance), which proves to be highly effective for training very deep networks.
- Architectural innovations, such as *skip connections* between layers to propagate gradient more easily through the network. Models that implement skip connections include *long-short term memory* (Hochreiter and Schmidhuber, 1997), highway networks (Srivastava et al., 2015) and residual networks (He et al., 2016).

1.2.3. Regularization in Neural Networks

There is a wide range of techniques for regularizing neural networks:

- *Weight norm penalty* is the most basic regularization technique. This is based on computing an L_1 or L_2 norm of the network's weights, and minimizing it with the training objective. This regularization technique has a nice probabilistic interpretation, where L_1 and L_2 penalties correspond to Laplace and Gaussian priors over weights, respectively.
- *Dropout* is a very successful technique introduced by Hinton et al. (2012), and was behind one of the great successes of deep learning in object recognition (Krizhevsky et al., 2012). The idea is to randomly drop (set to zero) units in hidden layers during training with some predefined probability (e.g., $p = 0.5$).

- *Data augmentation* is a family of techniques in which we try to add fake data to the original training set. Creating fake data in some tasks, such as image classification, can be easy, because we know the classifier should be *invariant* to a variety of transformations, such as translation and scaling.
- *Multi-task learning* introduced by Caruana (1997) is a form of regularization in neural networks in which we train networks on multiple *related* tasks simultaneously. This can be viewed as training multiple networks which *share parameters*. It forces hidden representation to explain a variety of related tasks – serving as an inductive bias for the main task of interest. Multi-task learning can be especially useful when related tasks provide missing information from the original task, and this is the basis of our contribution in Chapter 3.

In general, the best way to improve generalization is to get more labelled data. Since this is typically very challenging, a lot of research has been focused on improving unsupervised learning in order to leverage the huge amounts of unlabelled data available. The promise of unsupervised learning is to allow learning good representations of data, which can be used for downstream tasks. *Semi-supervised learning* aims at enforcing representations from unlabelled data to align with the supervised task. Another very promising research direction is *generative* modelling, in which the main focus is to learn models that can generate realistic data. We will give a general overview of generative models in deep learning in Section 1.3.

1.2.4. Convolutional Neural Networks

A powerful class of neural network models is the convolutional neural network (CNN). CNNs take advantage of the topological structure of input data, which allows the learning of more powerful representations with much more efficient (computationally and statistically) architectures. While CNNs are especially popular for 2D image data, they have been extensively applied to other types of data (e.g., text and video). In the following we will focus on 2D CNNs.

The main distinction of a CNN from a typical fully-connected network is that it uses two special types of layers: *convolutional* and *pooling* layers. The 2D convolutional layer can be characterized as follows: given a 2D input $\mathbf{x} \in \mathbb{R}^{n \times m}$ (e.g., an image with width n and height m), the convolutional layer uses a *kernel* or *filter* $\mathbf{K} \in \mathbb{R}^{p \times p}$, where typically $p \ll n,m$, and computes a *feature map* $\mathbf{f} \in \mathbb{R}^{(n-p+1)\times(m-p+1)}$ as follows:

$$\mathbf{f}_{i,j} = \phi\left(\text{vec}(\mathbf{K})^T \text{vec}(\mathbf{x}_{i,j})\right), \tag{1.2.12}$$

where $\mathbf{x}_{i,j}$ is an input patch of size $p \times p$ centered at the location (i,j), $\mathbf{f}_{i,j}$ is the result of the dot product of the vectorized matrices at the location (i,j) of the feature map, and ϕ is a typical nonlinearity. This means that the feature map is computed by convolving the kernel

on the full input. It is worth noting here that convolutional layers generally use several kernels and are applied to inputs with multiple channels.

The convolutional layer has two main advantages: first, it allows for *parameter sharing* across many spatial locations, which results in using much fewer parameters and much sparser connectivity compared to a fully connected layer. Second, it produces outputs (representations) that are *equivariant* to input translations. This is especially useful in image representation, because it means that a specific input feature (e.g., an edge) can be detected regardless of its location in the input.

Pooling layers are generally applied after a convolutional layer to subsample the resulting feature map. Subsampling is performed by replacing a local neighborhood in the feature map with a summary statistic, such as the maximum or average value. Pooling helps in achieving *invariance* of the resulting representation with respect to particular changes in the input.

CNNs are basic building blocks for many state-of-the-art models for visual object recognition and detection (Simonyan and Zisserman, 2014; Szegedy et al., 2014; He et al., 2016). Convolutional models have also been successful in other data types which enjoy specific topological structure, such as speech (Abdel-Hamid et al., 2012) and text (Kalchbrenner et al., 2014; Zhang et al., 2015).

1.2.5. Recurrent Neural Networks

Standard feedforward neural networks assume that inputs can be modeled as fixed-size vectors. This is not the case in sequential data, such as word sequences and speech signals. Recurrent neural networks (RNN) provide an extension to neural networks for variable-size inputs. It is worth noting that in principle even CNNs can handle variable size inputs, but they require aggressive sub-sampling to account for long sequences. The question of which model is more suitable for sequential data is still debated (Bai et al., 2018), but RNNs remain the most popular choice for modelling sequences.

Recurrent neural networks preserve the global structure of data by the means of recurrent connections. More formally, given an input sequence of vectors $\mathbf{s} = \mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_T$, an RNN defines the following recurrence:

$$\mathbf{h}_{t+1} = f_{\text{RNN}}\left(\mathbf{x}_t,\mathbf{h}_t\right), \tag{1.2.13}$$

where $\mathbf{h}_t$ is the hidden state of the RNN at time t, and typically $\mathbf{h}_0$ is initialized to some predefined value (e.g., zeros). What makes an RNN model powerful is that the hidden state at time t theoretically encodes information from all previous inputs in order. Achieving this in practice depends largely on how the function f_{RNN} is parameterized. The "vanilla" RNN model defines it as follows:

$$\mathbf{h}_{t+1} = \tanh\left(\mathbf{V}\mathbf{x}_t + \mathbf{W}\mathbf{h}_t + \mathbf{b}\right), \tag{1.2.14}$$

where $\mathbf{V}$, $\mathbf{W}$, and $\mathbf{b}$ are learned parameters. The main problem with this parametrization is that in practice it does not capture long-term dependencies very well, which is largely due to the vanishing gradient problem (Bengio et al., 1994). Several other parametrizations have been studied in the literature to overcome this issue, including LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014) units.

Assuming an appropriate parametrization for f_{RNN}, we can use RNNs for a wide range of tasks. In a supervised learning setting, one class of problems is to map the input sequence to another sequence of the same length. For example, given a sequence of words, we want to predict the part-of-speech tags for each word. In general, we can define an output for each time step as follows:

$$\hat{\mathbf{y}}_t = f_{\text{out}}(\mathbf{h}_t), \tag{1.2.15}$$

where the function f_{out} can be parameterized using a simple linear model or an MLP. The loss of one training example (full sequence) is the sum of losses of all time steps.

Generative modeling is where RNNs have recently had the most impact, for example in language modeling (Mikolov, 2012a) and handwriting generation (Graves, 2013). We can factorize the joint distribution of any sequence $\mathbf{s}$ as follows:

$$P(\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_T) = P(\mathbf{x}_1)P(\mathbf{x}_2 \mid \mathbf{x}_1)\ldots P(\mathbf{x}_T \mid \mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_{T-1}). \tag{1.2.16}$$

An RNN can model each conditional as an output function of the current hidden state, i.e.:

$$P(\mathbf{x}_t \mid \mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_{t-1}) = f_{\text{out}}(\mathbf{h}_t). \tag{1.2.17}$$

RNNs are typically trained by maximizing the log-likelihood of the training sequences, also known as *teacher forcing*.

1.3. Deep Generative Models

1.3.1. Overview

Generative models represent a class of *probabilistic* models, which estimate the true data generating distribution $P_{\mathcal{D}}$ (data distribution) with an estimate P_θ (model distribution).[2] They are called generative to emphasize their capabilities in *synthesizing* data by sampling from P_θ.

In an unsupervised learning setting[3], generative models are given a set of i.i.d. samples $\{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$ from $P_{\mathcal{D}}(\mathbf{x})$, which they use to estimate $P_\theta(\mathbf{x})$. One of the most standard

[2]This can be done explicitly or implicitly.

[3]Generative models can also be used in supervised learning settings, in which we typically estimate conditional distributions $P_\theta(y \mid \mathbf{x})$.

estimation principles is *maximum likelihood estimation*, in which we maximize the objective:

$$\mathcal{L}(P_\theta(\mathbf{x}), P_\mathcal{D}(\mathbf{x})) = \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}(\mathbf{x})} \left[\log \left(P_\theta(\mathbf{x})\right)\right]. \tag{1.3.1}$$

This objective is equivalent to minimizing the *Kullback-Leibler divergence* (KL divergence) between data and model distributions:

$$\begin{aligned} KL\left(P_\mathcal{D}(\mathbf{x}) \| P_\theta(\mathbf{x})\right) &= \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}(\mathbf{x})} \left[\log \frac{P_\mathcal{D}(\mathbf{x})}{P_\theta(\mathbf{x})}\right] \\ &= \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}(\mathbf{x})} \left[\log P_\mathcal{D}(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}(\mathbf{x})} \left[\log P_\theta(\mathbf{x})\right] \\ &= -H(P_\mathcal{D}(\mathbf{x})) - \mathbb{E}_{\mathbf{x}\sim P_\mathcal{D}(\mathbf{x})} \left[\log P_\theta(\mathbf{x})\right], \end{aligned} \tag{1.3.2}$$

where $H(P_\mathcal{D}(\mathbf{x}))$ is the entropy of $P_\mathcal{D}(\mathbf{x})$ and is independent of parameters θ. Beside maximum likelihood, there are many other estimation methods such as Bayesian estimation methods (e.g., maximum a posteriori), and *implicit density estimation methods*, such as noise-contrastive estimation (Gutmann and Hyvärinen, 2010) and generative adversarial methods (Goodfellow et al., 2014).

Deep generative models were developed to harness the power of deep neural networks in generative modelling. This allows for capturing complex relationships in data, but at the same time imposes challenges in learning.

Operations: In generative modelling, we are typically interested in learning models that describe high dimensional data, and perform one or more of the following operations:

- *Sampling*: produce a sample from $P_\theta(\mathbf{x})$. Samples can be useful for *synthesis* tasks, such as image, language or speech synthesis. They can also give an insight into the estimated distribution.
- *Likelihood evaluation*: given a sample $\mathbf{x}$, compute the model likelihood $P_\theta(\mathbf{x})$, or a proxy for it (e.g., energy values). This can be especially useful for applications like outlier detection or as a data-based evaluation metric.
- *Inference*: describe an observed sample $\mathbf{x}$ with a *latent code* or *representation* $\mathbf{z}$. Typically, $\mathbf{z}$ is assumed to have a simpler structure than $\mathbf{x}$, e.g., lower dimensionality, independence, sparsity, etc. Latent codes can be useful to explain *factors of variation* in observed data, and have potential applications in representation learning, dimensionality reduction and semi-supervised learning.

1.3.2. Dominating Methods

Most recently, three classes of methods in deep generative models have proven to be mostly successful: *generative adversarial networks*, *variational autoencoders*, and *autoregressive models*. In the following we will provide a quick review of each approach.

1.3.2.1. *Generative Adversarial Networks*

Generative adversarial networks (GAN) (Goodfellow et al., 2014) learn the data generating distribution by setting up a minimax game between two neural networks: a *generator* and a *discriminator*. The generator, which generates data samples by mapping random noise to the data domain, tries to "fool" the discriminator. The discriminator tries to distinguish between real data samples and generator samples.

More formally, let $\mathcal{X}$ be the data domain, $P_{\mathcal{D}}(\mathbf{x})$ the data distribution and $P_g(\mathbf{x})$ the generator distribution, which is implicitly defined by the sampling process:

$$\mathbf{x} = G(\mathbf{z}), \quad \mathbf{z} \sim P(\mathbf{z}), \tag{1.3.3}$$

where $\mathbf{z}$ is a random noise sample and $P(\mathbf{z})$ is its distribution (e.g., isotropic Gaussian), and G is the generator network.

The discriminator D is a binary classifier $D : \mathcal{X} \to [0,1]$, where $D(\mathbf{x})$ can be interpreted as the probability of $\mathbf{x}$ being a "real" sample (from data distribution).

Finally, the minimax game is defined by optimizing the following *value function*:

$$\max_D \min_G \; V(D,G) = \mathbb{E}_{\mathbf{x}\sim P_{\mathcal{D}}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right]. \tag{1.3.4}$$

Notably, the fixed-point solution to this game is attained when $P_g(\mathbf{x}) = P_{\mathcal{D}}(\mathbf{x})$ for all $\mathbf{x}$. In addition, Goodfellow et al. (2014) show that given an *optimal* discriminator D, minimizing $V(D,G)$ w.r.t. G amounts to minimizing the Jensen-Shannon divergence between $P_g(\mathbf{x})$ and $P_{\mathcal{D}}(\mathbf{x})$.

In practice, the objective 1.3.5 leads to vanishing gradients as the discriminator saturates. Goodfellow et al. (2014) propose the following heuristic training objective, also known as non-saturating GAN objective:

$$\begin{aligned} &\max_D \; \mathbb{E}_{\mathbf{x}\sim P_{\mathcal{D}}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right] \\ &\min_G \; \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\left[-\log D(G(\mathbf{z}))\right] \end{aligned} \tag{1.3.5}$$

GAN is an appealing generative modelling framework specifically because of its efficient sampling (parallelizable across data dimensions), and simple training with gradient-based methods. It has arguably produced the best samples among competing frameworks, especially in image domains. The main downside of GANs is the difficulty of training them, which is still a very active area of research. In addition, the standard GAN formulation does not provide inference and likelihood estimation capabilities.

1.3.2.2. *Variational Autoencoders*

A variational autoencoder (Kingma and Welling, 2013; Rezende et al., 2014) (VAE) is a probabilistic model which assumes observed data $\mathbf{x}$ is explained by the means of continuous

latent codes $\mathbf{z}$. That is, the estimated data distribution $P_\theta(\mathbf{x})$ is described as:

$$P_\theta(\mathbf{x}) = \int_{\mathbf{z}} P_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}, \tag{1.3.6}$$

where we typically assume $p(\mathbf{z})$ to be a simple isotropic Gaussian prior over latent space.

In practice, a VAE is composed of two networks: a *decoder*, which parameterizes a conditional probability $P_\theta(\mathbf{x} \mid \mathbf{z})$, and an *encoder* or *inference network*, which parameterizes a conditional probability over latent space $Q_\phi(\mathbf{z} \mid \mathbf{x})$.

Generating a sample from a VAE is also very simple:

$$\mathbf{x} \sim P_\theta(\mathbf{x} \mid \mathbf{z}), \quad \mathbf{z} \sim p(\mathbf{z}). \tag{1.3.7}$$

For real-valued data, $P_\theta(\mathbf{x} \mid \mathbf{z})$ can be a multivariate Gaussian distribution:

$$\mathbf{x} \sim \mathcal{N}(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})), \tag{1.3.8}$$

where $\mu_\theta(\mathbf{z})$ and $\Sigma_\theta(\mathbf{z}))$ are the mean and covariance matrix, parameterized by the decoder network.

Inference is done in a similar fashion. Given a data sample $\mathbf{x}$:

$$\mathbf{z} \sim \mathcal{N}(\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x})), \tag{1.3.9}$$

where we assume $Q_\phi(\mathbf{z} \mid \mathbf{x})$ to be a multivariate Gaussian distribution over latent space, for which the encoder network is used to parameterize its mean and covariance.

Learning follows the maximum likelihood principle, but since the integral in Eq. 1.3.6 is intractable, we maximize a *variational lower bound* or an *evidence lower bound* (ELBO) on the likelihood function, in which the conditional $Q_\phi(\mathbf{z} \mid \mathbf{x})$ is used as an *approximate posterior*.

In general, VAE offers exact and efficient sampling. Inference is also a key feature of VAE, and has been exploited in applications like semi-supervised learning (Kingma et al., 2014). Exact likelihood computation is still not possible, but the model offers a lower bound on it which can possibly be made tighter (Burda et al., 2015).

1.3.2.3. *Autoregressive Models*

Autoregressive models do not rely on a latent space, and directly describe the joint distribution over N-dimensional observed data $\mathbf{x}$ using the *chain rule of probability*:

$$P_\theta(\mathbf{x}) = P_\theta(x_1) \prod_{k=1}^{N} P_\theta(x_k \mid x_{<k}), \tag{1.3.10}$$

where $P_\theta(x_k \mid x_{<k})$ denotes the conditional distribution of x_k given all previous dimensions $x_{<k}$.

These models leverage the power of deep neural networks in modelling conditionals. One fundamental issue that arises here is how to scale these models to high-dimensional

data. A key solution to this problem is *parameter sharing* between networks parameterizing conditionals (Bengio and Bengio, 2000; Bengio et al., 2003).

These models have gained popularity in sequential data modelling, such as language modelling (Bengio et al., 2003), especially with the use of RNNs and their LSTM variant (Mikolov et al., 2013; Graves, 2013).

Autoregressive models have also proved to be effective in image domains, where the dimensionality can be on the order of tens or even hundreds of thousands (Larochelle and Murray, 2011; Germain et al., 2015; Van Oord et al., 2016).

The main advantage of these models is that they can be trained with an exact maximum likelihood objective with gradient-based methods. However, sampling is not efficient as it has to be done sequentially for each input variable x_k.

Chapter 2

Prologue to First Article

2.1. Article Details

Learning distributed representations from reviews for collaborative filtering. Amjad Almahairi, Kyle Kastner, Kyunghyun Cho, and Aaron Courville. *Proceedings of the 9th ACM Conference on Recommender Systems (RecSys 2015).*

Personal Contribution. I am the main contributor to this work with regards to designing and carrying out experiments, analyzing results and writing the manuscript.

2.2. Context

This work was initially motivated by finding an effective method for evaluating unsupervised learning of distributed representations for natural language. In addition, at the time of publishing this work, there were very few publications which used deep learning in improving recommender systems. Our work builds on the work of McAuley and Leskovec (2013), who introduced the Amazon Reviews dataset which provides a large number of user-product rating data accompanied with natural language reviews. They were also first to show that review text is a rich source of information, which can be leveraged in reducing data sparsity in recommender systems. Their approach was based on learning a topic-model of review text using Latent Dirichlet Allocation (LDA) (Blei et al., 2003), which can be used to regularize representations of users or products. One main question which was not studied in their work is whether learning a more flexible language model, e.g, based on neural networks, can lead to "better" regularization of user or product representations, i.e., lead to further improvements in the recommendation task.

2.3. Contributions

We studied two alternative approaches for modelling review text, and compared them to prior state-of-the-art LDA-based approaches, mainly with respect to improving recommendation. The first is based on a bag-of-words, product-of-experts model and the other is

based on RNNs. We performed comprehensive experiments, which show that the product-of-experts models consistently outperform the LDA-based ones. We show, however, that the flexibility offered by the RNN-based model does not necessarily lead to further improvements in this context. We also made a methodological contribution by highlighting the importance of defining a fixed train / test split in the Amazon Reviews dataset for attaining fair comparisons across published results.

2.4. Recent Developments

Since review text is typically available only at training time, our work focused on learning a language model of reviews as a related task which can help in *regularizing* representations of products used for recommendation. Several follow-up papers explored alternative neural network based approaches for leveraging review text for recommendations. Chen et al. (2016a); Zheng et al. (2017) use all reviews of a target user and a target product to learn their embeddings in a supervised rating prediction task. Catherine and Cohen (2017) shows that the predictive value of these embeddings mainly comes from reviews of the target user-item pair, which is not available at test time. They propose a knowledge distillation approach (Bucilua et al., 2006; Hinton et al., 2015) to alleviate this problem.

In general, deep learning techniques have gained a lot of popularity especially for learning user/product embeddings from additional sources of information (Barkan and Koenigstein, 2016; Vasile et al., 2016; Nedelec et al., 2017). Another tangential line of work has been focusing on leveraging sequential deep learning models (e.g, RNNs) for incorporating temporal dynamics in recommendations (Hidasi et al., 2016; Wu et al., 2016, 2017).

Evaluating learned representations of sentences, especially from unlabelled data remains an open problem. Hill et al. (2016) conducted a thorough comparison of a wide range of unsupervised approaches for learning sentence representations. Their evaluation was based on computing semantic relatedness of two sentence pairs by computing cosine distance between their embeddings, and correlating these distances with gold-standard human judgments. Interestingly, their experiments confirm our results that sentence representations learned by complex models, whose training cost is based on non-linear decoding of the representation, do not perform well when evaluation is based on a simple or linear evaluation method.

Chapter 3

Learning Distributed Representations from Reviews for Collaborative Filtering

3.1. Introduction

Recommendation systems are a crucial component of many e-commerce enterprises, providing businesses with metrics to direct consumers to items they may find appealing. A general goal of these systems is to predict a user's preference for a certain product, often represented as an integer-valued rating, e.g., between 1 (unsatisfied) and 5 (satisfied).

In order to predict the user's preference for a product, it is often beneficial to consider as many sources of information as possible, including the preference of the user for other products, the preferences of other users, as well as any side information such as characteristics of each user and product. A data-driven approach based on this idea is called *collaborative filtering*.

Collaborative filtering has been successfully used for recommendation systems (see, e.g., (Ricci et al., 2011)). A typical approach to using collaborative filtering for recommendation systems is to consider all the observed ratings given by a set of users to a set of products as elements in a matrix, where the row and column of this matrix correspond to users and products, respectively. As the observed ratings are typically only a small subset of the possible ratings (all users rating all products), this matrix is sparse. The goal of collaborative filtering is to fill in the missing values of this matrix: to predict, for each user, the rating of products the user has not rated. In this setting, collaborative filtering is usually cast as a problem of matrix factorization with missing values (Ilin and Raiko, 2010; Mnih and Salakhutdinov, 2007; Salakhutdinov and Mnih, 2008). The sparse matrix is factorized into a product of two matrices of lower rank representing a user matrix and a product matrix. Once these matrices are estimated, a missing observation can be trivially reconstructed by taking a dot product of a corresponding user vector (or representation) and a product vector (or representation).

In this formulation of collaborative filtering, an important issue of data sparsity arises. For instance, the dataset provided as a part of the Netflix Challenge[1] had only 100,480,507 observed ratings out of more than 8 billion possible ratings[2] (user / product pairs) meaning that 99% of the values were missing. This data sparsity easily leads to naive matrix factorization overfitting the training set of observed ratings (Ilin and Raiko, 2010).

In this work, we are interested in regularizing the collaborative filtering matrix factorization using an additional source of information: reviews written by users in natural language. Recent work has shown that better rating prediction can be achieved by incorporating this kind of text-based side information (McAuley and Leskovec, 2013; Ling et al., 2014; Bao et al., 2014). Motivated by these recent successes, here we explore alternative approaches to exploiting this side information. Specifically, we study how different models of reviews can impact the performance of the regularization.

We introduce two approaches to modeling reviews and compare these to the current state-of-the-art LDA-based approaches (McAuley and Leskovec, 2013; Ling et al., 2014). Both models have previously been studied as neural-network-based document models. One is based on the Bag-of-Words Paragraph Vector (Le and Mikolov, 2014). This model is similar to the existing LDA-based model, but, as we argue, it offers a more flexible natural language model. The other is a recurrent neural network (RNN) based approach. RNNs have recently become very popular models of natural language for a wide array of tasks (Le and Mikolov, 2014). Here we will find that despite the considerable additional modelling power brought by the RNN, it does not offer better performance when used as a regularizer in this context.

The proposed approaches are empirically evaluated on the Amazon Reviews Dataset (McAuley and Leskovec, 2013). We observe that the proposed bag-of-words language model outperforms the existing approach based on latent Dirichlet allocation (LDA) (Blei et al., 2003). We also confirm that the use of an RNN language model does not lead to improved performance. Overall, our experiments demonstrate that, in this particular application where we rely on the document model to regularize the collaborative filtering matrix factorization, controlling the model flexibility is very important.

We also make methodological contributions in studying the effect of the train / test splits used in experimentation. Previous works on this subject, e.g, (McAuley and Leskovec, 2013), (Ling et al., 2014) and (Bao et al., 2014), do not clearly identify how the data was split into train and test sets. Here we empirically demonstrate the importance of doing so. We show that for a given fixed split, conclusions regarding the relative performance of competing approaches do generalize to other splits, but comparing absolute performance across difference splits is highly problematic.

[1] `http://www.netflixprize.com/`
[2] 480,189 users and 17,770 movies

3.2. Matrix Factorization for Collaborative Filtering

Given a set of users $U = \{1, \cdots, N\}$ and a set of products (items) $I = \{1, \cdots, M\}$, let $r_{u,i} \in \{1, 2, \cdots, 5\}$ be the rating given by user u to product i. Let us assume that we are given the ratings of a set of observed user-product pairs $O_R \subset U \times I$. Collaborative filtering aims at building a model that is able to predict the rating of an unobserved user-product pair, i.e., $r_{u,i}$ where $(u,i) \notin O_R$.

In collaborative filtering based on matrix factorization, we estimate each user-product rating as

$$r_{u,i} \approx \hat{r}_{u,i} = \mu + \beta_u + \beta_i + \boldsymbol{\gamma}_u^\top \boldsymbol{\gamma}_i, \tag{3.2.1}$$

where μ, β_i and β_u are a global bias, a user-specific bias for the user u and a product-specific bias for the product i, respectively. The vectors $\boldsymbol{\gamma}_u$ and $\boldsymbol{\gamma}_i$ are the latent factors of the user u and the product i, respectively.

We estimate all the parameters in the r.h.s of Eq. (3.2.1) by minimizing the mean-squared error between the predicted ratings and the observed, true ratings:

$$C_R(\boldsymbol{\theta}) = \frac{1}{|O_R|} \sum_{(u,i) \in O_R} \left(\hat{r}_{u,i} - r_{u,i}\right)^2, \tag{3.2.2}$$

where $\boldsymbol{\theta} = \left\{\mu, \{\beta_u\}_{u=1}^N, \{\beta_i\}_{i=1}^M, \{\gamma_u\}_{u=1}^N, \{\boldsymbol{\gamma}_i\}_{i=1}^M\right\}$.

Once the parameters $\boldsymbol{\theta}$ are estimated by minimizing C_R, it is straightforward to predict the rating of an unobserved user-product pair (u,i) using Eq. (3.2.1).

3.2.1. Taming the Curse of Data Sparsity

It has been observed earlier, for instance in (Ilin and Raiko, 2010), that this matrix factorization approach easily overfits the observed ratings, leading to poor generalization performance on the held-out set, or unseen user-product pairs. This issue is especially serious in the case of recommendation systems, as it is highly likely that each user purchases/watches only a fraction of all the available products. For instance, in the Amazon Reviews Dataset more than 99.999% of ratings, or elements in the rating matrix are missing.

The issue of overfitting is often addressed by adding a regularization term Ω to the cost C_R in Eq. (3.2.2). One of the most widely used regularization term is a simple weight decay

$$\Omega(\boldsymbol{\theta}) = \sum_{\theta \in \boldsymbol{\theta}} \|\theta\|^2.$$

Hence parameters are estimated by minimizing $C_R(\boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\theta})$, where λ is a regularization coefficient.

Another approach is to interpret matrix factorization in a probabilistic framework (Ilin and Raiko, 2010; Mnih and Salakhutdinov, 2007; Salakhutdinov and Mnih, 2008). In this approach, all the parameters such as the user and product representations are considered as

latent random variables on which the distribution of the rating, an observed random variable, is conditioned. This probabilistic matrix factorization can automatically regularize itself by maintaining the confidence of the estimates/predictions based on the observations.

On the other hand, we can improve generalization, hence reduce overfitting, by simultaneously estimating the parameters $\boldsymbol{\theta}$ of the matrix factorization to perform well on another related task (Caruana, 1997). With a model predicting a rating by a user on a product, we can consider letting the model also try to account for the product review given by the user. This seems like a useful side task as users often write reviews that justify their ratings and describe features that affected their opinions.

We explore this approach of exploiting extra tasks to improve generalization performance of collaborative filtering based on matrix factorization.

3.3. Regularizing with Extra Data

3.3.1. Reviews as Extra Data

In many e-commerce systems, each rating is often accompanied with a user's review of the product. As mentioned earlier, it is natural to expect that the accompanying review is used by the user to justify her/his rating or opinion. Hence, we expect that a prediction model for ratings can be improved by taking into account user reviews (McAuley and Leskovec, 2013). As an illustrating example, a user, who wrote "this is a great adventure movie that children and adults alike would love!" for the movie "Free Willy", is likely to give a high rating to this movie.[3]

In this section, we will propose two approaches to utilizing this type of review data for improving the generalization performance of a rating prediction model based on matrix factorization.

3.3.2. Natural Language Review Modeling

More technically, let us suppose that we have access to the reviews for a set of user-product pairs $O_D \subseteq O_R$. Each review $d_{u,i} = \left(w_{u,i}^{(1)}, \cdots, w_{u,i}^{(n_{u,i})}\right)$ is a piece of natural language text written by a user u about an item i, which we represent as a sequence of words.

Following the multitask learning framework (Caruana, 1997), we build a model that jointly predicts the rating given by a user u to a product i and models the review written by the user u on the product i. The model has two components; matrix factorization in Eq. (3.2.1) and review modeling, which shares some of the parameters $\boldsymbol{\theta}$ from the rating prediction model.

[3]This is an actual sample from the Amazon Reviews datatset.

Here, we follow the approach from (McAuley and Leskovec, 2013) by modeling the conditional probability of each review given the corresponding product $\boldsymbol{\gamma}_i$:

$$p\left(d_{u,i} = \left(w_{u,i}^{(1)}, \cdots, w_{u,i}^{(n_{u,i})}\right) \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D\right), \tag{3.3.1}$$

where $\boldsymbol{\theta}_D$ is a set of parameters specific for this review model.[4]

We estimate the parameters of this review model ($\boldsymbol{\theta}_D$ and $\boldsymbol{\gamma}_i$'s) by minimizing the negative log-likelihood:

$$\underset{\boldsymbol{\theta}_D, \{\boldsymbol{\gamma}_i\}_{i=1}^M}{\arg\min}\ C_D(\boldsymbol{\theta}_D, \{\boldsymbol{\gamma}_i\}_{i=1}^M),$$

where

$$C_D(\boldsymbol{\theta}_D, \{\boldsymbol{\gamma}_i\}_{i=1}^M) = -\frac{1}{|O_D|} \sum_{(u,i)\in O_D} \log p\left(d_{u,i} = \left(w_{u,i}^{(1)}, \cdots, w_{u,i}^{(n_{u,i})}\right) \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D\right). \tag{3.3.2}$$

We jointly optimize the rating prediction model in Eq. (3.2.1) and the review model in Eq. (3.3.1) by minimizing the convex combination of C_R in Eq. (3.2.2) and C_D in Eq. (3.3.2):

$$\underset{\boldsymbol{\theta}, \boldsymbol{\theta}_D}{\arg\min}\, \alpha\, C_R(\boldsymbol{\theta}) + (1-\alpha) C_D(\boldsymbol{\theta}_D, \{\boldsymbol{\gamma}_i\}_{i=1}^M), \tag{3.3.3}$$

where the coefficient α is a hyperparmeter.

3.3.2.1. *BoWLF: Distributed Bag-of-Word*

The first model we propose to use is a distributed bag-of-words prediction. In this case, we represent each review as a bag of words, meaning

$$d_{u,i} = \left(w_{u,i}^{(1)}, \cdots, w_{u,i}^{(n_{u,i})}\right) \approx \left\{w_{u,i}^{(1)}, \cdots, w_{u,i}^{(n_{u,i})}\right\}. \tag{3.3.4}$$

This leads to

$$p(d_{u,i} \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D) = \prod_{t=1}^{n_{u,i}} p(w_{u,i}^{(t)} \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D).$$

We model $p(w_{u,i}^{(t)} \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D)$ as an affine transformation of the product representation $\boldsymbol{\gamma}_i$ followed by, so-called softmax, normalization:

$$p(w_{u,i}^{(t)} = j \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D) = \frac{\exp\{y_j\}}{\sum_{l=1}^{|V|} \exp\{y_l\}}, \tag{3.3.5}$$

where

$$\mathbf{y} = \mathbf{W}\boldsymbol{\gamma}_i + \mathbf{b}$$

and V, $\mathbf{W}$ and $\mathbf{b}$ are the vocabulary, a weight matrix and a bias vector. The parameters $\boldsymbol{\theta}_D$ of this review model include $\mathbf{W}$ and $\mathbf{b}$.

[4]Note that in principle we could also condition on the user factors $\boldsymbol{\gamma}_u$, but our initial experiments show that this can hurt generalization performance.

When we use this distributed bag-of-words together with matrix factorization for predicting ratings, we call this joint model the *bag-of-words regularized latent factor model* (BoWLF).

3.3.2.2. *LMLF: Recurrent Neural Network*

The second model of reviews we propose to use is a recurrent neural network (RNN) language model (LM) (Mikolov, 2012b). Unlike the distributed bag-of-words model, this RNN-LM does not make any assumption on how each review is represented, but takes a sequence of words as it is, preserving the order of the words.

In this case, we model the probability over a review which is a variable-length sequence of words by rewriting the probability as

$$p(d_{u,i} \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D) = p\left(w_{u,i}^{(1)} \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D\right) \prod_{t=2}^{n_{u,i}} p\left(w_{u,i}^{(t)} \mid w_{u,i}^{(1)}, \cdots, w_{u,i}^{(t-1)}, \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D\right),$$

We approximate each conditional distribution with

$$p\left(w_{u,i}^{(t)} = j \mid w_{u,i}^{(<t)}, \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D\right) = \frac{\exp\left\{y_j^{(t)}\right\}}{\sum_{l=1}^{|V|} \exp\left\{y_l^{(t)}\right\}},$$

where

$$\mathbf{y}^{(t)} = \mathbf{W}\mathbf{h}^{(t)} + \mathbf{b}$$

and

$$\mathbf{h}^{(t)} = \phi\left(\mathbf{h}^{(t-1)}, w_{u,i}^{(t-1)}, \boldsymbol{\gamma}_i\right).$$

There are a number of choices available for implementing the recurrent function ϕ. Here, we use a long short-term memory (LSTM, (Hochreiter and Schmidhuber, 1997)) which has recently been applied successfully to natural language-related tasks (Graves, 2013).

In the case of the LSTM, the recurrent function ϕ returns, in addition to its hidden state $\mathbf{h}^{(t)}$, the memory cell $\mathbf{c}^{(t)}$ such that

$$\left[\mathbf{h}^{(t)}; \mathbf{c}^{(t)}\right] = \phi\left(\mathbf{h}^{(t-1)}, \mathbf{c}^{(t-1)}, w_{u,i}^{(t-1)}, \boldsymbol{\gamma}_i\right),$$

where

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)})$$
$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \tilde{\mathbf{c}}^{(t)}.$$

The output $\mathbf{o}$, forget $\mathbf{f}$ and input $\mathbf{i}$ gates are computed by

$$\begin{bmatrix} \mathbf{o}^{(t)} \\ \mathbf{f}^{(t)} \\ \mathbf{i}^{(t)} \end{bmatrix} = \sigma(\mathbf{V}_g \mathbf{E}\left[w_{u,i}^{(t-1)}\right] + \mathbf{W}_g \mathbf{h}^{(t-1)} + \mathbf{U}_g \mathbf{c}^{(t-1)} + \mathbf{A}_g \boldsymbol{\gamma}_i + \mathbf{b}_g), \quad (3.3.6)$$

and the new memory content $\tilde{\mathbf{c}}^{(t)}$ by

$$\tilde{\mathbf{c}}^{(t)} = \tanh(\mathbf{V}_c \mathbf{E}\left[w_{u,i}^{(t-1)}\right] + \mathbf{W}_c \mathbf{h}^{(t-1)} + \mathbf{U}_c \mathbf{c}^{(t-1)} + \mathbf{A}_c \boldsymbol{\gamma}_i + \mathbf{b}_c), \quad (3.3.7)$$

where $\mathbf{E}$, $\mathbf{V}_g$, $\mathbf{W}_g$, $\mathbf{U}_g$, $\mathbf{b}_g$, $\mathbf{V}_c$, $\mathbf{W}_c$, $\mathbf{U}_c$, $\mathbf{b}_c$, $\mathbf{A}_g$ and $\mathbf{A}_c$ are the parameters of the RNN-LM. Note that $\mathbf{E}[w]$ denotes a row indexing by the word index w of the matrix $\mathbf{E}$.

Similarly to the BoWLF, we call the joint model of matrix factorization and this RNN-LM the *language model regularized latent factor model* (LMLF).

3.3.3. Related Work: LDA-based Approach

Similar approaches of modeling reviews to regularize matrix factorization have recently been proposed, however, with different review models such as LDA (McAuley and Leskovec, 2013; Ling et al., 2014) and non-negative matrix factorization (Bao et al., 2014). Here, we describe "Hidden Factors as Topics" (HFT) recently proposed in (McAuley and Leskovec, 2013), and discuss it with respect to the proposed approaches.

The HFT model is based on latent Dirichlet allocation (LDA, (Blei et al., 2003)), and similarly to the distributed bag-of-word model in Sec. 3.3.2.1, considers each review as a bag of words (see Eq. (3.3.4).) Thus, we start by describing how LDA models a review $d_{u,i}$. LDA is a generative model of a review/document. It starts by sampling a so-called topic proportion $\boldsymbol{\tau}$ from a Dirichlet distribution. The topic proportion $\boldsymbol{\tau}$ is used as a parameter to a multinomial topic distribution from which a topic is sampled, for each word in the document. The sampled topic defines a probability distribution over the words in a vocabulary. In other words, given a topic proportion, the LDA models a review with a mixture of multinomial distributions.

Instead of sampling the topic proportion from the top-level Dirichlet distribution in LDA, HFT replaces it with

$$\boldsymbol{\tau} = \frac{1}{\|\exp\{\kappa \boldsymbol{\gamma}_i\}\|_1} \exp\{\kappa \boldsymbol{\gamma}_i\},$$

where κ is a free parameter estimated along with all the other parameters of the model. In this case, the probability over a single review $d_{u,i}$ given an item $\boldsymbol{\gamma}_i$ becomes

$$p(d_{u,i} \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D) = \prod_{t=1}^{n_{u,i}} \sum_{k=1}^{\dim(\boldsymbol{\gamma}_i)} p(w_{u,i}^{(t)} \mid z_k = 1) p(z_k = 1 \mid \boldsymbol{\gamma}_i) \tag{3.3.8}$$
$$= \prod_{t=1}^{n_{u,i}} \sum_{k=1}^{\dim(\boldsymbol{\gamma}_i)} \tau_k p(w_{u,i}^{(t)} \mid z_k = 1)$$

where z_k is an indicator variable of the k-th topic out of $\dim(\boldsymbol{\gamma}_i)$, and τ_k is the k-th element of $\boldsymbol{\tau}$. The conditional probability over words given a topic is modeled with a matrix $\mathbf{W}^* = \left[w_{j,k}^*\right]_{|V| \times \dim(\boldsymbol{\gamma}_i)}$, in which each column sums to 1. The conditional probability over words given a product $\boldsymbol{\gamma}_i$ can be written as

$$p(w_{u,i}^{(t)} = j \mid \boldsymbol{\gamma}_i, \boldsymbol{\theta}_D) = \sum_{k=1}^{\dim(\boldsymbol{\gamma}_i)} w_{j,k}^* \frac{\exp\left\{\kappa \gamma_{i,k}\right\}}{\left\|\exp\left\{\kappa \boldsymbol{\gamma}_i\right\}\right\|_1}. \tag{3.3.9}$$

The matrix $\mathbf{W}^*$ is often parametrized by $w_{j,k}^* = \frac{\exp\left\{q_{j,k}\right\}}{\sum_l \exp\left\{q_{l,k}\right\}}$, where $\mathbf{Q} = [q_{j,k}]$ is an unconstrained matrix of the same size as $\mathbf{W}^*$. In practice, a bias term is added to the formulation above to handle frequent words.

3.3.4. Comparing HFT and BoWLF

From Eq. (3.3.5) and Eq. (3.3.9), we can see that the HFT and the proposed BoWLF (see Sec. 3.3.2.1) are closely related. Most importantly, both of them consider a review as a bag of words and parametrize the conditional probability of a word given a product representation with a single affine transformation (weight matrix plus offset vector).

The main difference is in how the product representation and the weight matrix interact to form a point on the $|V|$-dimensional simplex. In the case of HFT, both the product representation $\boldsymbol{\gamma}_i$ and the projection matrix $\mathbf{W}^*$ are separately stochastic (i.e., each $\boldsymbol{\gamma}_i$ and each column of $\mathbf{W}^*$ are interpretable as a probability distribution), while the BoWLF projects the result of the matrix-vector product $\mathbf{W}\boldsymbol{\gamma}_i$ onto the probability simplex.

This can be understood as the difference between a mixture of experts and a product of experts (Hinton, 1999). On a per word basis, the BoWLF in Eq. (3.3.5) can be re-written as a (conditional) product of experts by

$$p(w = j \mid \boldsymbol{\gamma}_i) = \frac{1}{Z(\boldsymbol{\gamma}_i)} \prod_{k=1}^{\dim(\boldsymbol{\gamma}_i)} \exp\left\{w_{j,k} \gamma_{i,k} + b_j\right\},$$

where $w_{j,k}$ and b_j are the element at the j-th row and k-th column of $\mathbf{W}$ and the j-th element of $\mathbf{b}$, respectively. On the other hand, an inspection of Eq. (3.3.8) reveals that, on a per word basis, the HFT model is clearly a mixture model, with the topics playing the role of the mixture components.

As argued in (Hinton, 1999), a product of experts can more easily model a *peaky* distribution, especially, in a high-dimensional space.[5] The reviews of each product tend to contain a small common subset of the whole vocabulary, while those subsets vastly differ from each other depending on the product. In other words, the conditional distribution of words given a product puts most of its probability mass on only a few product-specific words, while leaving most other words with nearly zero probabilities. Product of experts are naturally better suited to modeling peaky distributions rather than mixture models.

A more concrete way of understanding the difference between HFT and BoWLF may be to consider how the product representation and the weight matrix interact. In the case of the BoWLF, this is a simple matrix-vector product with no restrictions on the weight matrix. This means that both the product representation elements as well as the elements of the weight matrix are free to assume negative values. Thus, it is possible that an element of the product representation could exercise a strong influence *suppressing* the expression of a given set of words. Alternatively, with HFT model, as the model interprets the elements of the product representation as mixture components, these elements have no mechanism of suppressing probability mass assigned to words by the other elements of the product representation.

We suggest that this difference allows the BoWLF to better model reviews compared to the HFT, or any other LDA-based model by offering a mechanism for negative correlations between words to be explicitly expressed by elements of the product representation. By offering a more flexible and natural model of reviews, the BoWLF model can improve the rating prediction generalization performance. As we will see in Sec. 3.4, our experimental results support this proposition.

The proposed LMLF takes one step further by modeling each review with a chain of products of experts taking into account the order of words. This may seem an obvious benefit at first sight. However, it is not clear whether modelling word order in reviews would result in learning better latent factors for items or simply learn a better language model of reviews. Indeed, as we show in the following section, we observe that the LMLF can model model reviews very well, but does not necessarily do better than BoWLF at rating prediction.

3.4. Experiments

3.4.1. Dataset

We evaluate the proposed approaches on the Amazon Reviews dataset (McAuley and Leskovec, 2013).[6] There are approximate 35 million ratings and accompanying reviews from

[5]Note that the size of a usual vocabulary of reviews is on the order of thousands.
[6]`https://snap.stanford.edu/data/web-Amazon.html`

6,643,669 users and 2,441,053 products. The products are divided into 28 categories such as music and books. The reviews are on average 110 words long. We refer the reader to (Ling et al., 2014) for more detailed statistics.

3.4.2. Experimental Setup

Data Preparation. We closely follow the procedure from (McAuley and Leskovec, 2013) and (Ling et al., 2014), where the evaluation is done per category. We randomly select 80% of ratings, up to two million samples, as a training set, and split the rest evenly into validation and test sets, for each category. We preprocess reviews only by tokenizing them using a script from Moses[7], after which we build a vocabulary of 5000 most frequent words.

Evaluation Criteria. We use mean squared error (MSE) of the rating prediction to evaluate each approach. For assessing the performance on review modeling, we use the average negative log-likelihood.

Baseline. We compare the two proposed approaches, BoWLF (see Sec. 3.3.2.1) and LMLF (see Sec. 3.3.2.2), against three baseline methods; matrix factorization with L_2 regularization (MF, see Eqs. (3.2.1)–(3.2.2)), the HFT model from (McAuley and Leskovec, 2013) (see Sec. 3.3.3) and the RMR model from (Ling et al., 2014). In the case of HFT, we report the performance both by evaluating the model ourselves[8] and by reporting the results from (McAuley and Leskovec, 2013) directly. For RMR, we only report the results from (Ling et al., 2014).

Hyper-parameters. Both user $\boldsymbol{\gamma}_u$ and product $\boldsymbol{\gamma}_i$ vectors in Eq. (3.2.1) are five dimensional for all the experiments in this section. This choice was made mainly to make the results comparable to the previously reported ones in (McAuley and Leskovec, 2013) and (Ling et al., 2014). We initialize all the user and product representations by sampling each element from a zero-mean Gaussian distribution with its standard deviation set to 0.01. The biases, μ, β_u and β_i are all initialized to 0. All the parameters in BoWLF and LMLF are initialized similarly except for the recurrent weights of the RNN-LM in LMLF which were initialized to be orthogonal.

Training Procedure. When training MF, BoWLF and LMLF, we use minibatch RMSProp with the learning rate, momentum coefficient and the size of minibatch set to 0.01, 0.9 and 128, respectively. We trained each model at most 200 epochs, while monitoring the validation performance. For HFT, we follow (McAuley and Leskovec, 2013) which uses the Expectation Maximization algorithm together with L-BFGS. In all cases, we early-stop each training run based on the validation set performance. In the preliminary experiments, we found the choice of α in Eq. (3.3.3), which balances matrix factorization and review modeling, to be

[7] `https://github.com/moses-smt/mosesdecoder/`

[8] The code was kindly provided by the authors of (McAuley and Leskovec, 2013).

important. We searched for the α that maximizes the validation performance, in the range of $[0.1, 0.01]$.

3.4.3. Rating Prediction Results

Dataset	Dataset Size	(a) MF	(b) HFT	(c) BoWLF	(d) LMLF	BoWLF improvement over (a)	BoWLF improvement over (b)	HFT*	RMR**
Arts	27K	1.434 (0.04)	1.425 (0.04)	**1.413** (0.04)	1.426 (0.04)	2.15%	1.18%	1.388	1.371
Jewelry	58K	1.227 (0.04)	**1.208** (0.03)	1.214 (0.03)	1.218 (0.03)	1.24%	-0.59%	1.178	1.160
Watches	68K	1.511 (0.03)	1.468 (0.03)	**1.466** (0.03)	1.473 (0.03)	4.52%	0.20%	1.486	1.458
Cell Phones	78K	2.133 (0.03)	2.082 (0.02)	**2.076** (0.02)	2.077 (0.02)	5.76%	0.66%	N/A	2.085
Musical Inst.	85K	1.426 (0.02)	1.382 (0.02)	**1.375** (0.02)	1.388 (0.02)	5.12%	0.75%	1.396	1.374
Software	95K	2.241 (0.02)	2.194 (0.02)	**2.174** (0.02)	2.203 (0.02)	6.70%	2.06%	2.197	2.173
Industrial	137K	0.360 (0.01)	0.354 (0.01)	**0.352** (0.01)	0.356 (0.01)	0.76%	0.24%	0.357	0.362
Office Products	138K	1.662 (0.02)	1.656 (0.02)	**1.629** (0.02)	1.646 (0.02)	3.32%	2.72%	1.680	1.638
Gourmet Foods	154K	1.517 (0.02)	1.486 (0.02)	**1.464** (0.02)	1.478 (0.02)	5.36%	2.22%	1.431	1.465
Automotive	188K	1.460 (0.01)	1.429 (0.01)	**1.419** (0.01)	1.428 (0.01)	4.17%	1.03%	1.428	1.403
Kindle Store	160K	1.496 (0.01)	1.435 (0.01)	**1.418** (0.01)	1.437 (0.01)	7.83%	1.76%	N/A	1.412
Baby	184K	1.492 (0.01)	1.437 (0.01)	**1.432** (0.01)	1.443 (0.01)	5.95%	0.48%	1.442	N/A
Patio	206K	1.725 (0.01)	1.687 (0.01)	**1.674** (0.01)	1.680 (0.01)	5.10%	1.24%	N/A	1.669
Pet Supplies	217K	1.583 (0.01)	1.554 (0.01)	**1.536** (0.01)	1.544 (0.01)	4.74%	1.78%	1.582	1.562
Beauty	252K	1.378 (0.01)	1.373 (0.01)	**1.335** (0.01)	1.370 (0.01)	4.33%	3.82%	1.347	1.334
Shoes	389K	0.226 (0.00)	0.231 (0.00)	**0.224** (0.00)	0.225 (0.00)	0.23%	0.72%	0.226	0.251
Tools & Home	409K	1.535 (0.01)	1.498 (0.01)	**1.477** (0.01)	1.490 (0.01)	5.78%	2.15%	1.499	1.491
Health	428K	1.535 (0.01)	1.509 (0.01)	**1.481** (0.01)	1.499 (0.01)	5.35%	2.82%	1.528	1.512
Toys & Games	435K	1.411 (0.01)	1.372 (0.01)	**1.363** (0.01)	1.367 (0.01)	4.71%	0.89%	1.366	1.372
Video Games	463K	1.566 (0.01)	1.501 (0.01)	**1.481** (0.01)	1.490 (0.01)	8.47%	2.00%	1.511	1.510
Sports	510K	1.144 (0.01)	1.137 (0.01)	**1.115** (0.01)	1.127 (0.01)	2.94%	2.19%	1.136	1.129
Clothing	581K	0.339 (0.00)	0.343 (0.00)	**0.333** (0.00)	0.344 (0.00)	0.60%	1.01%	0.327	0.336
Amazon Video	717K	1.317 (0.01)	1.239 (0.01)	**1.184** (0.01)	1.206 (0.01)	13.33%	5.47%	N/A	1.270
Home	991K	1.587 (0.00)	1.541 (0.00)	**1.513** (0.00)	1.535 (0.01)	7.41%	2.79%	1.527	1.501
Electronics	1.2M	1.754 (0.00)	1.694 (0.00)	**1.671** (0.00)	1.698 (0.00)	8.29%	2.30%	1.724	1.722
Music	6.3M	1.112 (0.00)	0.970 (0.00)	**0.920** (0.00)	0.924 (0.00)	19.15%	4.94%	0.969	0.959
Movies & Tv	7.8M	1.379 (0.00)	1.089 (0.00)	**0.999** (0.00)	1.022 (0.00)	37.95%	9.01%	1.119	1.120
Books	12.8M	1.272 (0.00)	1.141 (0.00)	**1.080** (0.00)	1.110 (0.00)	19.21%	6.12%	1.135	1.113
All categories	35.3M	1.289	1.143	1.086	1.107	20.29%	5.64%		

Table 1. Prediction Mean Squared Error results on test data. Standard error of mean in parenthesis. Dimensionality of latent factors $\dim(\boldsymbol{\gamma}_i) = 5$ for all models. Best results for each dataset in bold. HFT* and RMR** represent original paper results over different data splits (McAuley and Leskovec, 2013; Ling et al., 2014).

We list results of the experiments in Table 1 for the 28 categories in terms of MSE with the standard error of mean shown in parentheses. From this table, we can see that except for a single category of "Jewelry", the proposed BoWLF outperforms all the other models with an improvement of 20.29% over MF and 5.64% over HFT across all categories.[9] In general, we note better performance of BoWLF and LMLF models over other methods especially as the size of the dataset grows, which is evident from Figs. 1 and 2.

[9]Due to the use of different splits, the results by HFT reported in (McAuley and Leskovec, 2013) and RMR in (Ling et al., 2014) are not directly comparable.

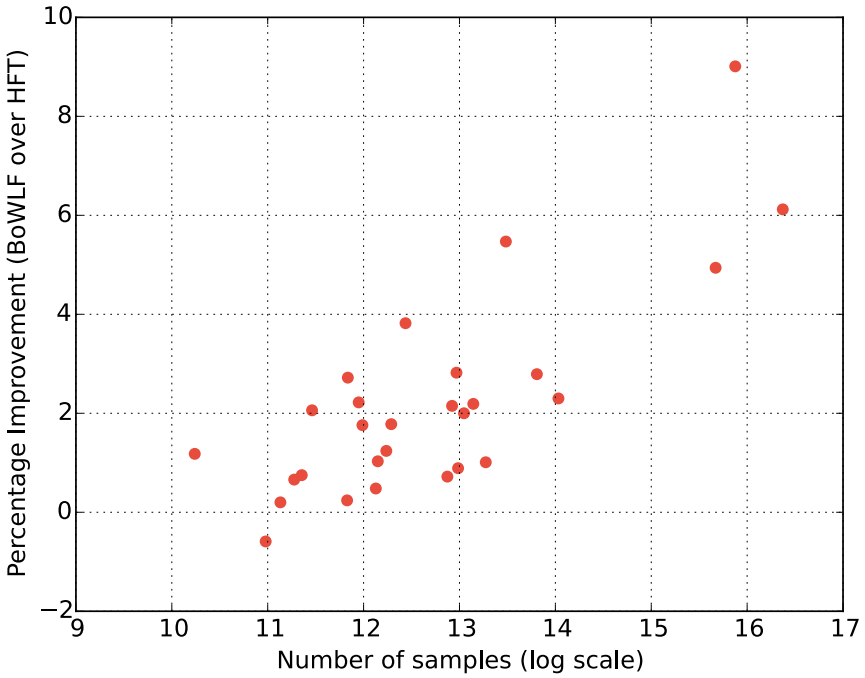


Fig. 1. Scatterplot showing performance improvement over the number of samples. We see a performance improvement of BoWLF over HFT as dataset size increases.

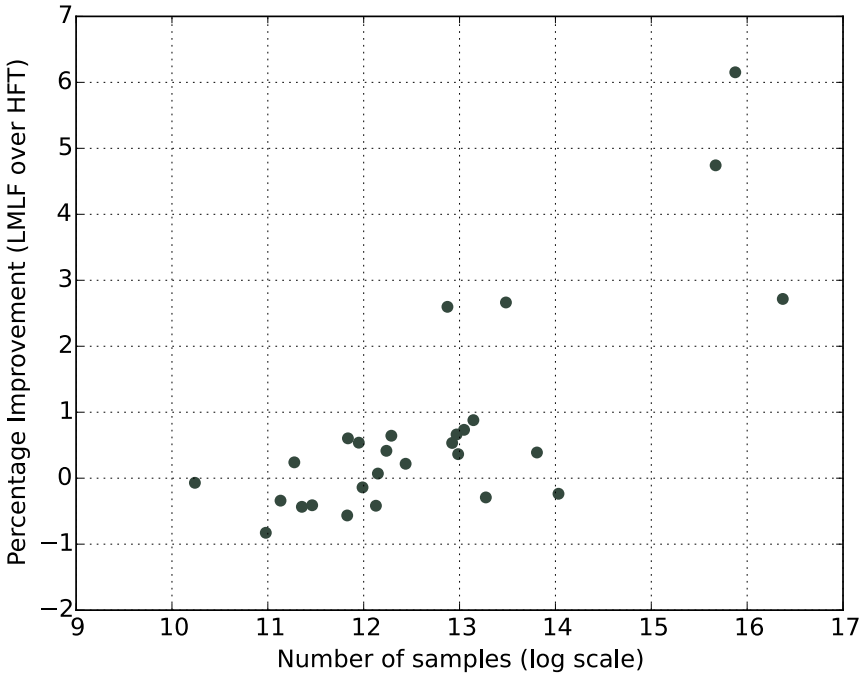


Fig. 2. Scatterplot showing performance improvement over the number of samples. We see a modest performance improvement of LMLF over HFT as dataset size increases.

Interestingly, BoWLF always outperforms LMLF. These results indicate that the complex language model, which the LMLF learns using an LSTM network, does not seem to improve

over a simple bag-of-word representation, which the BoWLF learns, in terms of the learned product representations.

This can be understood from how the product representation, which is used *linearly* by the rating prediction model, is handled by each model. The word distribution modeled by the BoWLF depends linearly on the product representation, which requires the product-related structure underlying reviews be encoded linearly as well. On the other hand, LMLF *nonlinearly* manipulates the product representation to approximate the distribution over reviews. In other words, the LMLF does not necessarily encode the underlying product-related structure inside the product representation in the way the rating prediction model can easily decode (Mikolov et al., 2013).

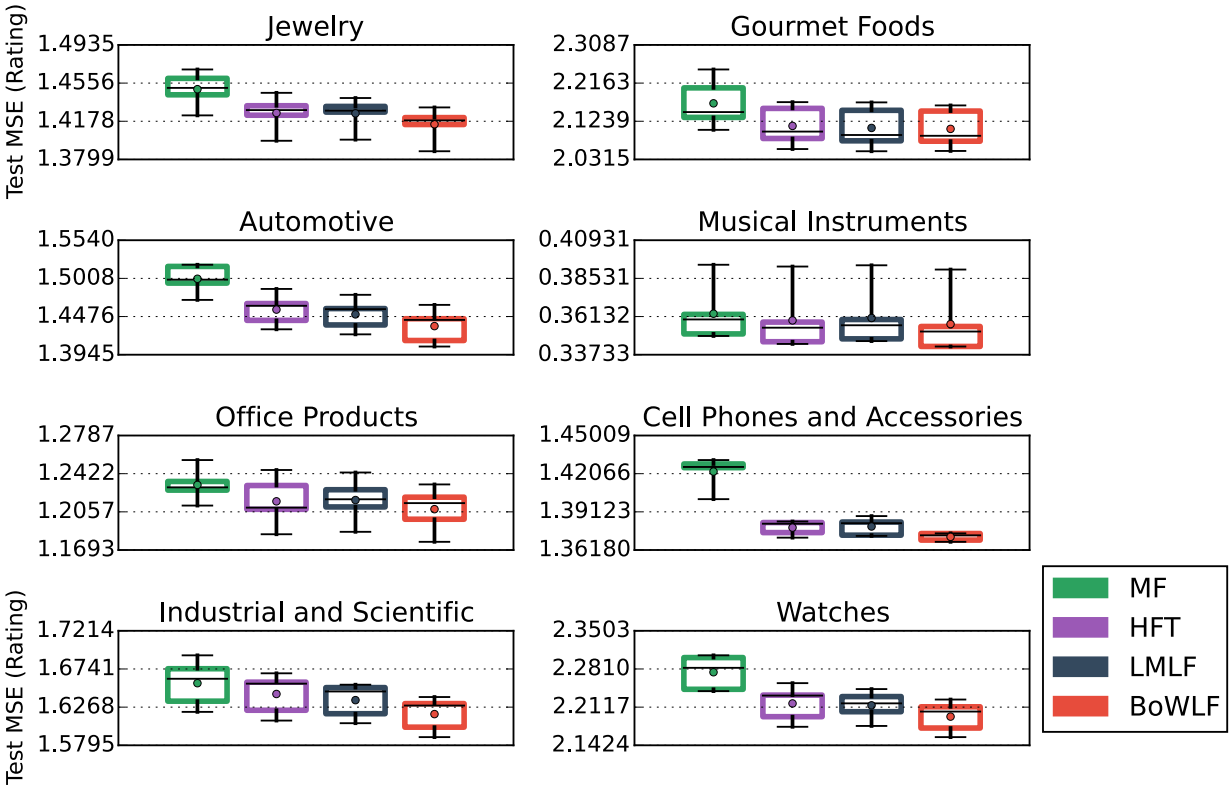


Fig. 3. Box and whisker plot showing K-fold ($K = 5$) experiments. Point represents the mean over all folds. Center line represents median. Box extents represent 25^{th} and 75^{th} percentile. Whisker extents show minimum and maximum values.

3.4.4. Impact of Training / Test Data Split

Comparing the results of the original HFT paper with the results we get training over the same split, it becomes clear that models trained on different splits are not directly comparable. To further explore the importance of the chosen split for model selection, we

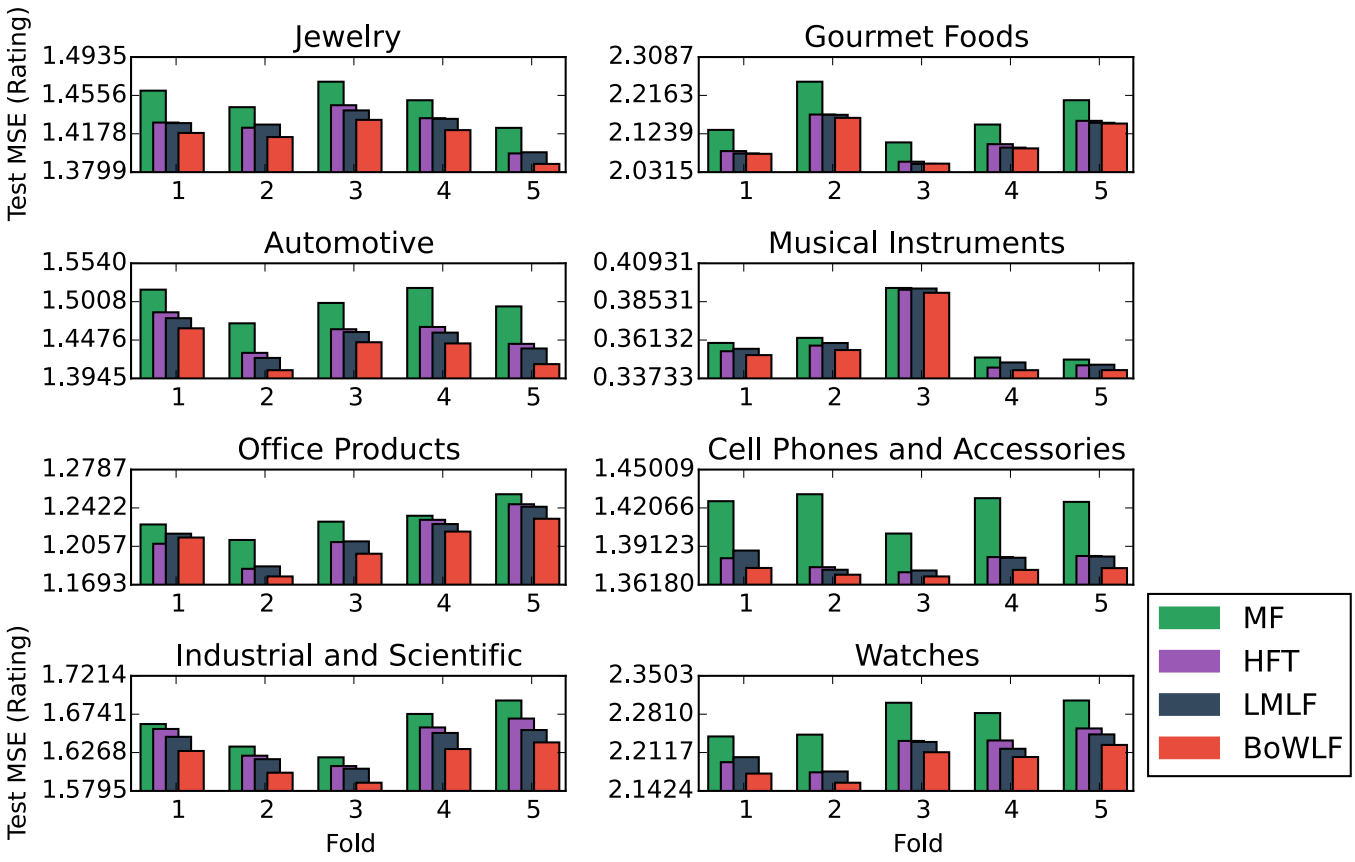


Fig. 4. Bar chart showing showing K-fold ($K = 5$) experiments. Although values across folds vary, relative performance is consistent.

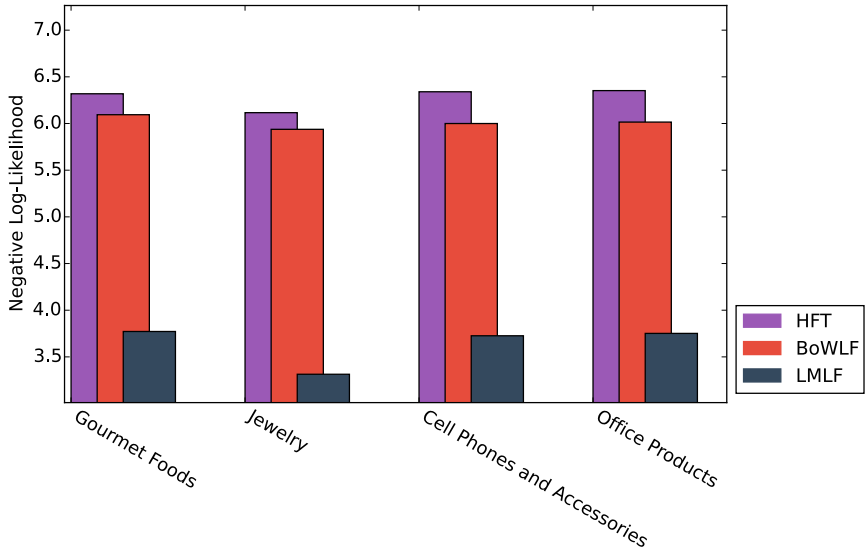


Fig. 5. Bar chart showing showing negative log-likelihood (NLL) on test data for several datasets. LMLF is superior in NLL but does not improve rating prediction over BoWLF.

perform experiments over five randomly selected folds and compare each model on every fold.

One of the challenges in pursuing empirical work on the Amazon review dataset is the current absence of a standard train / test split of the data. Here we evaluate the importance of establishing a standard data split.

Fig. 3 shows the results of experiments comparing the performance of each model over different splits. We perform 5-fold validation. That is, each model is trained 5 times, each on 80% of the data and we report performance on the remaining 20%. The result on the test reveal several important points. First, we note that the variance over splits can be large, meaning that comparing across different splits could be misleading when performing model selection. On the other hand, as shown in Fig 4, the relative performance of each model is consistent over the different splits. This implies that a single random split can be used for model selection and evaluation as long as this split is held constant over all evaluated models.[10]

Taken together, Figs. 3 and 4 illustrate the importance of standardizing the dataset splits. Without standardization, performance measured between different research groups becomes incomparable and the use of this dataset as a benchmark is rendered difficult or even impossible.

3.4.5. Effect of Language Model

One way to analyze models which use text information is to compare their negative log-likelihood (NLL) scores on the same test dataset. We find BoWLF has a stronger language model than HFT, which is reflected in the NLL results, and in this case it appears to contribute to a better rating prediction. As shown in Fig. 5, LMLF has a much better language model than both HFT and BoWLF, but as discussed earlier, LMLF *does not* lead to better rating predictions than BoWLF. LMLF appears to be largely equivalent to HFT in prediction strength, despite having a much better language model. As discussed above in Sec. 3.4.3, this suggests that the strong nonlinearity in the LSTM helps modeling reviews, but not necessarily result in the linearly-decodable product representation, leading to less improvement in rating prediction.

Contrary to LDA-based approaches, the latent dimensions of the product representations learned by BoWLF do not necessarily have clear interpretations as topics. However, the neighborhoods learned by BoWLF are interpretable, where we use the cosine distance between two product representations, i.e., $1 - \frac{\gamma_i^\top \gamma_j}{\|\gamma_i\|_2 \|\gamma_j\|_2}$. The BoWLF product space neighbors seem qualitatively superior to the neighbors given by HFT as seen in Table 2. Note in particular the association of "MTR Simply Tomato Soup" with other soups by BoWLF, while HFT neighbors seem much broader, including crackers, noodles, and gummy bears.

[10] Averaging results over multiple splits (shared by all models) would further reduce the variance in results.

This observation is consistent with the interpretation of the differences in the mathematical form of HFT and BoWLF (as argued in Sec. 3.3.4). The ability of BoWLF to form peakier distributions over words, given the product representation, allows the model to be more discriminating and more closely group similar products. Furthermore, we can see that the neighbors based on the product representations from the LMLF are qualitatively worse than those from the BoWLF, which indirectly confirms that the underlying product-related structure encoded by the LMLF is more difficult to extract linearly.

While drawing firm conclusions from this small set of neighbors is obviously ill-advised, the general trend appears to hold in more extensive testing. Broadly speaking, this further strengthens the idea that stronger product representations lead to improvements in rating prediction.

Product	HFT	BoWLF	LMLF
Extra Spearmint Sugarfree Gum	Hong Kong Fu Xiang Yuan Moon Cakes French Chew - Vanilla Peck's Anchovette	Dubble Bubble Gum Trident Sugarless White Gum Gold Mine Nugget Bubble Gum	Gumballs Special Assorted Bazooka Bubble Gum Gourmet Spicy Beef Jerky
Dark Chocolate Truffle	Tastykake Kreamies Kakes Ceam .. Miko - Awase Miso Soyabean Paste Haribo Berries Gummi Candy	Ritter Sport Corn Flakes Chocolate Chocolate Dobosh Torte Sugar Free, Milk Chocolate Pecan Turtles	Fantis Grape Leaves Grape Flavoring Tutti Fruitti Flavoring
MTR Simply Tomato Soup	Wellington Cracked Pepper Crackers Maggi Instant Noodles Haribo Gummi Candy	MTR Mulligatawny Soup hai Kitchen Coconut Ginger Soup Miko - Awase Miso Soyabean Paste	Muir Glen Organic Soup Soy Ginger Saba Noodles Alessi Soup

Table 2. Nearest neighbors (cosine similarity) based on product representations estimated by HFT, BoWLF and LMLF, for Gourmet Foods dataset. Qualitatively, the ability to regularize the product representations seems to correlate well with the quality of the neighbourhoods formed in product representation space.

3.5. Conclusion

We develop two new models (BoWLF and LMLF) which exploit text reviews to regularize rating prediction on the Amazon Reviews datasets. BoWLF achieves state of the art results on 27 of the 28 datasets, while LMLF outperforms HFT (but not BoWLF) as dataset size increases. Additionally, we explore the methodology behind the choice of data split, clearly demonstrating that models trained on different data subsets cannot be directly compared. Performing K-fold cross-validation ($K = 5$), we confirm that BoWLF achieves superior performance across dataset splits. The resulting product neighborhoods measured by cosine similarity between product representations are intuitive, and correspond with human analysis of the data. Overall we find that BoWLF has a 20.29% average improvement over basic matrix factorization and a 5.64% average improvement over HFT.

We found that the proposed LMLF slightly lagged behind the BoWLF. As we discuss above, we believe this could be due to the nonlinear nature of language model based on

a recurrent neural network. This nonlinearity results in the product-related structure underlying reviews being nonlinearly encoded in the product representation, which cannot be easily extracted by the linear rating prediction model. However, this will need to be further investigated in addition to analyzing the exact effect of language modeling on prediction performance.

Chapter 4

Prologue to Second Article

4.1. Article Details

Dynamic Capacity Networks. Amjad Almahairi, Nicolas Ballas, Tim Cooijmans, Yin Zheng, Hugo Larochelle and Aaron Courville. *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)*.

Personal Contribution. I am the main contributor to this work with regards to designing and carrying out experiments, analyzing results and writing the manuscript.

4.2. Context

Scaling up the size of learning models has been key to many remarkable successes of deep learning (Krizhevsky et al., 2012; Szegedy et al., 2014; He et al., 2016). In typical neural network architectures all parameters are "activated" for all regions of a given input, which leads to computational cost scaling quadratically in terms of data dimensionality and model parameters.

Conditional computation (Bengio et al., 2013) proposes to solve this problem by activating only a fraction of the model parameters for a given input sample, and thus allowing for increasing model size without proportional increase in computations. Bengio et al. (2013) proposed learning stochastic gating units (Bengio et al., 2013) to guide activating subsets of the model. This approach, however, did not yield substantial computational advantages due to several technical difficulties, such as effective branching in GPUs.

On the other hand, hard-attention models (Mnih et al., 2014; Xu et al., 2015b) have gained a lot popularity at the time of publishing this work. These models were shown to learn effective policies for deciding which portion of the input is worth focusing or "attending" to. One drawback of these models is that they were difficult to train. Our work was inspired by these papers to perform conditional computation with a simple, yet effective attention mechanism.

4.3. Contributions

This work introduces Dynamic Capacity Network (DCN), a novel conditional computation model, which can adaptively assign computation across different regions of the input. This is achieved by combining modules of two types: low-capacity sub-networks and high-capacity sub-networks. The low-capacity sub-networks are applied across most of the input, but also provide a guide to select a few portions of the input on which to apply the high-capacity sub-networks. The selection is made using an efficient, saliency-based attention mechanism, and can be trained end-to-end with back-propagation. The model achieves state-of-the-art results on Cluttered MNIST benchmark (Mnih et al., 2014), and can attain significant computational advantages over standard convolutional architectures. We apply the model on a real-world scenario of recognizing sequences of multi-digit house numbers in images from Street View House Numbers (SVHN) (Netzer et al., 2011), for which we assume almost no data pre-processing on test data.

4.4. Recent Developments

Our approach relies on a fixed heuristic for selecting salient input regions to focus computations on. While this attention mechanism can be effective, especially when the object of interest is localized within the input, it does not adapt to input data. Several follow-up papers focused on *learning* different strategies for controlling model capacity/computations (Bengio et al., 2015; Liu and Deng, 2017; McGill and Perona, 2017; Lin et al., 2017). Most notably, Lin et al. (2017) learns an agent that decides the importance of each convolutional kernel and performs channel-wise pruning depending on the input. Their approach was applied on several off-the-shelf convolutional models with compelling results.

While our work was more focused on CNNs, a lot of recent developments focused on reducing computational overhead in RNNs. Graves (2016) proposed Adaptive Computation Time (ACT) to dynamically decide the number of computation steps an RNN processes inputs at each time step. Figurnov et al. (2017) applied the same approach for dynamically deciding the depth of a Residual Network (He et al., 2016). Other works reduce computation in RNNs by updating only a subset of the hidden state at each time step (Jernite et al., 2016), or skipping the hidden state entirely and thus reducing the effective depth of the RNNs computational graph (Campos et al., 2018).

Shazeer et al. (2017) propose a method which can significantly increase the number of parameters in a single layer while keeping computation on par with (or even less than) than in standard architectures. The idea is based on using a large mixture of experts (small networks), where only a few of them are activated via a gating network. Their work suits well modern GPU clusters, and achieves impressive speedups and results in language modelling and machine translation.

Chapter 5

Dynamic Capacity Networks

5.1. Introduction

Deep neural networks have recently exhibited state-of-the-art performance across a wide range of tasks, including object recognition (Szegedy et al., 2014) and speech recognition (Graves and Jaitly, 2014). Top-performing systems, however, are based on very deep and wide networks that are computationally intensive. One underlying assumption of many deep models is that all input regions contain the same amount of information. Indeed, convolutional neural networks apply the same set of filters uniformly across the spatial input (Szegedy et al., 2014), while recurrent neural networks apply the same transformation at every time step (Graves and Jaitly, 2014). Those networks lead to time-consuming training and inference (prediction), in large part because they require a large number of weight/activation multiplications.

Task-relevant information, however, is often not uniformly distributed across input data. For example, objects in images are spatially localized, i.e. they exist only in specific regions of the image. This observation has been exploited in *attention-based* systems (Mnih et al., 2014), which can reduce computations significantly by learning to selectively focus or "attend" to few, task-relevant, input regions. Attention employed in such systems is often referred to as "hard-attention", as opposed to "soft-attention" which integrates smoothly all input regions. Models of hard-attention proposed so far, however, require defining an explicit predictive model, whose training can pose challenges due to its non-differentiable cost.

In this work we introduce the *Dynamic Capacity Network* (DCN) that can adaptively assign its capacity across different portions of the input, using a gradient-based hard-attention process. The DCN combines two types of modules: small, low-capacity sub-networks, and large, high-capacity sub-networks. The low-capacity sub-networks are active on the whole input, but are used to direct the high-capacity sub-networks, via our attention mechanism, to task-relevant regions of the input.

A key property of the DCN's hard-attention mechanism is that it *does not* require a policy network trained by reinforcement learning. Instead, we can train DCNs end-to-end with backpropagation. We evaluate a DCN model on the attention benchmark task Cluttered MNIST (Mnih et al., 2014), and show that it outperforms the state of the art.

In addition, we show that the DCN's attention mechanism can deal with situations where it is difficult to learn a task-specific attention policy due to the lack of appropriate data. This is often the case when training data is mostly canonicalized, while at test-time the system is effectively required to perform transfer learning and deal with substantially different, noisy real-world images. The Street View House Numbers (SVHN) dataset (Netzer et al., 2011) is an example of such a dataset. The task here is to recognize multi-digit sequences from real-world pictures of house fronts; however, most digit sequences in training images are well-centred and tightly cropped, while digit sequences of test images are surrounded by large and cluttered backgrounds. Learning an attention policy that focuses only on a small portion of the input can be challenging in this case, unless test images are pre-processed to deal with this discrepancy [1]. DCNs, on the other hand, can be leveraged in such transfer learning scenarios, where we learn low and high capacity modules independently and only combine them using our attention mechanism at test-time. In particular, we show that a DCN model is able to efficiently recognize multi-digit sequences, directly from the original images, without using any prior information on the location of the digits.

Finally, we show that DCNs can perform efficient region selection, in both Cluttered MNIST and SVHN, which leads to significant computational advantages over standard convolutional models.

5.2. Dynamic Capacity Networks

In this section, we describe the Dynamic Capacity Network (DCN) that dynamically distributes its network capacity across an input.

We consider a deep neural network h, which we decompose into two parts: $h(\mathbf{x}) = g(f(\mathbf{x}))$ where f and g represent respectively the *bottom layers* and *top layers* of the network h while $\mathbf{x}$ is some input data. Bottom layers f operate directly on the input and output a representation, which is composed of a collection of vectors each of which represents a region in the input. For example, f can output a feature map, i.e. vectors of features each with a specific spatial location, or a probability map outputting probability distributions at each different spatial location. Top layers g consider as input the bottom layers' representations $f(x)$ and output a distribution over labels.

DCN introduces the use of two alternative sub-networks for the bottom layers f: the *coarse layers* f_c or the *fine layers* f_f, which differ in their capacity. The fine layers correspond

[1] This is the common practice in previous work on this dataset, e.g, (Goodfellow et al., 2013; Ba et al., 2014; Jaderberg et al., 2015)

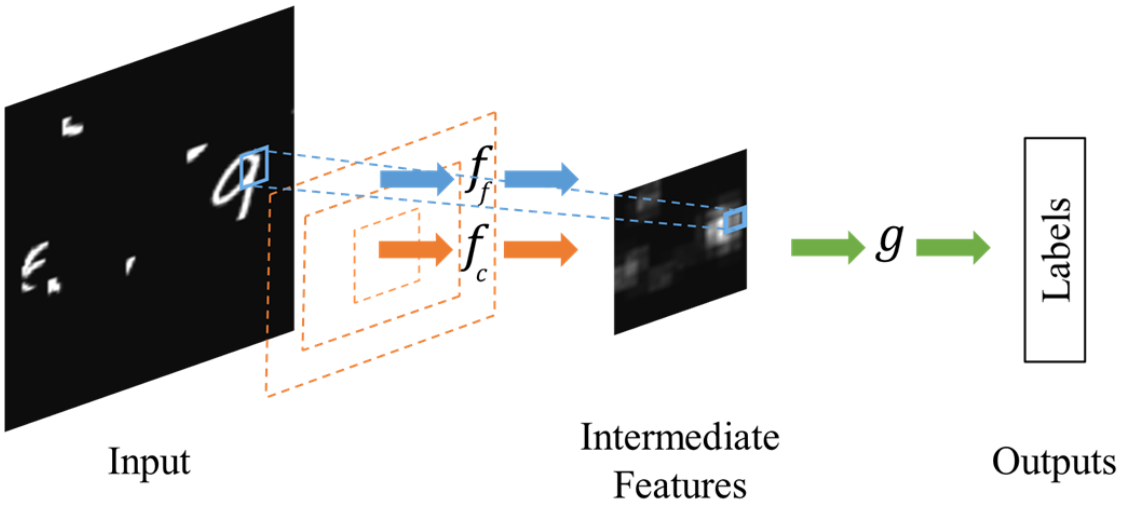


Fig. 1. DCN overview. Our model applies the coarse layers on the whole image to get $f_c(\mathbf{x})$, chooses a set of salient patches $\mathbf{X}^s$, applies the fine layers only on the salient patches $\mathbf{X}^s$ to obtain a set of few fine representation vectors $f_f(\mathbf{X}^s)$, and finally combines them to make its prediction.

to a high-capacity sub-network which has a high-computational requirement, while the coarse layers constitute a low-capacity sub-network. Consider applying the top layers only on the fine representation, i.e. $h_f(\mathbf{x}) = g(f_f(\mathbf{x}))$. We refer to the composition $h_f = g \circ f_f$ as the *fine model*. We assume that the fine model can achieve very good performance, but is computationally expensive. Alternatively, consider applying the top layers only on the coarse representation, i.e. $h_c(\mathbf{x}) = g(f_c(\mathbf{x}))$. We refer to this composition $h_c = g \circ f_c$ as the *coarse model*. Conceptually, the coarse model can be much more computationally efficient, but is expected to have worse performance than the fine model.

The key idea behind DCN is to have g use representations from either the coarse or fine layers in an adaptive, dynamic way. Specifically, we apply the coarse layers f_c on the whole input $\mathbf{x}$, and leverage the fine layers f_f only at a few "important" input regions. This way, the DCN can leverage the capacity of f_f, but at a lower computational cost, by applying the fine layers only on a small portion of the input. To achieve this, DCN requires an attentional mechanism, whose task is to identify good input locations on which to apply f_f. In the remainder of this section, we focus on 2-dimensional inputs. However, our DCN model can be easily extended to be applied to any type of N-dimensional data.

5.2.1. Attention-based Inference

In DCN, we would like to obtain better predictions than those made by the coarse model f_c while keeping the computational requirement reasonable. This can be done by selecting a few *salient* input regions on which we use the fine representations instead of the coarse ones. DCN inference therefore needs to identify the important regions in the input with respect to

the task at hand. For this, we use a novel approach for attention that uses backpropagation in the coarse model h_c to identify *few vectors in the coarse representation* to which the distribution over the class label is most sensitive. These vectors correspond to input regions which we identify as *salient* or task-relevant.

Given an input image $\mathbf{x}$, we first apply the coarse layers on all input regions to compute the coarse representation vectors:

$$f_c(\mathbf{x}) = \{\mathbf{c}_{i,j} \mid (i,j) \in [1, s_1] \times [1, s_2]\}, \tag{5.2.1}$$

where s_1 and s_2 are spatial dimensions that depend on the image size, and $\mathbf{c}_{i,j} = f_c(\mathbf{x}_{i,j}) \in \mathbb{R}^D$ is a representation vector associated with the input region (i,j) in $\mathbf{x}$, i.e. corresponds to a specific receptive field or a patch in the input image. We then compute the output of the model based completely on the coarse vectors, i.e. the coarse model's output $h_c(\mathbf{x}) = g(f_c(\mathbf{x}))$.

Next, we identify a few *salient* input regions using an attentional mechanism that exploits a *saliency map* generated using the coarse model's output. The specific measure of saliency we choose is based on the *entropy* of the coarse model's output, defined as:

$$H = -\sum_{l=1}^{C} \mathbf{o}_c^{(l)} \log \mathbf{o}_c^{(l)}, \tag{5.2.2}$$

where we denote $\mathbf{o}_c = h_c(\mathbf{x})$ as the vector output of the coarse model and C is the number of class labels. The saliency $\mathbf{M}$ of an input region position (i,j) is given by the norm of the gradient of the entropy H with respect to the coarse vector $\mathbf{c}_{i,j}$:

$$M_{i,j} = ||\nabla_{\mathbf{c}_{i,j}} H||_2 = \sqrt{\sum_{r=1}^{D} \left(\frac{\partial}{\partial \mathbf{c}_{i,j}^{(r)}} - \sum_{l=1}^{C} \mathbf{o}_c^{(l)} \log \mathbf{o}_c^{(l)} \right)^2}, \tag{5.2.3}$$

where $\mathbf{M} \in \mathbb{R}^{s_1 \times s_2}$. The use of the entropy gradient as a saliency measure encourages selecting input regions that could affect the uncertainty in the model's predictions the most. In addition, computing the entropy of the output distribution does not require observing the true label, hence the measure is available at inference time. Note that computing all entries in matrix $\mathbf{M}$ can be done using a single backward pass of backpropagation through the top layers and is thus efficient and simple to implement.

Using the saliency map $\mathbf{M}$, we select a set of k input region positions with the highest saliency values. We denote the selected set of positions by $\mathbf{I}^s \subseteq [1, s_1] \times [1, s_2]$, such that $|\mathbf{I}^s| = k$. We denote the set of selected input regions by $\mathbf{X}^s = \{\mathbf{x}_{i,j} \mid (i,j) \in \mathbf{I}^s\}$ where each $\mathbf{x}_{i,j}$ is a patch in $\mathbf{x}$. Next we apply the fine layers f_f *only on the selected patches* and obtain a small set of fine representation vectors:

$$f_f(\mathbf{X}^s) = \{\mathbf{f}_{i,j} \mid (i,j) \in \mathbf{I}^s\}, \tag{5.2.4}$$

where $\mathbf{f}_{i,j} = f_f(\mathbf{x}_{i,j})$. This requires that $\mathbf{f}_{i,j} \in \mathbb{R}^D$, i.e. the fine vectors have the same dimensionality as the coarse vectors, allowing the model to use both of them interchangeably.

We denote the representation resulting from combining vectors from both $f_c(\mathbf{x})$ and $f_f(\mathbf{X}^s)$ as the *refined representation* $f_r(\mathbf{x})$. We discuss in Section 5.4 different ways in which they can be combined in practice. Finally, the DCN output is obtained by feeding the refined representation into the top layers, $g(f_r(\mathbf{x}))$. We denote the composition $g \circ f_r$ by the *refined model*.

5.2.2. End-to-End Training

In this section, we describe an end-to-end procedure for training the DCN model that leverages our attention mechanism to learn f_f and f_c jointly. We emphasize, however, that DCN modules can be trained independently, by training a coarse and a fine model independently and combining them only at test-time using our attention based inference. In Section 5.4.2 we show an example of how this modular training can be used for transfer learning.

In the context of image classification, suppose we have a training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}); i = 1 \ldots m\}$, where each $\mathbf{x}^{(i)} \in \mathbb{R}^{h \times w}$ is an image, and $y^{(i)} \in \{1, \ldots, C\}$ is its corresponding label. We denote the parameters of the coarse, fine and top layers by θ_c, θ_f, and θ_t respectively. We learn all of these parameters (denoted as θ) by minimizing the cross-entropy objective function (which is equivalent to maximizing the log-likelihood of the correct labels):

$$J = -\sum_{i=1}^{m} \log p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \theta\right), \tag{5.2.5}$$

where $p(\cdot \mid \mathbf{x}^{(i)}; \theta) = g(f_r(\mathbf{x}^{(i)}))$ is the conditional multinomial distribution defined over the C labels given by the refined model (Figure 1). Gradients are computed by standard back-propagation through the refined model, i.e. propagating gradients at each position into either the coarse or fine features, depending on which was used.

An important aspect of the DCN model is that the final prediction is based on combining representations from two different sets of layers, namely the coarse layers f_c and the fine layers f_f. Intuitively, we would like those representations to have close values such that they can be interchangeable. This is important for two reasons. First, we expect the top layers to have more success in correctly classifying the input if the transition from coarse to fine representations is smooth. The second is that, since the saliency map is based on the gradient *at the coarse representation values* and since the gradient is a local measure of variation, it is less likely to reflect the benefit of using the fine features if the latter is very different from the former.

To encourage similarity between the coarse and fine representations while training, we use a hint-based training approach inspired by Romero et al. (2014). Specifically, we add

an additional term to the training objective that minimizes the squared distance between coarse and fine representations:

$$\sum_{\mathbf{x}_{i,j} \in \mathbf{X}^s} \|f_c(\mathbf{x}_{i,j}) - f_f(\mathbf{x}_{i,j})\|_2^2. \tag{5.2.6}$$

There are two important points to note here. First, we use this term to optimize *only the coarse layers* θ_c. That is, we encourage the coarse layers to mimic the fine ones, and let the fine layers focus only on the signal coming from the top layers. Secondly, computing the above *hint* objective over representations at all positions would be as expensive as computing the full fine model; therefore, we encourage in this term similarity only over the selected salient patches.

5.3. Related Work

This work can be classified as a conditional computation approach. The goal of conditional computation, as put forward by Bengio (2013), is to train very large models for the same computational cost as smaller ones, by avoiding certain computation paths depending on the input. There have been several contributions in this direction. Bengio et al. (2013) use stochastic neurons as gating units that activate specific parts of a neural network. Our approach, on the other hand, uses a hard-attention mechanism that helps the model to focus its computationally expensive paths only on important input regions, which helps in both scaling to larger effective models and larger input sizes.

Several recent contributions use attention mechanisms to capture visual structure with biologically inspired, foveation-like methods, e.g. (Larochelle and Hinton, 2010; Denil et al., 2012; Ranzato, 2014; Mnih et al., 2014; Ba et al., 2014; Gregor et al., 2015). In Mnih et al. (2014); Ba et al. (2014), a learned sequential attention model is used to make a hard decision as to where to look in the image, i.e. which region of the image is considered in each time step. This so-called "hard-attention" mechanism can reduce computation for inference. The attention mechanism is trained by reinforcement learning using policy search. In practice, this approach can be computationally expensive during training, due to the need to sample multiple interaction sequences with the environment. On the other hand, the DRAW model (Gregor et al., 2015) uses a "soft-attention" mechanism that is fully differentiable, but requires processing the whole input at each time step. Our approach provides a simpler hard-attention mechanism with computational advantages in both inference and learning.

The saliency measure employed by DCN's attention mechanism is related to pixel-wise saliency measures used in visualizing neural networks (Simonyan et al., 2013). These measures, however, are based on the gradient of the classification loss, which is not applicable at test-time. Moreover, our saliency measure is defined over contiguous regions of the input

rather than on individual pixels. It is also task-dependent, as a result of defining it using a coarse model trained on the same task.

Other works such as matrix factorization (Jaderberg et al., 2014; Denton et al., 2014) and quantization schemes (Chen et al., 2010; Jégou et al., 2011; Gong et al., 2014) take the same computational shortcuts for all instances of the data. In contrast, the shortcuts taken by DCN specialize to the input, avoiding costly computation except where needed. However, the two approaches are orthogonal and could be combined to yield further savings.

Our use of a regression cost for enforcing representations to be similar is related to previous work on model compression (Bucilua et al., 2006; Hinton et al., 2015; Romero et al., 2014). The goal of model compression is to train a small model (which is faster in deployment) to imitate a much larger model or an ensemble of models. Furthermore, Romero et al. (2014) have shown that middle layer hints can improve learning in deep and thin neural networks. Our DCN model can be interpreted as performing model compression on the fly, without the need to train a large model up front.

5.4. Experiments

In this section, we present an experimental evaluation of the proposed DCN model. To validate the effectiveness of our approach, we first investigate the Cluttered MNIST dataset (Mnih et al., 2014). We then apply our model in a transfer learning setting to a real-world object recognition task using the Street View House Numbers (SVHN) dataset (Netzer et al., 2011).

5.4.1. Cluttered MNIST

We use the 100×100 Cluttered MNIST digit classification dataset (Mnih et al., 2014). Each image in this dataset is a hand-written MNIST digit located randomly on a 100×100 black canvas and cluttered with digit-like fragments. Therefore, the dataset has the same size of MNIST: 60000 images for training and 10000 for testing.

5.4.1.1. *Model Specification*

In this experiment we train a DCN model end-to-end, where we learn coarse and fine layers jointly. We use 2 convolutional layers as coarse layers, 5 convolutional layers as fine layers and one convolutional layer followed by global max pooling and a softmax as the top layers. The coarse and fine layers produce feature maps, i.e. feature vectors each with a specific spatial location. The set of selected patches $\mathbf{X}^s$ is composed of eight patches of size 14×14 pixels. We use here a refined representation of the full input $f_r(\mathbf{x})$ in which fine feature vectors are swapped in place of coarse ones:

Model	Test Error
RAM	8.11%
DRAW	3.36%
Coarse Model	3.69%
Fine Model	1.70%
DCN w/o hints	1.71%
DCN with hints	**1.39**%

Table 1. Results on Cluttered MNIST

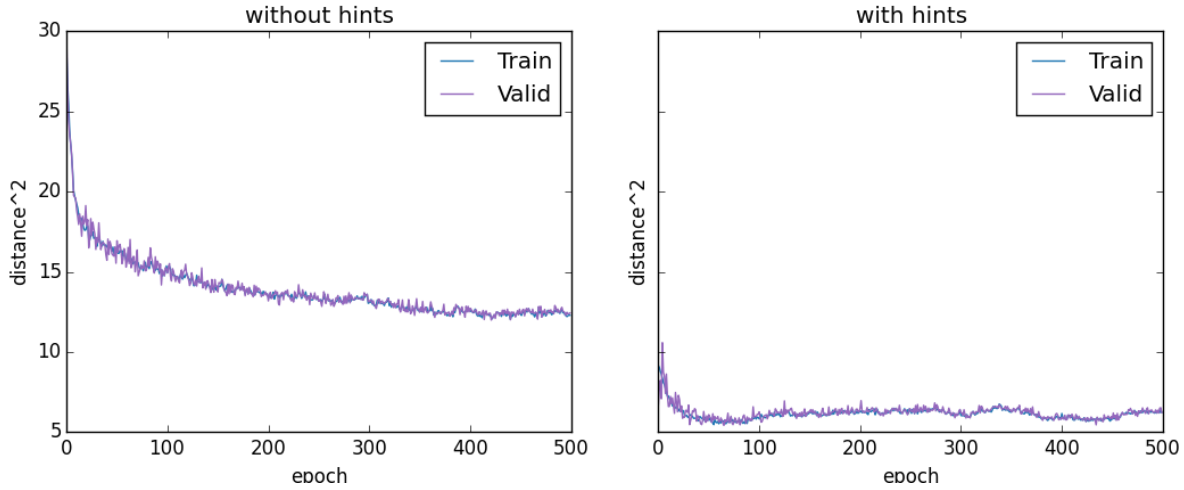


Fig. 2. The effect of using the hints objective. We show the squared distance between coarse and fine features over salient regions during training in two cases: with and without using the hints objective. We observe that this regularizer helps in minimizing the distance and improves the model's generalization.

$$f_r(\mathbf{x}) = \{\mathbf{r}_{i,j} \mid (i,j) \in [1, s_1] \times [1, s_2]\} \tag{5.4.1}$$

$$\mathbf{r}_{i,j} = \begin{cases} f_f(\mathbf{x}_{i,j}), & \text{if } \mathbf{x}_{i,j} \in \mathbf{X}^s \\ f_c(\mathbf{x}_{i,j}), & \text{otherwise.} \end{cases} \tag{5.4.2}$$

5.4.1.2. *Baselines*

We use as baselines for our evaluation the coarse model (top layers applied only on coarse representations), the fine model (top layers applied only on fine representations), and we compare with previous attention-based models RAM (Mnih et al., 2014) and DRAW (Gregor et al., 2015).

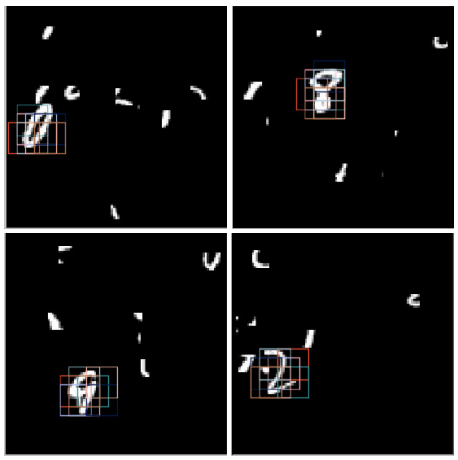


Fig. 3. Sample of selected patches in Cluttered MNIST

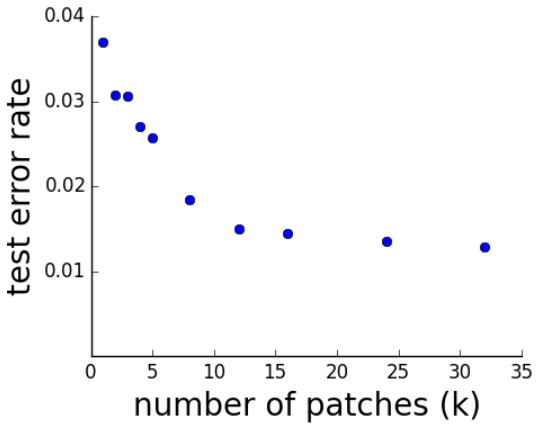


Fig. 4. Test error vs. number of selected patches: taking more patches yields lower error, but with diminishing returns.

5.4.1.3. *Empirical Evaluation*

Results of our experiments are shown in Table 1. We get our best DCN result when we add the hint term in Eq. (5.2.6) in the training objective, which we observe to have a regularization effect on DCN. We can see that the DCN model performs significantly better than the previous state-of-the-art result achieved by RAM and DRAW models. It also outperforms the fine model, which is a result of being able to focus only on the digit and ignore clutter. In Figure 2 we explore more the effect of the hint objective during training, and confirm that it can indeed minimize the squared distance between coarse and fine representations. To show how the attention mechanism of the DCN model can help it focus on the digit, we plot in Figure 3 the patches it finds in some images from the validation set, after only 9 epochs of training.

The DCN model is also more computationally efficient. A forward pass of the fine model requires the computation of the fine layers representations on whole inputs and a forward pass of the top layers leading to 84.5M multiplications. On the other hand, DCN applies only the coarse layers on the whole input. It also requires the computation of the fine representations for 8 input patches and a forward pass of the top layers. The attention mechanism of the DCN model requires an additional forward and backward pass through the top layers which leads to approximately 27.7M multiplications in total. As a result, the DCN model here has 3 times fewer multiplications than the fine model. In practice we observed a time speed-up by a factor of about 2.9. Figure 4 shows how the test error behaves when we increase the number of patches. While taking additional patches improves accuracy, the marginal improvement becomes insignificant beyond 10 or so patches. The number of patches effectively controls a trade-off between accuracy and computational cost.

5.4.2. SVHN

We tackle in this section a more challenging task of transcribing multi-digit sequences from natural images using the Street View House Numbers (SVHN) dataset (Netzer et al., 2011). SVHN is composed of real-world pictures containing house numbers and taken from house fronts. The task is to recognize the full digit sequence corresponding to a house number, which can be of length 1 to 5 digits. The dataset has three subsets: train (33k), extra (202k) and test (13k). In the following, we trained our models on 230k images from both the train and extra subsets, where we take a 5k random sample as a validation set for choosing hyper-parameters.

The typical experimental setting in previous literature, e.g. (Goodfellow et al., 2013; Ba et al., 2014; Jaderberg et al., 2015), uses the location of digit bounding boxes as extra information. Input images are generally cropped, such that digit sequences are centred and most of the background and clutter information is pruned. We argue that our DCN model can deal effectively with real-world noisy images having large portions of clutter or background information. To demonstrate this ability, we investigate a more general problem setting where the images are uncropped and the digits locations are unknown. We apply our models on SVHN images *in their original sizes, and without any extra bounding box information.* [2]

An important property of the SVHN dataset is the large discrepancy between the train/extra sets and the test set. Most of the extra subset images (which dominate the training data) have their digits well-centred with little cluttered background, while test images have more variety in terms of digit location and background clutter. Figure 5 shows samples of these images. We can tackle this training/test dataset discrepancy by training a DCN model in a transfer learning setting. We train the coarse and fine layers of the DCN independently on the training images that have little background-clutter, and then combine them using our attention mechanism, which does not require explicit training, to decide on which subsets of the input to apply the fine layers.

5.4.2.1. *Multi-Digit Recognition Model*

We follow the model proposed in (Goodfellow et al., 2013) for learning a probabilistic model of the digit sequence given an input image $\mathbf{x}$. The output sequence $\mathbf{S}$ is defined using a collection of N random variables, $S_1, \ldots, S_N$, representing the elements of the sequence and an extra random variable S_0 representing its length. The probability of a given sequence $\mathbf{s} = \{s_1, \ldots, s_n\}$ is given by:

$$p(\mathbf{S} = \mathbf{s} \mid \mathbf{x}) = p(S_0 = n \mid \mathbf{x}) \prod_{i=1}^{n} p(S_i = s_i \mid \mathbf{x}), \tag{5.4.3}$$

[2]The only pre-processing we perform on the data is converting images to grayscale.

Fig. 5. The 4 left images are samples from the extra subset, and the 4 right images are samples from the test subset. We notice that extra images are well-centred and have much less background compared to test images.

where $p(S_0 = n \mid \mathbf{x})$ is the conditional distribution of the sequence length and $p(S_i = s_i \mid \mathbf{x})$ is the conditional distribution of the i-th digit in the sequence. In particular, our model on SVHN has 6 softmaxes: 1 for the length of the sequence (from 1 to 5), and 5 for the identity of each digit or a null character if no digit is present (11 categories). Note that while the conditional independence assumption is valid for modelling sequences of digits in house numbers, an alternative, and probably more general approach, would be to model the probability of sequences with an RNN, which does not rely on any independence assumption.

5.4.2.2. *Model Specification*

The coarse and fine bottom layers, f_c and f_f, are fully-convolutional, composed of respectively 7 and 11 layers. The representation, produced by either the fine or coarse layers, is a *probability map*, which is a collection of independent full-sequence prediction vectors, each vector corresponding to a specific region of the input. We denote the prediction for the i-th output at position (j,k) by $p^{(j,k)}(S_i \mid \mathbf{x})$.

The top layer g is composed of one global average pooling layer which combines predictions from various spatial locations to produce the final prediction $p(\mathbf{S} \mid \mathbf{x})$.

Since we have multiple outputs in this task, we modify the saliency measure used by the DCN's attention mechanism to be the sum of the entropy of the 5 digit softmaxes:

$$H = -\sum_{i=1}^{5}\sum_{j=1}^{11} p(S_i = s_j \mid \mathbf{x}) \log p(S_i = s_j \mid \mathbf{x}). \tag{5.4.4}$$

When constructing the saliency, instead of using the gradient with respect to the probability map, we use the gradient with respect to the feature map below it. This is necessary to avoid identical gradients as g, the top function, is composed by only one average pooling.

We also use a refined model that computes its output by applying the pooling top layer g only on the k independent predictions from fine layers, ignoring the coarse layers. We have found empirically that this results in a better model, and suspect that otherwise the predictions from the salient regions are drowned out by the noisy predictions from uninformative regions.

We train the coarse and fine layers of DCN independently in this experiment, minimizing $\log p(\mathbf{S} \mid \mathbf{x})$ using SGD. For the purposes of training only, we resize images to 64×128.

5.4.2.3. *Baselines*

As mentioned in the previous section, each of the coarse representation vectors in this experiment corresponds to multi-digit recognition probabilities computed at a given region, which the top layer g simply averages to obtain the baseline coarse model:

$$p(S_i \mid \mathbf{x}) = \frac{1}{d_1 \times d_2} \sum_{j,k} p^{(j,k)}(S_i \mid \mathbf{x}). \tag{5.4.5}$$

The baseline fine model is defined similarly.

As an additional baseline, we consider a "soft-attention" coarse model, which takes the coarse representation vectors over all input regions, but uses a top layer that performs a weighted average of the resulting location-specific predictions. We leverage the entropy to define a weighting scheme which emphasizes important locations:

$$p(S_i \mid \mathbf{x}) = \sum_{j,k} w_{i,j,k} p^{(j,k)}(S_i \mid \mathbf{x}). \tag{5.4.6}$$

The weight $w_{i,j,k}$ is defined as the *normalized inverse entropy* of the i-th prediction by the (j,k)-th vector, i.e. :

$$w_{i,j,k} = \sum_{q,r} \frac{H_{i,j,k}^{-1}}{H_{i,q,r}^{-1}}, \tag{5.4.7}$$

where $H_{i,j,k}$ is defined as:

$$H_{i,j,k} = -\sum_{l=1}^{C} p^{j,k}(S_i = s_l \mid \mathbf{x}) \log p^{j,k}(S_i = s_l \mid \mathbf{x}), \tag{5.4.8}$$

and C is either 5 for S_0 or 11 for all other S_i. As we will see, this weighting improves the coarse model's performance in our SVHN experiments. We incorporate this weighting in DCN to aggregate predictions from the salient regions.

To address scale variations in the data, we extend all models to multi-scale by processing each image several times at multiple resolutions. Predictions made at different scales are considered independent and averaged to produce the final prediction.

Model	Test Error
Coarse model, 1 scale	40.6%
Coarse model, 2 scales	40.0%
Coarse model, 3 scales	40.0%
Fine model, 1 scale	25.2%
Fine model, 2 scales	23.7%
Fine model, 3 scales	23.3%
Soft-attention, 1 scale	31.4%
Soft-attention, 2 scales	31.1%
Soft-attention, 3 scales	30.8%
DCN, 6 patches, 1 scale	20.0%
DCN, 6 patches, 2 scales	18.2%
DCN, 9 patches, 3 scales	16.6%

Table 2. Results on SVHN dataset without using bounding box information.

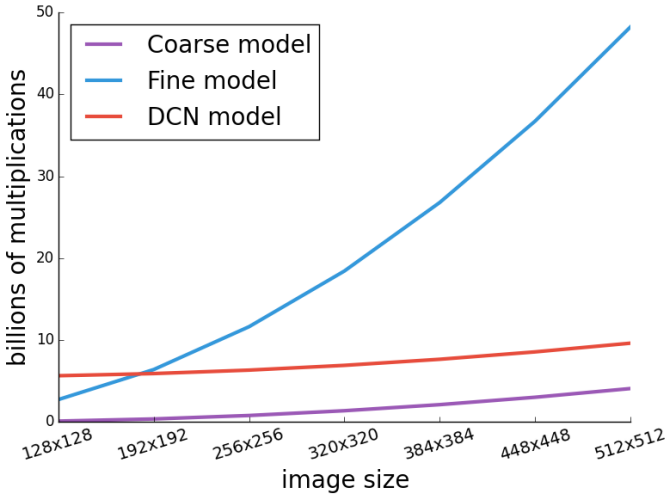


Fig. 6. Number of multiplications of the coarse, fine and DCN architectures on SVHN experiment given different image input sizes.

It is worth noting that all previous literature on SVHN dealt with a simpler task where images are cropped and resized. In this experiment we deal with a more general setting, and our results cannot be directly compared with these results.

5.4.2.4. *Empirical Evaluation*

Table 2 shows results of our experiment on SVHN. The coarse model has an error rate of 40.6%, while by using our proposed soft-attention mechanism, we decrease the error rate to 31.4%. This confirms that the entropy is a good measure for identifying important regions when task-relevant information is not uniformly distributed across input data.

Fig. 7. A sample of the selected patches in SVHN images. The images are processed by the DCN inference procedure in their original sizes. They are resized here for illustration purposes.

The fine model, on the other hand, achieves a better error rate of 25.2%, but is more computationally expensive. Our DCN model, which selects only 6 regions on which to apply the high-capacity fine layers, achieves an error rate of 20.0%. The DCN model can therefore outperform, in terms of classification accuracy, the other baselines. This verifies our assumption that by applying high capacity sub-networks only on the input's most informative regions, we are able to obtain high classification performance. Figure 7 shows a sample of the selected patches by our attention mechanism.

An additional decrease of the test errors can be obtained by increasing the number of processed scales. In the DCN model, taking 3 patches at 2 scales (original and 0.75 scales), leads to 18.2% error, while taking 3 patches at 3 scales (original, 0.75 and 0.5 scales) leads to an error rate of 16.6%. Our DCN model can reach its best performance of 11.6% by taking all possible patches at 3 scales, but it does not offer any computational benefit over the fine model.

We also investigate the computational benefits of the DCN approach as the dimensions of the input data increase. Figure 6 reports the total number of multiplications as a function of the image size, which each of the fine, coarse and DCN models incur. We compute these numbers based on the architectures we use in the SVHN experiment. We can see that as the input size increases, the DCN's number of multiplications grows at the same rate as the coarse model, because the number of computations devoted to the "high-capacity" modules remains constant as the input size grows. We also verify the actual computational time of these models by taking the largest 100 images in the SVHN test set, and computing the average inference time taken by all the models. [3] The smallest of these images has a size of 363×735 pixels, while the largest has a size of 442×1083 pixels. On average, the coarse and the soft-attention models take 8.6 milliseconds, while the fine model takes 62.6 milliseconds.

[3]We evaluate all models on an NVIDIA Titan Black GPU card.

On the largest 100 SVHN test images, the DCN requires on average 10.8 milliseconds for inference.

5.5. Conclusion

We have presented the DCN model, which is a novel approach for conditional computation. We have shown that using our visual attention mechanism, our network can adaptively assign its capacity across different portions of the input data, focusing on important regions of the input. Our model achieved state-of-the-art performance on the Cluttered MNIST digit classification task, and provided computational benefits over traditional convolutional network architectures. We have also validated our model in a transfer learning setting using the SVHN dataset, where we tackled the multi-digit recognition problem without using any a priori information on the digits' location. We have shown that our model outperforms other baselines, yet remains tractable for inputs with large spatial dimensions.

Chapter 6

Prologue to Third Article

6.1. Article Details

Calibrating Energy-based Generative Adversarial Networks. Zihang Dai, Amjad Almahairi, Philip Bachman, Eduard Hovy and Aaron Courville. *Proceedings of the 4th International Conference on Learning Representations (ICLR 2017).*

Personal Contribution. Zihang Dai wrote the mathematical analysis of this work, and both of us worked jointly on the implementation, including the parametric instantiations of Sec. 7.4, conducting experiments and analyzing results. I co-wrote the manuscript with the other authors.

6.2. Context

This work was motivated by the objective of learning a general-purpose probabilistic generative model, which not only can generate compelling samples efficiently, but also produce point-wise likelihood estimates which can be used as a data-driven evaluation metric. Our work is inspired by the work of Kim and Bengio (2016), which also proposed using generative adversarial networks (GANs) in learning energy-based models.

GANs have proven to be an expressive class of probabilistic generative models which can produce state-of-the-art samples, especially in image generation tasks. Nevertheless, they lack the capability of producing point-wise likelihood estimates, or even unnormalized energy values.

6.3. Contributions

We propose equipping GANs with the ability to produce direct energy estimates for samples. Specifically, we develop a flexible adversarial training framework, and prove this framework not only ensures the generator converges to the true data distribution, but also enables the discriminator to retain the density information at the global optimum. We derive the analytic form of the induced solution, and analyze its properties. As it turns out, our

framework is based on two key ideas: (i) using a real-valued discriminator (ii) adding a regularization term that maximizes the entropy of the generator's distribution. In order to make the proposed framework trainable in practice, we introduce two approximation techniques for entropy maximization of generator's distribution. Empirically, the experiment results closely match our theoretical analysis, verifying that the discriminator is able to recover the energy of data distribution. In addition, we show that our framework can produce state-of-the-art samples.

6.4. Recent Developments

Our proposed framework tackles one of the limitations of GANs with respect to providing point-wise likelihood estimates. GANs remain, as of the writing of this thesis, one of the most active research areas in machine learning.

One of the important contributions of this work is providing practical methods for maximizing entropy of generator distribution. These methods have been shown to be effective in other GAN frameworks, such as StackGAN (Huang et al., 2017) which produces state-of-the-art conditional generation samples. In addition, Dai et al. (2017) show that it can very effective in semi-supervised learning settings with GANs. A more recent work (Kumar et al., 2019) proposed using mutual information estimators to maximize the generator's entropy.

Another important limitation of standard GAN framework is the lack of an inference mechanism. This was addressed in ALI (Dumoulin et al., 2016)) and BiGAN (Donahue et al., 2016) frameworks, where they add an inference network to the model and the discriminator distinguishes between joints distribution of latent codes and data. Liu et al. (2017) propose adding a regularization term to ALI that minimizes conditional entropy, as a way to increases mutual information between latent codes and data and reduce non-identifiability in ALI.

Stabilizing training of GANs, which is arguably their biggest downside, has been the focus of much research recently. One of the most notable advances in this front is WGAN (Arjovsky et al., 2017), which proposes to replace the Jensen-Shannon divergence of standard GAN with Wasserstein-1 distance between data and generator distributions. The follow-up model WGAN-GP (Gulrajani et al., 2017) introduces the gradient penalty formulation, in order to bound the Lipschitz constant of the discriminator. Gradient penalties, however, proved to be useful even with standard GAN formulation (Roth et al., 2017; Fedus et al., 2018).

Chapter 7

Calibrating Energy-based Generative Adversarial Networks

7.1. Introduction

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) 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 (Radford et al., 2015; Salimans et al., 2016; Zhao et al., 2016) 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. (2014), 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 (Zhao et al., 2016) 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. (2016). Our work is inspired by the work of Kim and Bengio (2016), who propose maximizing the entropy of the generator's output.

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 an 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 the similar approach of (Kim and Bengio, 2016). Finally, for a specific instantiation of the general formulation, we investigate two approximation techniques to optimize the training objective, and verify our results empirically.

7.2. Related Work

Following a similar motivation, the field of Inverse Reinforcement Learning (IRL) (Ng and Russell, 2000) 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 (Abbeel and Ng, 2004; Ziebart et al., 2008) aims at learning a policy (analogous to the generator) using the reward signals recovered by IRL. Notably, Ho and Ermon (2016) 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 (2016) provide a duality-based interpretation of the problem, which inspires our theoretical analysis. However, as the focus of (Ho and Ermon, 2016) 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. (2016) and Kim and Bengio (2016). In the next section, We show how one can derive both of these approaches from different assumptions regarding regularization of the generative model.

7.3. Alternative Formulation of Adversarial Training

7.3.1. Background

Before presenting the proposed formulation, we first state some basic assumptions required by the analysis, and introduce notations used throughout this chapter. Following the original work on GANs (Goodfellow et al., 2014), 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 work. 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, $p_{\text{data}} \in \mathcal{P} : \mathcal{X} \to \mathbb{R}$ and $p_{\text{gen}} \in \mathcal{P} : \mathcal{X} \to \mathbb{R}$ 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.

7.3.2. 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 at 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, i.e., a discriminator which assigns the same probability/score to all inputs. In this work, we propose to tackle this single training signal constraint directly. Specifically, we describe an 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,

$$\max_{c} \min_{p_{\text{gen}} \in \mathcal{P}} \quad \mathbb{E}_{x \sim p_{\text{gen}}}\Big[c(x)\Big] - \mathbb{E}_{x \sim p_{\text{data}}}\Big[c(x)\Big] + K(p_{\text{gen}}), \tag{7.3.1}$$

where $c(x) : \mathcal{X} \to \mathbb{R}$ is the discriminator that assigns each data point an unbounded scalar cost, and $K(p_{\text{gen}}) : \mathcal{P} \to \mathbb{R}$ 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. (7.3.1), 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

$$L(p_{\text{gen}}, c) = \mathbb{E}_{x \sim p_{\text{gen}}}\Big[c(x)\Big] - \mathbb{E}_{x \sim p_{\text{data}}}\Big[c(x)\Big] + K(p_{\text{gen}}), \tag{7.3.2}$$

then $L(p_{\text{gen}}, c)$ is the Lagrange dual function of the following optimization problem

$$\begin{aligned} \min_{p_{\text{gen}} \in \mathcal{P}} \quad & K(p_{\text{gen}}) \\ \text{s.t.} \quad & p_{\text{gen}}(x) - p_{\text{data}}(x) = 0, \forall x \in \mathcal{X} \end{aligned} \tag{7.3.3}$$

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 (Ho and Ermon, 2016, 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:

$$\begin{aligned} \min_{p_{\text{gen}}} \quad & K(p_{\text{gen}}) \\ \text{s.t.} \quad & p_{\text{gen}}(x) - p_{\text{data}}(x) = 0, \forall x \\ & -p_{\text{gen}}(x) \leq 0, \forall x \\ & \sum_{x \in \mathcal{X}} p_{\text{gen}}(x) - 1 = 0. \end{aligned} \tag{7.3.4}$$

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 (7.3.3) is a convex optimization problem. Furthermore, since (i) dom_K is open, and (ii) there exists a feasible solution $p_{\text{gen}} = p_{\text{data}}$ to (7.3.4), by the refined Slater's condition (Boyd and Vandenberghe, 2004, page 226), we can further verify that strong duality holds for (7.3.4). 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 7.3.1. *By the KKT conditions of the convex problem* (7.3.4)*, 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:*

$$
\begin{aligned}
& c^*(x) = -\frac{\partial K(p_{gen})}{\partial p_{gen}(x)}\bigg|_{p_{gen}=p_{data}} - \lambda^* + \mu^*(x), \forall x \in \mathcal{X}, \\
& \textit{where} \quad \mu^*(x) = \begin{cases} 0, & p_{data}(x) > 0 \\ u_x, & p_{data}(x) = 0 \end{cases}, \\
& \qquad \lambda^* \in \mathbb{R}, \textit{ is an under-determined real number independent of } x, \\
& \qquad u_x \in \mathbb{R}_{\geq 0}, \textit{ is an under-determined non-negative real number.}
\end{aligned}
\tag{7.3.5}
$$

The detailed proof of proposition 7.3.1 is provided in Section 7.7.1. From (7.3.5), 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 (7.3.5), 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

$$c^*_{\text{ent}}(x) = -\log p_{\text{data}}(x) - 1 - \lambda^* + \mu^*(x), \forall x \in \mathcal{X}, \tag{7.3.6}$$

where $\mu^*(x)$ and λ^* have the same definitions as in (7.3.5).

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 (Ziebart et al., 2008) and the directed-generator-trained energy-based model (Kim and Bengio, 2016). 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

$$c^*_{\ell_2}(x) = -p_{\text{data}}(x) - \lambda^* + \mu^*(x), \forall x \in \mathcal{X}, \tag{7.3.7}$$

with $\mu^*(x), \lambda^*$ similarly defined as in (7.3.5).

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 (7.3.6), 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

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 p_{gen} at all. With $K(p_{\text{gen}}) = \text{const}$, we simply have

$$c^*_{\text{cst}}(x) = -\lambda^* + \mu^*(x), \forall x \in \mathcal{X}, \tag{0.8}$$

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 (7.3.1) simplifies to:

$$\max_{c} \min_{p_{\text{gen}} \in \mathcal{P}} \quad \mathbb{E}_{x \sim p_{\text{gen}}}\Big[c(x)\Big] - \mathbb{E}_{x \sim p_{\text{data}}}\Big[c(x)\Big], \tag{0.9}$$

which is very similar to the EBGAN objective (Zhao et al., 2016, equation (2) and (4)). As we show in Section 7.7.2, compared to the objective in (0.9), 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 7.7.2 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 7.7.3 for detailed analysis).

7.4. 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 do not have an analytical form for 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 (7.3.1) requires the generator distribution. Faced with this problem, we focus on the max-entropy formulation, and exploit two different approximations of the regularization term $K(p_{\text{gen}}) = -H(p_{\text{gen}})$.

7.4.1. 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:

$$-\nabla_\theta H(p_{\text{gen}}) = \mathbb{E}_{z\sim p_z}\left[\nabla_\theta \log p_{\text{gen}}(g_\theta(z))\right] = \mathbb{E}_{z\sim p_z}\left[\frac{\partial g_\theta(z)}{\partial \theta}\frac{\partial \log p_{\text{gen}}(g_\theta(z))}{\partial g_\theta(z)}\right], \tag{7.4.1}$$

where the first equality relies on the "reparametrization trick". Equation 7.4.1 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

$$\frac{\partial \log p_{\text{gen}}(x_i)}{\partial x_i} \approx \mu_i - x_i, \quad \text{where } \mu_i = \frac{1}{k}\sum_{x'\in \text{KNN}(x_i)} x' \text{ is the mean of the Gaussian} \tag{7.4.2}$$

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

$$-\nabla_\theta H(p_{\text{gen}}) \approx \alpha\frac{1}{k}\sum_{x_i = g_\theta(z_i)} \frac{\mu_i - x_i}{\|\mu_i - x_i\|_2} \tag{7.4.3}$$

where α is the hyper-parameter and μ_i is defined as in equation (7.4.2).

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.

7.4.2. 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:

$$\begin{aligned} I(p_{\text{gen}}(x); p_{\text{gen}}(z)) &= H(p_{\text{gen}}(x)) - H(p_{\text{gen}}(x \mid z)) \\ &= H(p_{\text{gen}}(z)) - H(p_{\text{gen}}(z \mid x)), \end{aligned} \tag{7.4.4}$$

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:

$$H(p_{\text{gen}}(x)) = H(p_{\text{gen}}(z)) - H(p_{\text{gen}}(z \mid x)) + H(p_{\text{gen}}(x \mid z)) \tag{7.4.5}$$

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:

$$H(p_{\text{gen}}(z \mid x)) = \mathbb{E}_{x \sim p_{\text{gen}}(x)} \left[\mathbb{E}_{z \sim p_{\text{gen}}(z|x)} \left[-\log p_{\text{gen}}(z \mid x) \right] \right] \tag{7.4.6}$$

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 $q_{\text{gen}}(z \mid x)$:

$$\begin{aligned} H(p_{\text{gen}}(z \mid x)) &= \mathbb{E}_{x \sim p_{\text{gen}}(x)} \left[\mathbb{E}_{z \sim p_{\text{gen}}(z|x)} \left[-\log q_{\text{gen}}(z \mid x) \right] - \mathrm{KL}(p_{\text{gen}}(z \mid x) \| q_{\text{gen}}(z \mid x)) \right] \\ &\leq \mathbb{E}_{x \sim p_{\text{gen}}(x)} \left[\mathbb{E}_{z \sim p_{\text{gen}}(z|x)} \left[-\log q_{\text{gen}}(z \mid x) \right] \right] \\ &= \mathcal{U}(q_{\text{gen}}). \end{aligned} \tag{7.4.7}$$

We can also rewrite the variational upper bound as:

$$\mathcal{U}(q_{\text{gen}}) = \mathbb{E}_{x,z \sim p_{\text{gen}}(x,z)} \left[-\log q_{\text{gen}}(z \mid x) \right] = \mathbb{E}_{z \sim p_{\text{gen}}(z)} \left[\mathbb{E}_{x \sim p_{\text{gen}}(x|z)} \left[-\log q_{\text{gen}}(z \mid x) \right] \right], \tag{7.4.8}$$

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 $H(p_{\text{gen}}(x))$.

7.5. 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 (Radford et al., 2015), except in Sec. 7.5.1 where we use fully-connected models. [1]

7.5.1. 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 1.

Since the data lies in 2D 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) 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 Figures 2, 3 for baseline models, and Figures 4, 5 for the proposed models. As we can see, all four models can generate perfect samples. However, both GAN and EGAN-Const lead to a degenerate discriminator's 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.

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

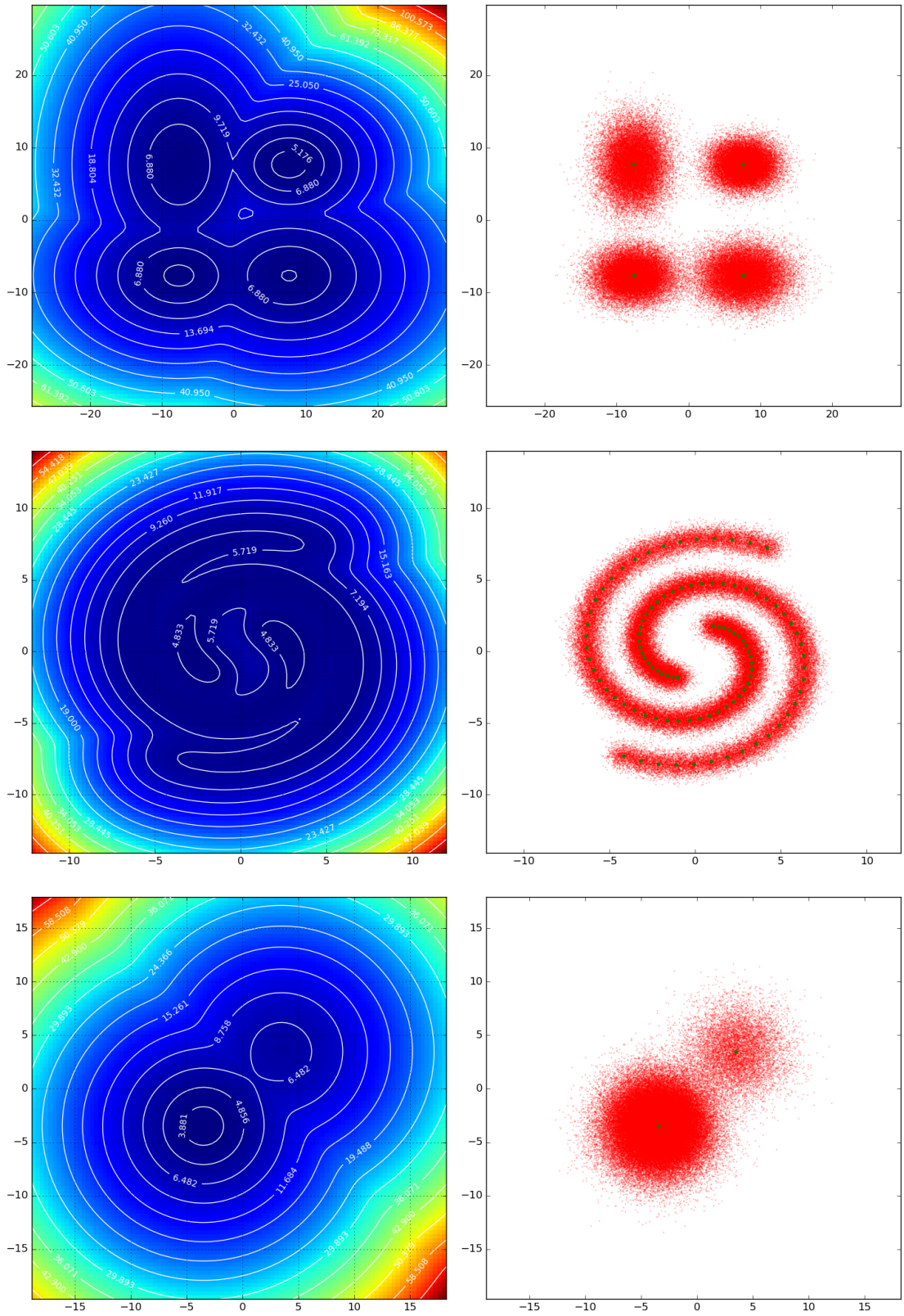


Fig. 1. True energy functions and samples from synthetic distributions. Green dots in the sample plots indicate the mean of each Gaussian component.

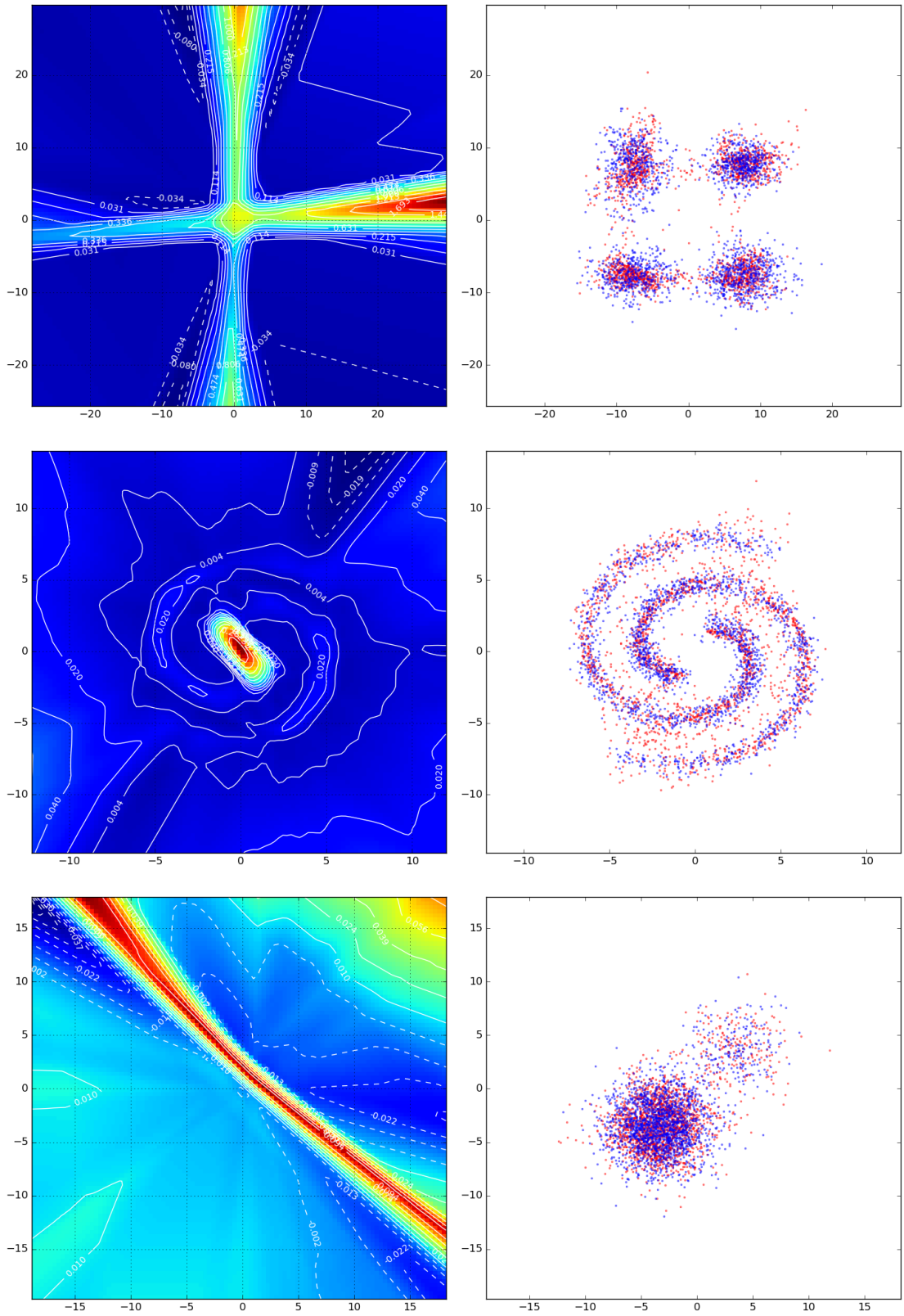


Fig. 2. Learned energies and samples from standard **GAN**, whose discriminator cannot retain density information at the optimum. In the sample plots, blue dots indicate generated samples, and red dots indicate real ones.

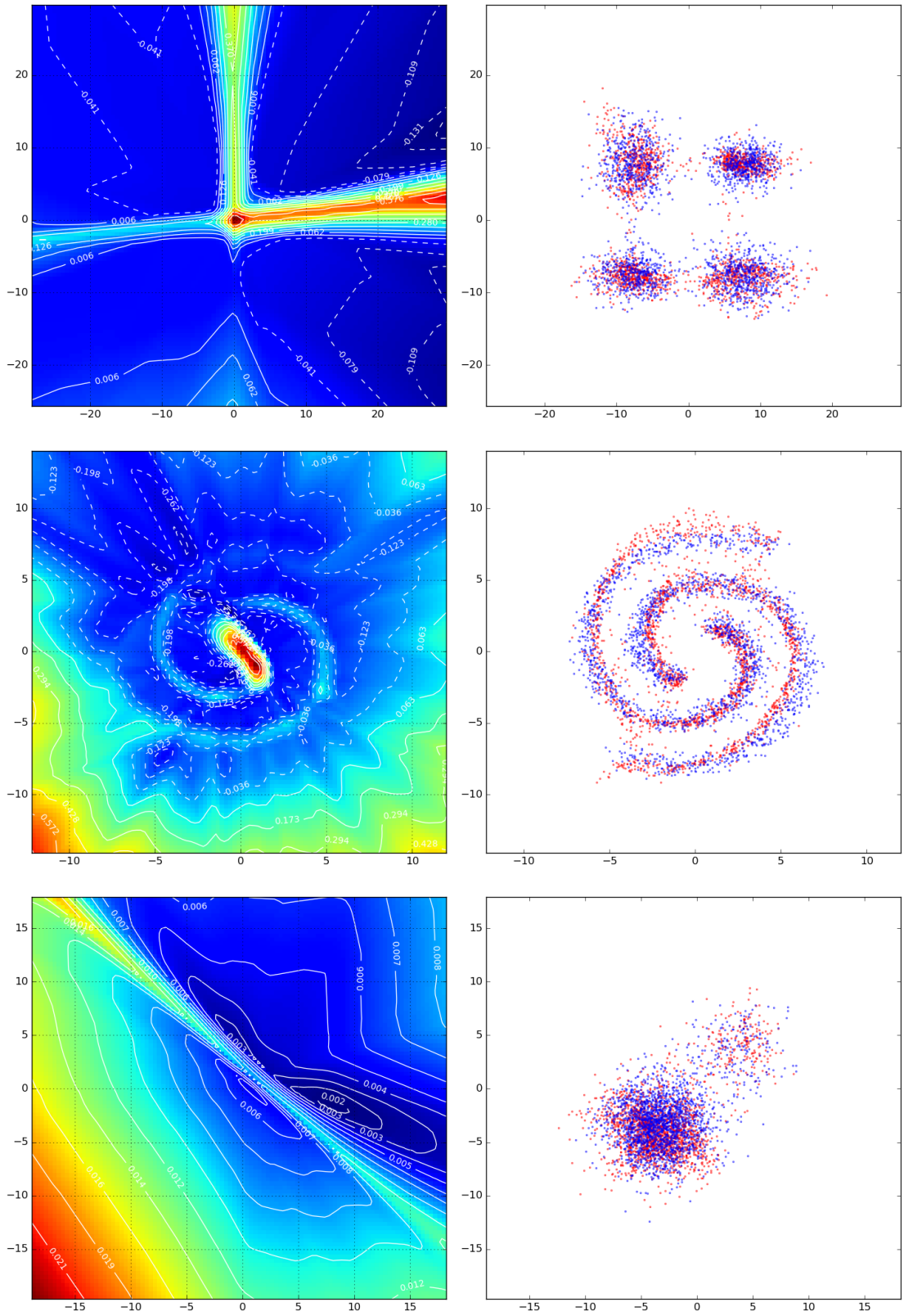


Fig. 3. Learned energies and samples from Energy GAN without regularization **(EGAN-Const)**, whose discriminator cannot retain density information at the optimum. In the sample plots, blue dots indicate generated samples, and red dots indicate real ones.

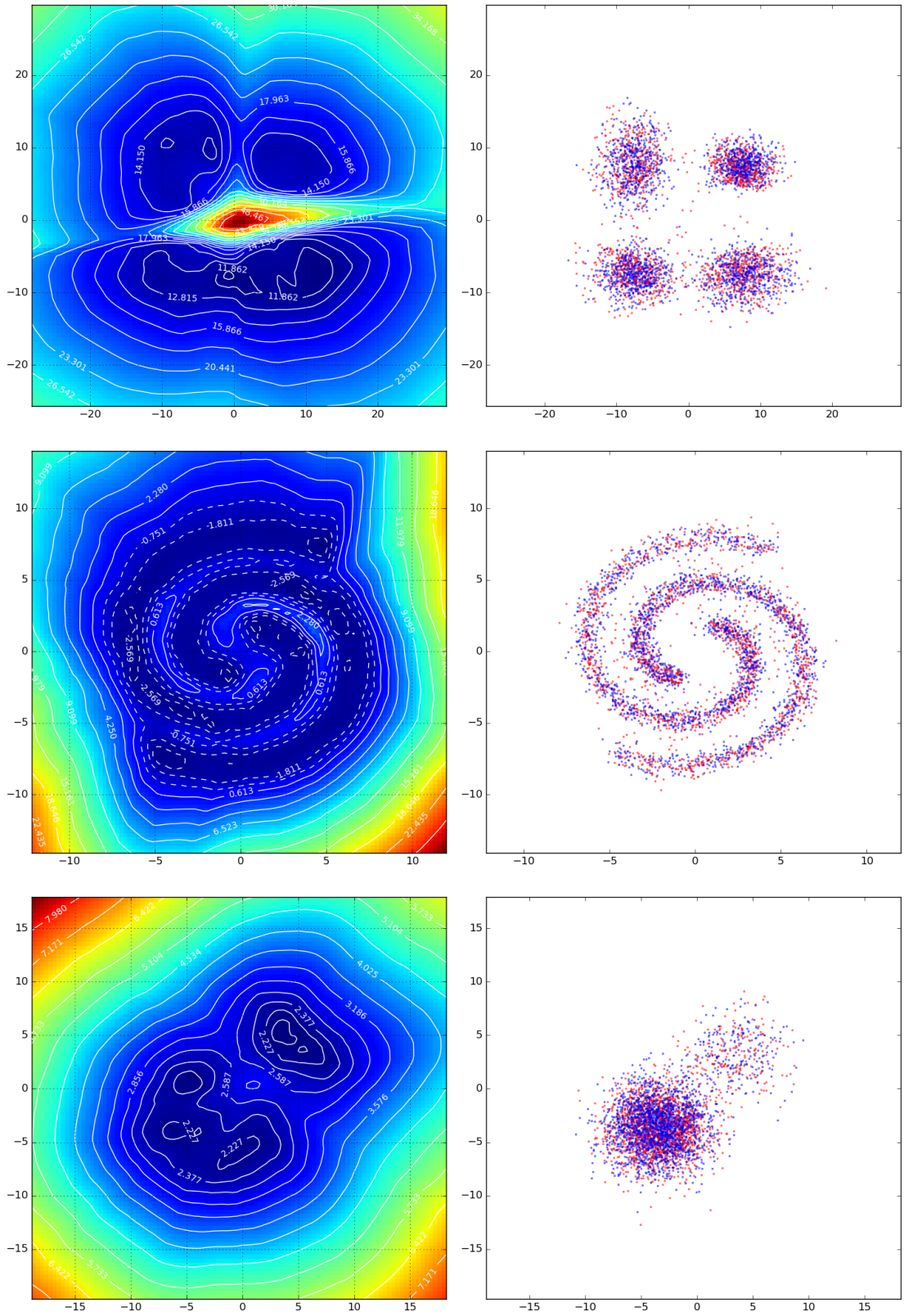


Fig. 4. Learned energies and samples from Entropy regularized Energy GAN with variational inference approximation **(EGAN-Ent-VI)**, whose discriminator can retain density information at the optimum. Blue dots are generated samples, and red dots are real ones.

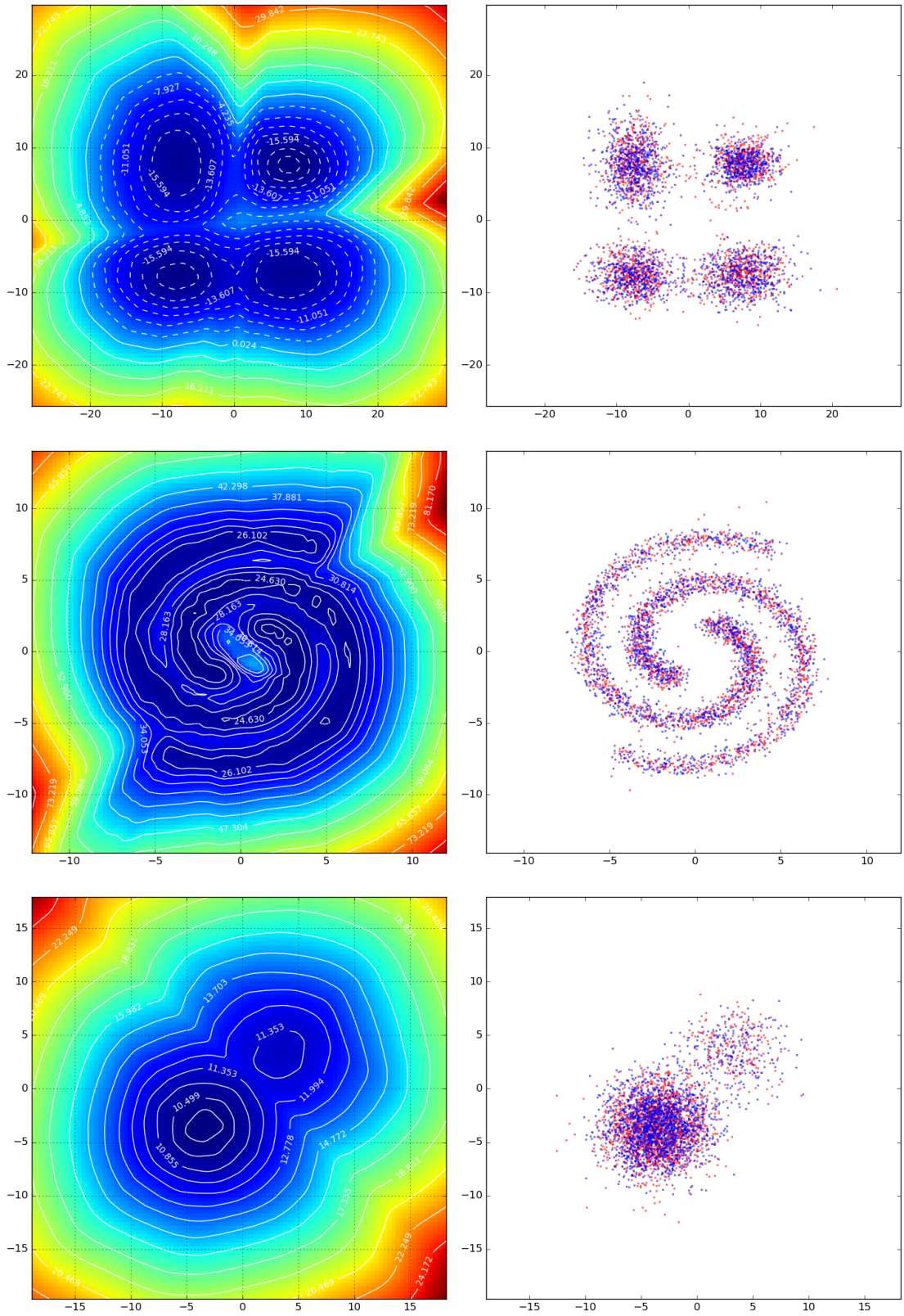


Fig. 5. Learned energies and samples from Entropy regularized Energy GAN with nearest neighbor approximation **(EGAN-Ent-NN)**, whose discriminator can retain density information at the optimum. Blue dots are generated samples, and red dots are real ones.

	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 1. Quantitative evaluation on 2D data using pairwise KL divergence between distributions. Bold face indicate the lowest divergence within group.

7.5.1.1. *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}

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 1 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 4 and 5. 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.

7.5.1.2. *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 Figures 6 and 7. As we can see in Figure 6, 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 Figure 6-(a). However, the direction of gradients in the Figure 6-(e) does not match this.
- It is inaccurate in magnitude. As we can see, the entropy approximation gradient (Figure 6-(e)) has much larger norm than the discriminator gradient (Figure 6-(d)). As a result, the total gradient (Figure 6-(f)) 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 Figure 7. As a result, it leads to more accurate energy contour, as well as faster training. Furthermore, from Figure 7-(e), we can see the entropy gradient does have the cancel-out effect on the discriminator gradient, which again matches our theory.

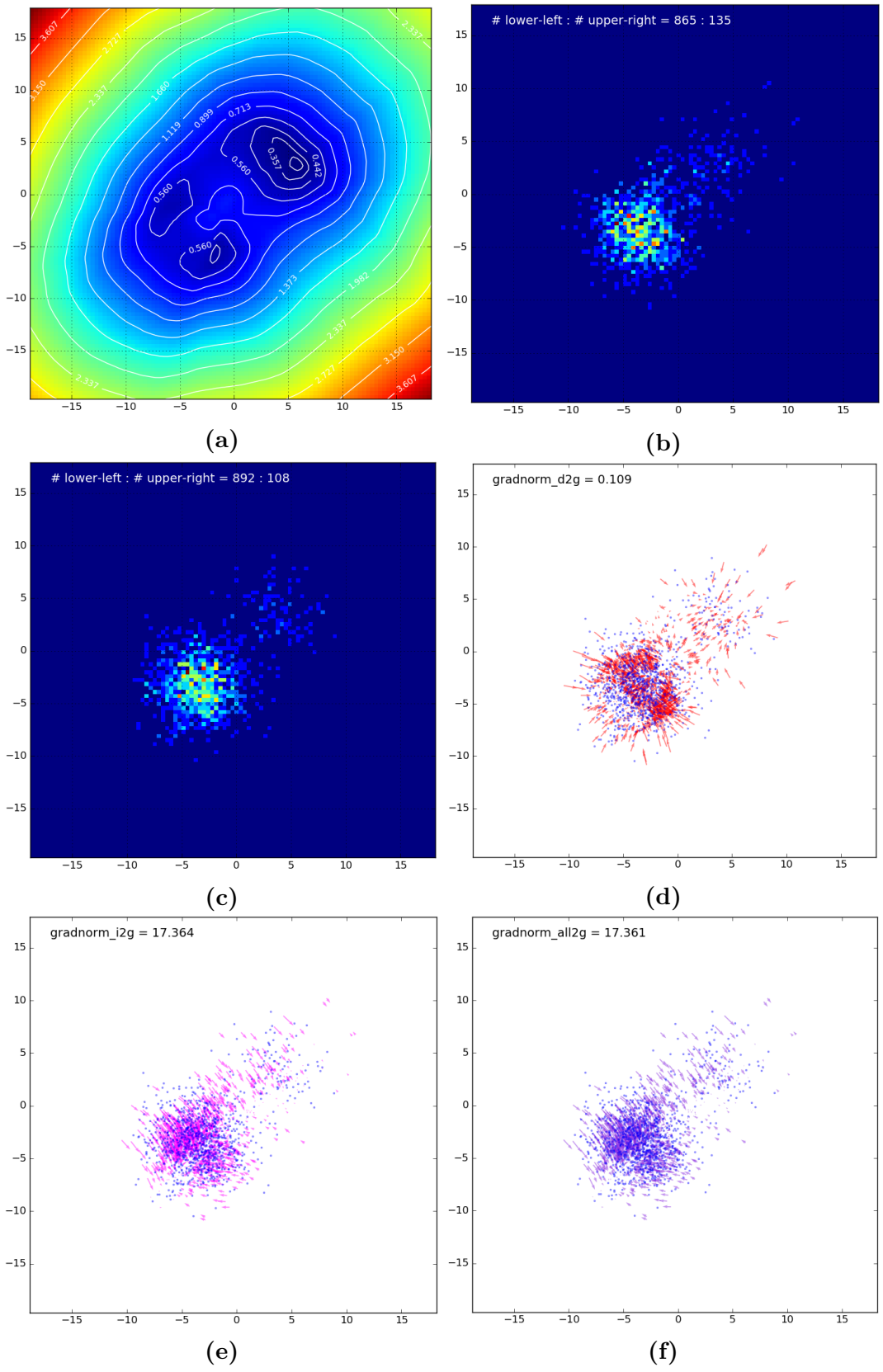


Fig. 6. Training details under variational inference entropy approximation. (a) Current energy plot. (b) Frequency map of generated samples. (c) Frequency map of real samples. (d) Discriminator's gradient w.r.t. each training sample. (e) VI Entropy gradient w.r.t. each training samples. (f) All gradient (discriminator + entropy) w.r.t. each training sample.

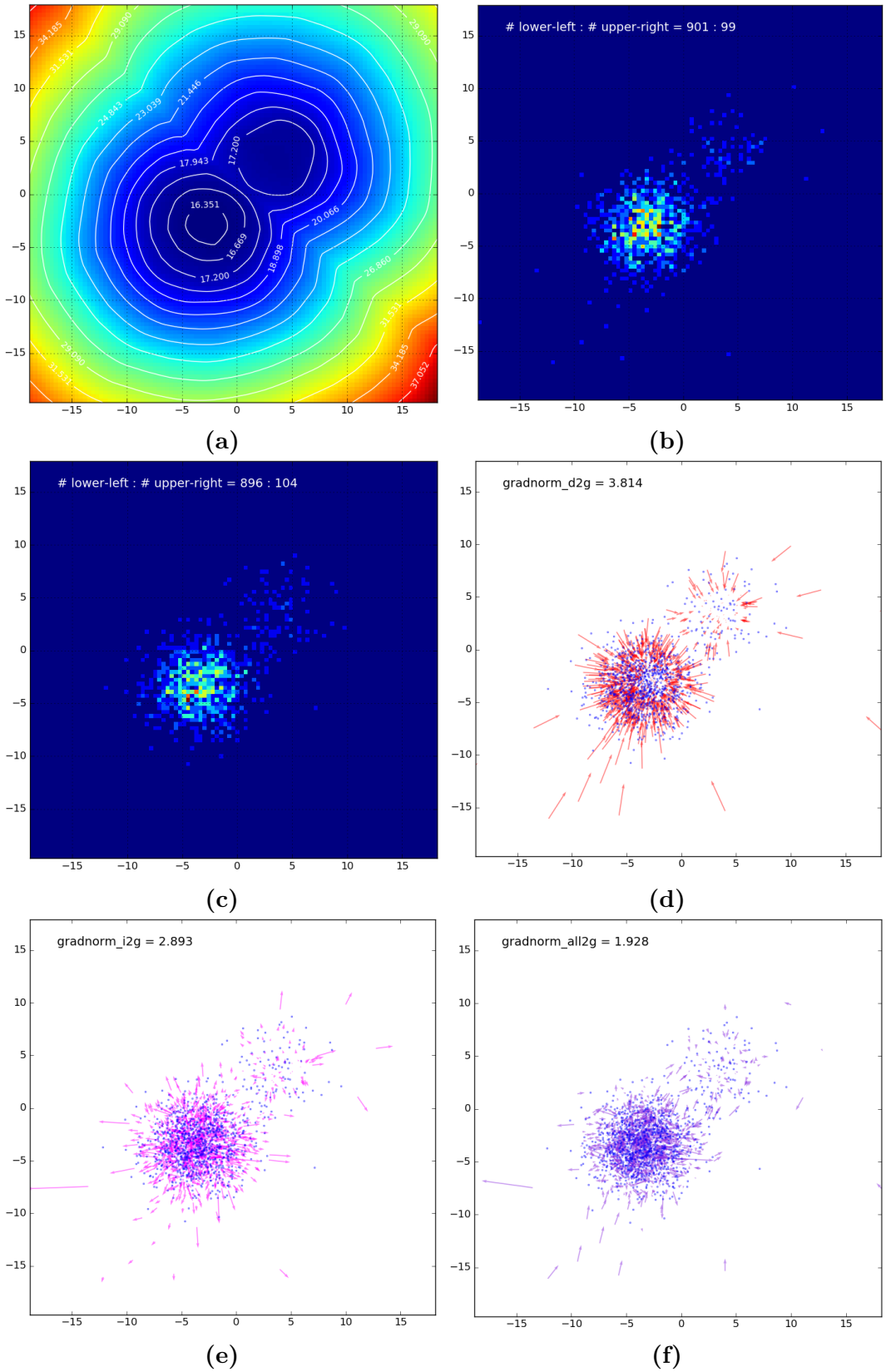


Fig. 7. Training details under nearest neighbor entropy approximation. (a) Current energy plot. (b) Frequency map of generated samples. (c) Frequency map of real samples. (d) Discriminator's gradient w.r.t. each training sample. (e) NN Entropy gradient w.r.t. each training samples. (f) All gradient (discriminator + entropy) w.r.t. each training sample.

7.5.2. 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. We show in Figure 8a the mean of all training samples, which can give a sense of a "canonical" or the most common style (i.e., highest density) of digit 1 in NIST. Figure 8 shows the ranking of all 1000 generated and real images (from the test set) for the three models. 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. We note that we observe the same trend in modeling other digits.

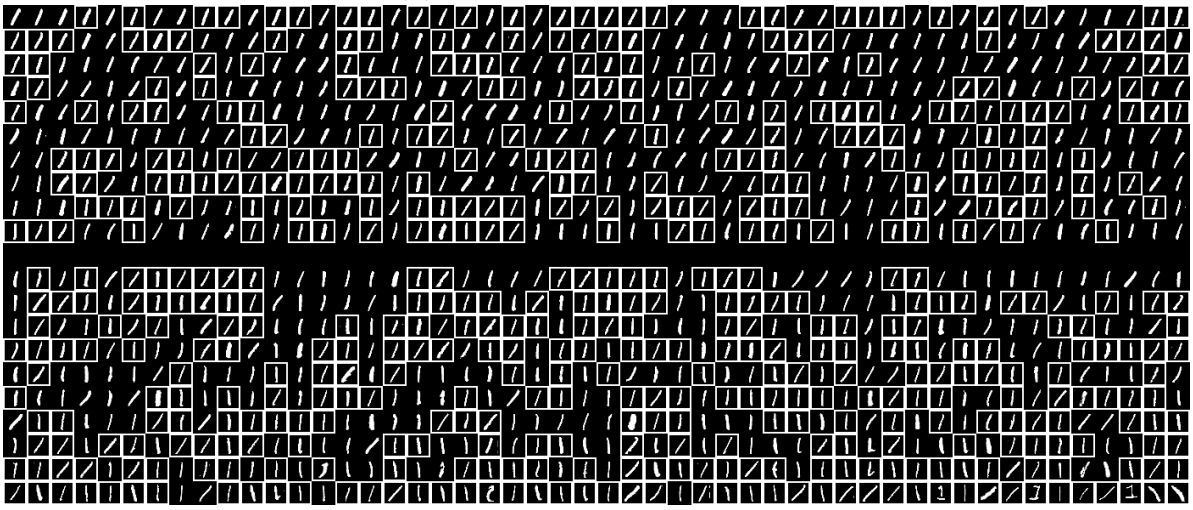
7.5.2.1. *Classifier performance using discriminator features*

In order to provide more quantitative intuitions on the learned discriminator at convergence, we adopt a proxy measure using discriminator features. Specifically, we take the last-layer activations of a converged discriminator network as **fixed** pretrained features, and build a linear classifier on top of them. Hypothetically, if the discriminator does not degenerate, the extracted last-layer features should maintain more information about each data point than 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 activations from their discriminator networks as fixed input features. Based on these fixed features, we train a randomly initialized linear classifier to on the task of classification of MNIST digits. Based on 10 runs (with different initialization) of each of the three models, the test classification performance is summarized in Table 2. For comparison purpose, we also include a baseline where the input features are extracted from a discriminator network with random weights.

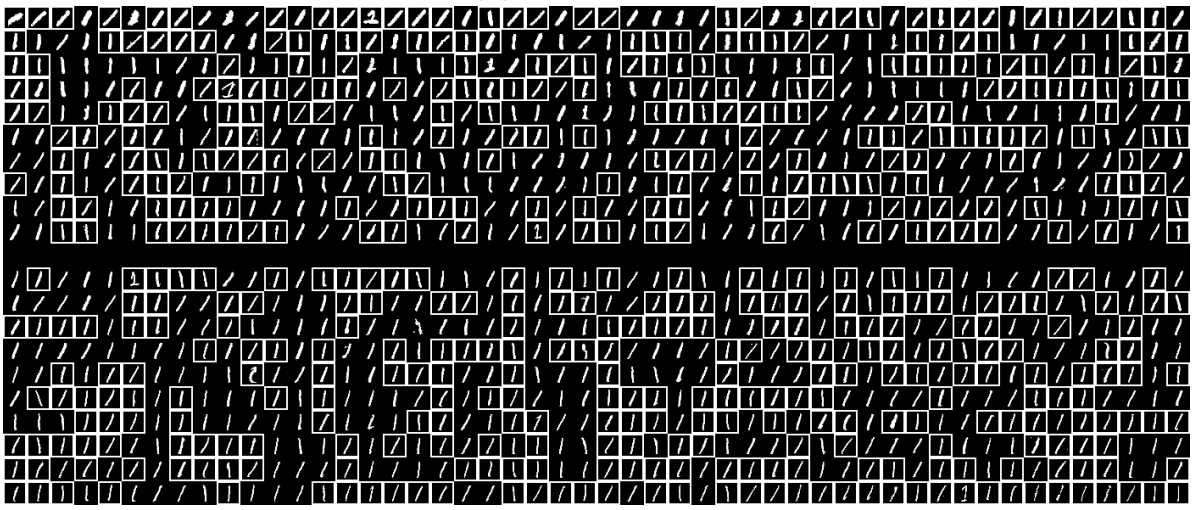
Based on the proxy measure, EGAN-Ent-NN seems to maintain more information of data, which suggests that the discriminator from our proposed formulation is more informative.

[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.

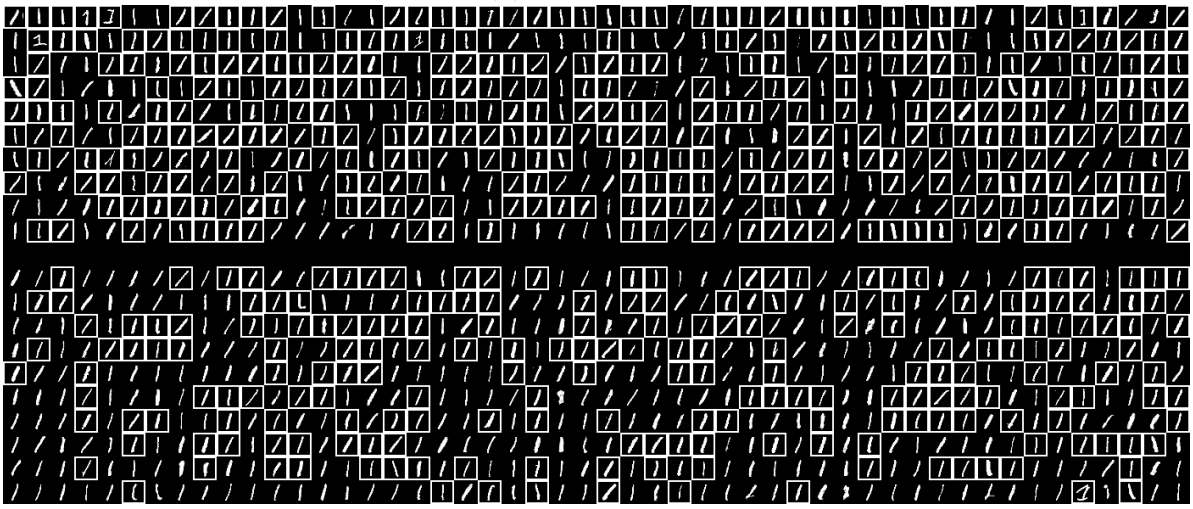
(a) Mean of all digit-1 in NIST



(b) EGAN-Ent-NN



(c) EGAN-Const



(d) GAN

Fig. 8. (a) Mean 1's in NIST. (b)-(d) 1000 generated and test images (bounding box) ranked according their assigned energies by each model.

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 2. Test performance of linear classifiers based on last-layer discriminator features.

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 measure should be understood cautiously.

7.5.3. 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 9 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 in both CIFAR-10 and CelebA datasets.

(a) CIFAR-10 **(b)** CelebA

Fig. 9. Samples generated from our model.

We futher validate the quality of our model's samples using the *Inception score* proposed by (Salimans et al., 2016) [3] on CIFAR-10. Table 3 shows the scores of our EGAN-Ent-VI, the best GAN model from (Salimans et al., 2016) which uses only unlabeled data, and an EGAN-Const model which has the same architecture as our model. We notice that even

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

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

Table 3. Inception scores on CIFAR-10. † As reported in (Salimans et al., 2016) without using labeled data.

without employing suggested techniques in (Salimans et al., 2016), 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.

7.6. Conclusion

In this chapter, we have introduced an adversarial learning approach that is able to provide sensible energy estimates for samples. Our formulation 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 show that we can also obtain samples of state-of-the-art quality.

Future research directions include developing better entropy approximations, which can scale better on harder tasks. In addition, this framework has very promising applications in sequence generation tasks (e.g., language generation), which, as of writing this thesis, remains to be one of the most challenging research problems for generative adversarial networks.

7.7. Additional Theoretical Analysis

7.7.1. Optimal discriminator form under the proposed formulation

Proof of proposition 7.3.1. 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

$$\begin{aligned} L(p_{\text{gen}}, c, \mu, \lambda) =& K(p_{\text{gen}}) + \sum_{x \in \mathcal{X}} c(x) \Big(p_{\text{gen}}(x) - p_{\text{data}}(x) \Big) \\ & - \sum_{x \in \mathcal{X}} \mu(x) p_{\text{gen}}(x) + \lambda \Big(\sum_{x \in \mathcal{X}} p_{\text{gen}}(x) - 1 \Big) \end{aligned} \tag{7.7.1}$$

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

$$\begin{aligned}
\left.\frac{\partial K(p_{\text{gen}})}{\partial p_{\text{gen}}(x)}\right|_{p_{\text{gen}}=p_{\text{data}}} + c^*(x) - \mu^*(x) + \lambda^* = 0, \quad \forall x \qquad & \text{(stationarity)} \\
\mu^*(x) p^*_{\text{gen}}(x) = 0, \quad \forall x \qquad & \text{(complement slackness)} \\
\mu^*(x) \geq 0, \quad \forall x \qquad & \text{(dual feasibility)} \\
p^*_{\text{gen}}(x) \geq 0, \quad p^*_{\text{gen}}(x) = p_{\text{data}}(x), \quad \forall x \qquad & \text{(primal feasibility)} \\
\sum_{x \in \mathcal{X}} p_{\text{gen}}(x) = 1 \qquad & \text{(primal feasibility)}
\end{aligned} \tag{7.7.2}$$

Rearranging the conditions above, we get $p^*_{\text{gen}}(x) = p_{\text{data}}(x), \forall x \in \mathcal{X}$ as well as equation (7.3.5), which concludes the proof. □

7.7.2. Optimal conditions of EBGAN

In (Zhao et al., 2016), 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 (Goodfellow et al., 2014). 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

$$\begin{aligned}
c^*(x; p_{\text{gen}}) &= \arg\max_{c \in \mathcal{C}} \quad -\mathbb{E}_{p_{\text{gen}}} \max(0, m - c(x)) - \mathbb{E}_{p_{\text{data}}} c(x) \\
&= \arg\max_{c \in \mathcal{C}} \quad \mathbb{E}_{p_{\text{gen}}} \min(0, c(x) - m) - \mathbb{E}_{p_{\text{data}}} c(x)
\end{aligned} \tag{7.7.3}$$

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 (7.7.3) is independent w.r.t. each x, the optimal solution can be easily derived as

$$c^*(x; p_{\text{gen}}) = \begin{cases} 0, & p_{\text{gen}}(x) < p_{\text{data}}(x) \\ m, & p_{\text{gen}}(x) > p_{\text{data}}(x) \\ \alpha_x, & p_{\text{gen}}(x) = p_{\text{data}}(x) > 0 \\ \beta_x, & p_{\text{gen}}(x) = p_{\text{data}}(x) = 0 \end{cases} \tag{7.7.4}$$

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},

$$\begin{aligned} p^*_{\text{gen}} &= \arg\min_{p_{\text{gen}} \in \mathcal{P}} \mathbb{E}_{x \sim p_{\text{gen}}} c^*(x; p_{\text{gen}}) - \mathbb{E}_{x \sim p_{\text{data}}} c^*(x; p_{\text{gen}}) \\ &= \arg\min_{p_{\text{gen}} \in \mathcal{P}} \sum_{x \in \mathcal{X}} \Big[p_{\text{gen}}(x) - p_{\text{data}}(x) \Big] c^*(x; p_{\text{gen}}) \end{aligned} \tag{7.7.5}$$

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

Proposition 7.7.1. *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 (7.7.4) into (7.7.5), the $L(p_{\text{gen}})^*$ can be written as

$$\begin{aligned} L(p^*_{\text{gen}}) &= \sum_{x \in \mathcal{X}_{<} \bigcup \mathcal{X}_{<} \bigcup \mathcal{X}_{=}} \Big[p^*_{\text{gen}}(x) - p_{\text{data}}(x) \Big] c^*(x; p^*_{\text{gen}}) \\ &= \sum_{x \in \mathcal{X}_{<}} \Big[p^*_{\text{gen}}(x) - p_{\text{data}}(x) \Big] c^*(x; p^*_{\text{gen}}) + \sum_{x \in \mathcal{X}_{>}} \Big[p^*_{\text{gen}}(x) - p_{\text{data}}(x) \Big] c^*(x; p^*_{\text{gen}}) \\ &= m \sum_{x \in \mathcal{X}_{>}} p^*_{\text{gen}}(x) - p_{\text{data}}(x) \\ &> 0 \end{aligned} \tag{7.7.6}$$

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

$$L(p'_{\text{gen}}) = 0 < L(p^*_{\text{gen}}) \tag{7.7.7}$$

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 (7.7.4), it directly follows that $c^*(x; p^*_{\text{gen}}) = \alpha_x$, which completes the proof. □

7.7.3. 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 (Nowozin et al., 2016, equation (6)) family, which include the original GAN formulation as a special case.

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

$$\max_{c} \min_{p_{\text{gen}} \in \mathcal{P}} \quad \mathbb{E}_{x \sim p_{\text{gen}}}\Big[f^\star(c(x))\Big] - \mathbb{E}_{x \sim p_{\text{data}}}\Big[c(x)\Big], \tag{7.7.8}$$

where the $f^\star(\cdot)$ denotes the convex conjugate (Boyd and Vandenberghe, 2004) 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

$$c^*(x) = f'(\frac{p_{\text{data}}(x)}{p_{\text{gen}}(x)}) \tag{7.7.9}$$

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 (7.7.8), since the term $K(p_{\text{gen}})$ is a constant w.r.t. the discriminator, it does not change the result given by equation (7.7.9) 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.

Chapter 8

Prologue to Fourth Article

8.1. Article Details

Augmented CycleGAN: Learning Many-to-Many Mappings from Unpaired Data. Amjad Almahairi, Sai Rajeswar, Alessandro Sordoni, Philip Bachman and Aaron Courville. *Proceedings of the 35^{rd} International Conference on Machine Learning (ICML 2018).*

Personal Contribution. I am the main contributor to this work with regards to designing and carrying out experiments, analyzing results and writing the manuscript.

8.2. Context

Obtaining large amounts of labelled data for structured prediction tasks, such as image segmentation and translation, can be very difficult and expensive. Learning inter-domain mappings from unpaired data has huge potential in improving performance in structured prediction tasks by reducing the need for paired data.

CycleGAN (Zhu et al., 2017a) was proposed for learning image-to-image translation using unpaired data, but critically assumes the underlying inter-domain mapping is approximately deterministic and one-to-one. This assumption renders the model ineffective for tasks requiring flexible, many-to-many mappings.

8.3. Contributions

We propose a new model, called Augmented CycleGAN, which learns many-to-many mappings between domains. We examine Augmented CycleGAN qualitatively and quantitatively on several image datasets. In addition, we show that Augmented CycleGAN can be effectively applied in a semi-supervised learning settings.

8.4. Recent Developments

Two concurrent works, appeared on arXiv only few weeks after our work, and proposed methods similar to Augmented CycleGAN for learning many-to-many mappings between image domains (Huang et al., 2018; Lee et al., 2018). Both approaches explicitly encode an image in two latent spaces: content and style, where content space is shared between domains, and style is specific to each domain. A minor technical difference in Huang et al. (2018) is the addition of a reconstruction cost for content codes, while in Lee et al. (2018) a KL-divergence term is used to regularize style codes. In addition, Gonzalez-Garcia et al. (2018) follow a similar approach but propose a cross-domain autoencoder to achieve cross-domain disentanglement.

The potential of learning flexible inter-domain mappings from unpaired data extends beyond image domains. There has been a lot of progress recently in the area of unsupervised machine translation (Lample et al., 2018a; Artetxe et al., 2018; Yang et al., 2018; Lample et al., 2018b). These approaches share some similarities to image-to-image translation models with regards to mapping both languages to codes in a shared space using shared encoders, and employing adversarial methods to match distributions of codes, and using reconstruction losses. However, they critically rely on powerful autoregressive RNN decoders that can capture diversity in language reasonably well. Another promising research direction is domain transfer of audio signal. Recently, Mor et al. (2018) propose a method for translating music across different musical features, such as instruments and genres. Their method relies on a shared WaveNet (Oord et al., 2016) encoder and multiple WaveNet decoders.

Chapter 9

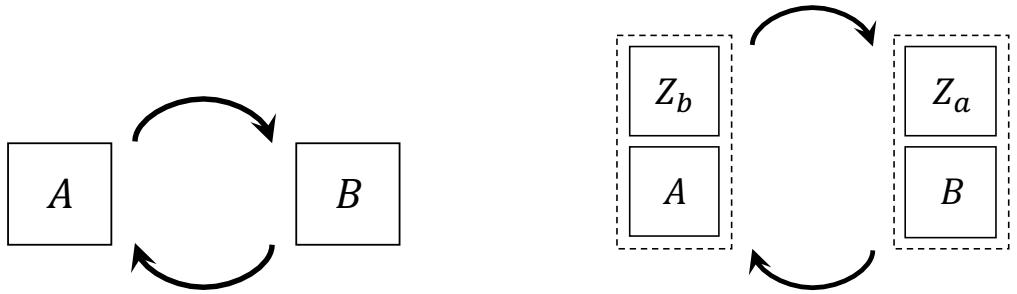
Augmented CycleGAN: Learning Many-to-Many Mappings from Unpaired Data

9.1. Introduction

The problem of learning mappings between domains from unpaired data has recently received increasing attention, especially in the context of image-to-image translation (Zhu et al., 2017a; Kim et al., 2017; Liu et al., 2017). This problem is important because, in some cases, paired information may be scarce or otherwise difficult to obtain. For example, consider tasks like face transfiguration (male to female), where obtaining explicit pairs would be difficult as it would require artistic authoring. An effective unsupervised model may help when learning from relatively few paired examples, as compared to training strictly from the paired examples. Intuitively, forcing inter-domain mappings to be (approximately) invertible by a model of limited capacity acts as a strong regularizer.

Motivated by the success of Generative Adversarial Networks (GANs) in image generation (Goodfellow et al., 2014; Radford et al., 2015), existing unsupervised mapping methods such as CycleGAN (Zhu et al., 2017a) learn a generator which produces images in one domain given images from the other. Without the use of pairing information, there are many possible mappings that could be inferred. To reduce the space of the possible mappings, these models are typically trained with a *cycle-consistency* constraint which enforces a strong connection across domains, by requiring that mapping an image from the source domain to the target domain and then back to source will result in the same starting image. This framework has been shown to learn convincing mappings across image domains and proved successful in a variety of related applications (Tung et al., 2017; Wolf et al., 2017; Hoffman et al., 2017).

One major limitation of CycleGAN is that it only learns one-to-one mappings, i.e., the model associates each input image with a single output image. We believe that most relationships across domains are more complex, and better characterized as *many-to-many*. For example, consider mapping silhouettes of shoes to images of shoes. While the mapping that CycleGAN learns can be superficially convincing (e.g. it produces a single reasonable



(a) CycleGAN **(b)** Augmented CycleGAN

Fig. 1. (a) Original CycleGAN model. (b) We propose to learn many-to-many mappings by cycling over the original domains augmented with auxiliary latent spaces. By marginalizing out auxiliary variables, we can model many-to-many mappings in between the domains.

shoe with a particular style), we would like to learn a mapping that can capture diversity of the output (e.g. produces multiple shoes with different styles). The limits of one-to-one mappings are more dramatic when the source domain and target domain substantially differ. For instance, it would be difficult to learn a CycleGAN model when the two domains are descriptive facial attributes and images of faces.

We propose a model for learning many-to-many mappings between domains from unpaired data. Specifically, we "augment" each domain with auxiliary latent variables and extend CycleGAN's training procedure to the augmented spaces. The mappings in our model take as input a sample from the source domain and a latent variable, and output both a sample in the target domain and a latent variable (Fig. 1b). The learned mappings are one-to-one in the augmented space, but many-to-many in the original domains after marginalizing over the latent variables.

Our contributions are as follows. (i) We introduce the Augmented CycleGAN model for learning many-to-many mappings across domains in an unsupervised way. (ii) We show that our model can learn mappings which produce a diverse set of outputs for each input. (iii) We show that our model can learn mappings across substantially different domains, and we apply it in a semi-supervised setting for mapping between faces and attributes with competitive results.

9.2. Unsupervised Learning of Mappings Between Domains

9.2.1. Problem Setting

Given two domains A and B, we assume there exists a mapping, potentially many-to-many, between their elements. The objective is to recover this mapping using *unpaired* samples from distributions $p_d(a)$ and $p_d(b)$ in each domain. This can be formulated as a conditional generative modeling task where we try to estimate the *true conditionals* $p(a|b)$ and $p(b|a)$ using samples from the true marginals. An important assumption here is that

elements in domains A and B are highly dependent; otherwise, it is unlikely that the model would uncover a meaningful relationship without any pairing information.

9.2.2. CycleGAN Model

The CycleGAN model (Zhu et al., 2017a) estimates these conditionals using two mappings $G_{AB} : A \to B$ and $G_{BA} : B \to A$, parameterized by neural networks, which satisfy the following constraints:

(1) **Marginal matching**: The output of each mapping should match the empirical distribution of the target domain, when marginalized over the source domain.

(2) **Cycle-consistency**: Mapping an element from one domain to the other, and then back, should produce a sample close to the original element.

Marginal matching in CycleGAN is achieved using the generative adversarial networks framework (GAN) (Goodfellow et al., 2014). Mappings G_{AB} and G_{BA} are given by neural networks trained to fool adversarial discriminators D_B and D_A, respectively. Enforcing marginal matching on target domain B, marginalized over source domain A, involves minimizing an adversarial objective with respect to G_{AB}:

$$\mathcal{L}^B_{\text{GAN}}(G_{AB}, D_B) = \mathbb{E}_{b\sim p_d(b)}\Big[\log D_B(b)\Big] + \mathbb{E}_{a\sim p_d(a)}\Big[\log(1 - D_B(G_{AB}(a)))\Big], \tag{9.2.1}$$

while the discriminator D_B is trained to maximize it. A similar adversarial loss $\mathcal{L}^A_{\text{GAN}}(G_{BA}, D_A)$ is defined for marginal matching in the reverse direction.

Cycle-consistency enforces that, when starting from a sample a from A, the reconstruction $a' = G_{BA}(G_{AB}(a))$ remains close to the original a. For image domains, closeness between a and a' is typically measured with L_1 or L_2 norms. When using the L_1 norm, cycle-consistency starting from A can be formulated as:

$$\mathcal{L}^A_{\text{CYC}}(G_{AB}, G_{BA}) = \mathbb{E}_{a\sim p_d(a)}\Big\|G_{BA}(G_{AB}(a)) - a\Big\|_1. \tag{9.2.2}$$

And similarly for cycle-consistency starting from B. The full CycleGAN objective is given by:

$$\mathcal{L}^A_{\text{GAN}}(G_{BA}, D_A) + \mathcal{L}^B_{\text{GAN}}(G_{AB}, D_B) + \gamma\mathcal{L}^A_{\text{CYC}}(G_{AB}, G_{BA}) + \gamma\mathcal{L}^B_{\text{CYC}}(G_{AB}, G_{BA}), \tag{9.2.3}$$

where γ is a hyper-parameter that balances between marginal matching and cycle-consistency.

The success of CycleGAN can be attributed to the complementary roles of marginal matching and cycle-consistency in its objective. Marginal matching encourages generating realistic samples in each domain. Cycle-consistency encourages a tight relationship between domains. It may also help prevent multiple items from one domain mapping to a single item from the other, analogous to the troublesome mode collapse in adversarial generators (Li et al., 2017).

9.2.3. Limitations of CycleGAN

A fundamental weakness of the CycleGAN model is that it learns deterministic mappings. In CycleGAN, and in other similar models (Kim et al., 2017; Yi et al., 2017), the conditionals between domains correspond to delta functions: $\hat{p}(a|b) = \delta(G_{BA}(b))$ and $\hat{p}(b|a) = \delta(G_{AB}(a))$, and cycle-consistency forces the learned mappings to be inverses of each other. When faced with complex cross-domain relationships, this results in CycleGAN learning an arbitrary one-to-one mapping instead of capturing the true, structured conditional distribution more faithfully. Deterministic mappings are also an obstacle to optimizing cycle-consistency when the domains differ substantially in complexity, in which case mapping from one domain (e.g., class labels) to the other (e.g., real images) is generally one-to-many. Next, we discuss how to extend CycleGAN to capture more expressive relationships across domains.

9.2.4. CycleGAN with Stochastic Mappings

A straightforward approach for extending CycleGAN to model many-to-many relationships is to equip it with stochastic mappings between A and B. Let Z be a latent space with a standard Gaussian prior $p(z)$ over its elements. We define mappings $G_{AB} : A \times Z \to B$ and $G_{BA} : B \times Z \to A$[1]. Each mapping takes as input a vector of auxiliary noise and a sample from the source domain, and generates a sample in the target domain. Therefore, by sampling different $z \sim p(z)$, we could in principle generate multiple b's conditioned on the same a and vice-versa. We can write the marginal matching loss on domain B as:

$$\mathcal{L}^B_{\text{GAN}}(G_{AB}, D_B) = \mathbb{E}_{b\sim p_d(b)}\Big[\log D_B(b)\Big] + \mathbb{E}_{\substack{a\sim p_d(a)\\ z\sim p(z)}}\Big[\log(1 - D_B(G_{AB}(a,z)))\Big]. \tag{9.2.4}$$

Cycle-consistency starting from A is now given by:

$$\mathcal{L}^A_{\text{CYC}}(G_{AB}, G_{BA}) = \mathbb{E}_{\substack{a\sim p_d(a)\\ z_1,z_2\sim p(z)}} \big\|G_{BA}(G_{AB}(a, z_1), z_2) - a\big\|_1 \tag{9.2.5}$$

The full training loss is similar to the objective in Eqn. 9.2.3. We refer to this model as *Stochastic CycleGAN*.

In principle, stochastic mappings can model multi-modal conditionals, and hence generate a richer set of outputs than deterministic mappings. However, Stochastic CycleGAN suffers from a fundamental flaw: the cycle-consistency in Eq. 9.2.5 encourages the mappings to ignore the latent z. Specifically, the unimodality assumption implicit in the reconstruction error from Eq. 9.2.5 forces the mapping G_{BA} to be many-to-one when cycling $A \to B \to A'$, since any b generated for a given a must map to $a' = G_{BA}(b, z) \approx a$, for all z. For the cycle $B \to A \to B'$, G_{AB} is similarly forced to be many-to-one. The only way for to G_{BA} and G_{AB} to be both many-to-one and mutual inverses is if they collapse to being (roughly) one-to-one. We could possibly mitigate this degeneracy by introducing a VAE-like encoder

[1]To avoid clutter in notation, we reuse the same symbols of deterministic mappings.

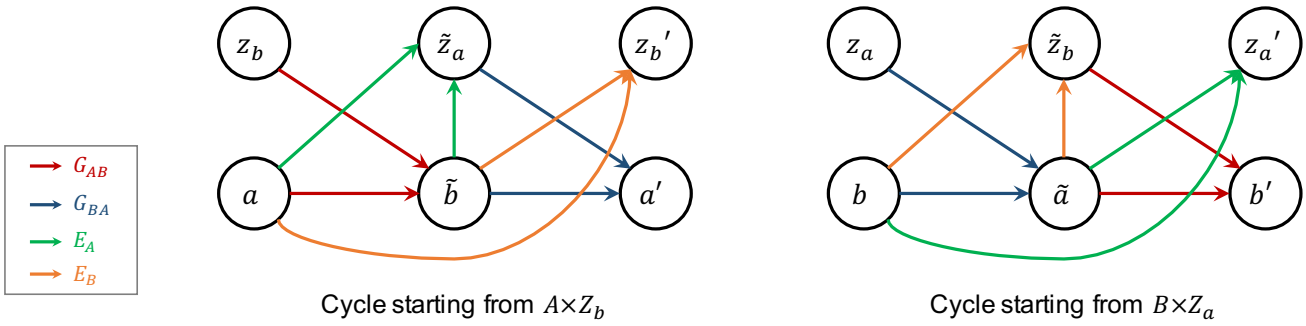


Fig. 2. Cycles starting from augmented spaces in Augmented CycleGAN. Model components identified with color coding.

and exchanging the L_1 error in Eq. 9.2.5 for a more complex variational bound on conditional log-likelihood. In the next section, we discuss an alternative approach to learning complex, stochastic mappings between domains.

9.3. Approach

In order to learn many-to-many mappings across domains, we propose to learn to map between pairs of items $(a, z_b) \in A \times Z_b$ and $(b, z_a) \in B \times Z_a$, where Z_a and Z_b are latent spaces that capture any missing information when transforming an element from A to B, and vice-versa. For example, when generating a female face ($b \in B$) which resembles a male face ($a \in A$), the latent code $z_b \in Z_b$ can capture female face variations (e.g., hair length or style) independent from a. Similarly, $z_a \in Z_a$ captures variations in a generated male face independent from the given female face. This approach can be described as learning mappings between *augmented spaces* $A \times Z_b$ and $B \times Z_a$ (Figure 1b); hence, we call it *Augmented CycleGAN*. By learning to map a pair $(a, z_b) \in A \times Z_b$ to $(b, z_a) \in B \times Z_a$, we can (i) learn a stochastic mapping from a to multiple items in B by sampling different $z_b \in Z_b$, and (ii) infer latent codes z_a containing information about a not captured in the generated b, which allows for doing proper reconstruction of a. As a result, we are able to optimize both marginal matching and cycle consistency while using stochastic mappings. We present details of our approach in the next sections. [2]

9.3.1. Augmented CycleGAN

Our proposed model has four components. First, the two mappings $G_{AB} : A \times Z_b \to B$ and $G_{BA} : B \times Z_a \to A$, which are the conditional generators of items in each domain. These models are similar to those used in Stochastic CycleGAN. We also have two encoders $E_A : A \times B \to Z_a$ and $E_B : A \times B \to Z_b$, which enable optimization of cycle-consistency with

[2]Our model captures many-to-many relationships because it captures both one-to-many and many-to-one: one item in A maps to many items in B, and many items in B map to one item in A (cycle). The same is true in the other direction.

stochastic, structured mappings. All components are parameterized with neural networks – see Fig. 2. We define mappings over augmented spaces in our model as follows. Let $p_z(z_a)$ and $p_z(z_b)$ be standard Gaussian priors over Z_a and Z_b, which are independent from $p_d(b)$ and $p_d(a)$. Given a pair $(a, z_b) \sim p_d(a)p_z(z_b)$, we generate a pair $(\tilde{b}, \tilde{z}_a)$ as follows:

$$\tilde{b} = G_{AB}(a, z_b),\ \tilde{z}_a = E_A(a, \tilde{b}). \tag{9.3.1}$$

That is, we first generate a sample in domain B, then we use it along with a to generate latent code $\tilde{z}_a$. Note here that by sampling different $z_b \sim p_z(z_b)$, we can generate multiple $\tilde{b}$'s conditioned on the same a. In addition, given the pair $(a, \tilde{b})$, we can recover information about a which is not captured in $\tilde{b}$, via $\tilde{z}_a$. Similarly, given a pair $(b, z_a) \sim p_d(b)p_z(z_a)$, we generate a pair $(\tilde{a}, \tilde{z}_b)$ as follows:

$$\tilde{a} = G_{BA}(b, z_a),\ \tilde{z}_b = E_B(b, \tilde{a}). \tag{9.3.2}$$

Learning in Augmented CycleGAN follows a similar approach to CycleGAN – optimizing both marginal matching and cycle-consistency losses, albeit over augmented spaces.

Marginal Matching Loss. We adopt an adversarial approach for marginal matching over $B \times Z_a$ where we use two independent discriminators D_B and D_{Z_a} to match generated pairs to real samples from the independent priors $p_d(b)$ and $p_z(z_a)$, respectively. Marginal matching loss over B is defined as in Eqn 9.2.4. Marginal matching over Z_a is given by:

$$\mathcal{L}^{Z_a}_{\text{GAN}}(E_A, G_{AB}, D_{Z_a}) = \mathbb{E}_{z_a \sim p_z(z_a)}\Big[\log D_{Z_a}(z_a)\Big] + \mathbb{E}_{\substack{a \sim p_d(a) \\ z_b \sim p_z(z_b)}}\Big[\log(1 - D_{Z_a}(\tilde{z}_a))\Big], \tag{9.3.3}$$

where $\tilde{z}_a$ is defined by Eqn 9.3.1. As in CycleGAN, the goal of marginal matching over B is to insure that generated samples $\tilde{b}$ are realistic. For latent codes $\tilde{z}_a$, marginal matching acts as a regularizer for the encoder, encouraging the marginalized encoding distribution to match a simple prior $p_z(z_a)$. This is similar to adversarial regularization of latent codes in adversarial autoencoders (Makhzani et al., 2016). We define similar losses $\mathcal{L}^{A}_{\text{GAN}}(G_{BA}, D_A)$ and $\mathcal{L}^{Z_b}_{\text{GAN}}(E_B, G_{BA}, D_{Z_b})$ for marginal matching over $A \times Z_b$.

Cycle Consistency Loss. We define two cycle-consistency constraints in Augmented CycleGAN starting from each of the two augmented spaces, as shown in Fig. 2. In cycle-consistency starting from $A \times Z_b$, we ensure that given a pair $(a,z_b) \sim p_d(a)p_z(z_b)$, the model is able to produce a faithful reconstruction of it after being mapped to $(\tilde{b}, \tilde{z_a})$. This is achieved with two losses; first for reconstructing $a \sim p_d(a)$:

$$\begin{aligned} \mathcal{L}^{A}_{\text{CYC}}(G_{AB}, G_{BA}, E_A) = &\ \mathbb{E}_{\substack{a \sim p_d(a) \\ z_b \sim p_z(z_b)}} \Big\| a' - a \Big\|_1, \\ \tilde{b} = G_{AB}(a, z_b),\ &\tilde{z}_a = E_A(a, \tilde{b}),\ a' = G_{BA}(\tilde{b}, \tilde{z}_a). \end{aligned} \tag{9.3.4}$$

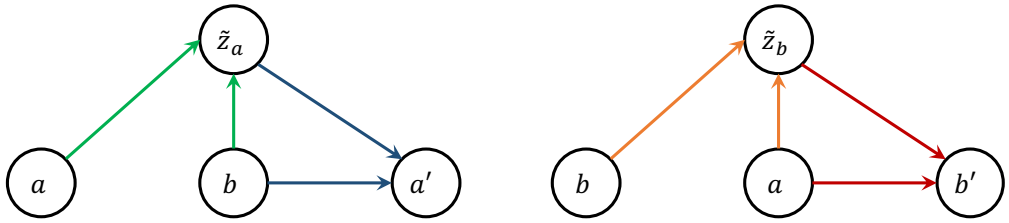


Fig. 3. Augmented CycleGAN when pairs $(a, b) \sim p_d(a, b)$ from the true joint distribution are observed. Instead of producing $\tilde{b}$ and $\tilde{a}$, the model uses samples from the joint distribution.

The second is for reconstructing $z_b \sim p_z(z_b)$:

$$\mathcal{L}^{Z_b}_{\text{CYC}}(G_{AB}, E_B) = \mathbb{E}_{\substack{a\sim p_d(a) \\ z_b\sim p_z(z_b)}} \left\| z'_b - z_b \right\|_1,$$
$$z'_b = E_B(a, \tilde{b}), \quad \tilde{b} = G_{AB}(a, z_b). \tag{9.3.5}$$

These reconstruction costs represent an autoregressive decomposition of the basic CycleGAN cycle-consistency cost from Eq. 9.2.2, after extending it to the augmented domains. Specifically, we decompose the required reconstruction distribution $p(b, z_a|a, z_b)$ into the conditionals $p(b|a, z_b)$ and $p(z_a|a, z_b, b)$.

Just like in CycleGAN, the cycle loss in Eqn. 9.3.4 enforces the dependency of generated samples in B on samples of A. Thanks to the encoder E_A, the model is able to reconstruct a because it can recover information loss in generated $\tilde{b}$ through $\tilde{z}_a$. On the other hand, the cycle loss in Eqn. 9.3.5 enforces the dependency of a generated sample $\tilde{b}$ on the given latent code z_b. In effect, it increases the mutual information between z_b and b conditioned on a, i.e., $I(b,z_b|a)$ (Chen et al., 2016b; Li et al., 2017).

Training Augmented CycleGAN in the direction $A \times Z_b$ to $B \times Z_a$ is done by optimizing:

$$\mathcal{L}^{B}_{\text{GAN}}(D_B, G_{AB}) + \mathcal{L}^{z_a}_{\text{GAN}}(D_{Z_a}, E_A, G_{AB}) + \gamma_1 \mathcal{L}^{A}_{\text{CYC}}(G_{AB}, G_{BA}, E_A) + \gamma_2 \mathcal{L}^{z_b}_{\text{CYC}}(G_{AB}, E_B), \tag{9.3.6}$$

where γ_1 and γ_2 are a hyper-parameters used to balance objectives. We define a similar objective for the direction going from $B \times Z_a$ to $A \times Z_b$, and train the model on both objectives simultaneously.

9.3.2. Semi-supervised Learning with Augmented CycleGAN

In cases where we have access to paired data, we can leverage it to train our model in a semi-supervised setting (Fig. 3). Given pairs sampled from the true joint, i.e., $(a, b) \sim p_d(a,b)$, we can define a supervision cost for the mapping G_{AB} as follows:

$$\mathcal{L}^{A}_{\text{SUP}}(G_{BA}, E_A) = \mathbb{E}_{(a,b)\sim p_d(a,b)} \left\| G_{BA}(b, \tilde{z}_a) - a \right\|_1, \tag{9.3.7}$$

where $\tilde{z}_a = E_A(a, b)$ infers a latent code which can produce a given b via $G_{BA}(b, \tilde{z}_a)$. We also apply an adversarial regularization cost on the encoder, in the form of Eqn. 9.3.3. Similar supervision and regularization costs can be defined for G_{BA} and E_B, respectively.

9.3.3. Modeling Stochastic Mappings

We note here some design choices that we found important for training our stochastic mappings. We discuss architectural and training details further in Sec. 9.5. In order to allow the latent codes to capture diversity in generated samples, we found it crucial to condition multiple layers of the network with latent codes (i.e., using latent codes as additional input to the layers), especially lower-dimensional ones that capture high-level information. This allows the latent codes to capture high-level variations of the output. On the contrary, when we condition lower-level layers (high-dimensional layers, close to the output) we find that latent codes can capture only low-level, pixel-wise variations. We also found that Conditional Normalization (CN) (Dumoulin et al., 2017; Perez et al., 2017) for conditioning layers can be more effective than concatenation, which is more commonly used (Radford et al., 2015; Zhu et al., 2017b). The basic idea of CN is to replace parameters of affine transformations in normalization layers (Ioffe and Szegedy, 2015) of a neural network with a learned function of the conditioning information. We apply CN by learning two linear functions f and g which take a latent code z as input and output scale and shift parameters of normalization layers in intermediate layers, i.e., $\gamma = f(z)$ and $\beta = g(z)$. When activations are normalized over spatial dimensions only, we get Conditional Instance Normalization (CIN), and when they are also normalized over batch dimension, we get Conditional Batch Normalization (CBN).

9.4. Related Work

There has been a surge of interest recently in unsupervised learning of cross-domain mappings, especially for image translation tasks. Previous attempts for image-to-image translation have unanimously relied on GANs to learn mappings that produce compelling images. In order to constrain learned mappings, some methods have relied on cycle-consistency based constraints similar to CycleGAN (Kim et al., 2017; Yi et al., 2017; Royer et al., 2017), while others relied on weight sharing constraints (Liu and Tuzel, 2016; Liu et al., 2017). However, the focus in all of these methods was on learning conditional image generators that produce single output images given the input image. Notably, Liu et al. (2015) propose to map inputs from both domains into a shared latent space. This approach may constrain too much the space of learnable mappings, for example in cases where the domains differ substantially (class labels and images).

Unsupervised learning of mappings have also been addressed recently in language translation, especially for machine translation (Lample et al., 2018a) and text style transfer (Shen

Model (Paired %)	Avg. L_1
CycleGAN (0%)	0.1837
StochCGAN (0%)	0.0794
Δ-GAN† (10%)	0.0748
AugCGAN (0%)	**0.0698**
AugCGAN (10%)	**0.0562**

Table 1. Reconstruction error for shoes given edges in the test set. †Same architecture as our model.

Model (Paired %)	MSE
Δ-GAN* (10%)	0.0102
Δ-GAN† (10%)	0.0096
Δ-GAN* (20%)	0.0092
AugCGAN (0%)	**0.0079**
AugCGAN (10%)	**0.0052**

Table 2. MSE on edges given shoes in the test set. * From (Gan et al., 2017). †Same architecture as our model.

et al., 2017). These methods also rely on some notion of cycle-consistency over domains in order to constrain the learned mappings. They rely heavily on the power of the RNN-based decoders to capture complex relationships across domains while we propose to use auxiliary latent variables. The two approaches may be synergistic, as it was recently suggested in (Gulrajani et al., 2016).

Recently, Zhu et al. (2017b) proposed the BiCycleGAN model for learning multi-modal mappings but in fully supervised setting. This model extends the pix2pix framework in (Isola et al., 2017) by learning a stochastic mapping from the source to the target, and shows interesting diversity in the generated samples. Several modeling choices in BiCycleGAN resemble our proposed model, including the use of stochastic mappings and an encoder to handle multi-modal targets. However, our approach focuses on unsupervised many-to-many mappings, which allows it to handle domains with no or very little paired data.

9.5. Experiments

9.5.1. Edges-to-Photos

We first study a one-to-many image translation task between edges (domain A) and photos of shoes (domain B).[3] Training data is composed of almost 50K shoe images with corresponding edges (Yu and Grauman, 2014; Zhu et al., 2016; Isola et al., 2017), but as in previous approaches (e.g., (Kim et al., 2017)), we assume no pairing information while training unsupervised models. Stochastic mappings in our Augmented CycleGAN (AugCGAN) model are based on ResNet conditional image generators of (Zhu et al., 2017a), where we inject noise with CIN to all intermediate layers. As baselines, we train: CycleGAN, Stochastic CycleGAN (StochCGAN) and Triangle-GAN (Δ-GAN) of (Gan et al., 2017) which share the same architectures and training procedure for fair comparison. [4]

[3]Public code available at: `https://github.com/aalmah/augmented_cyclegan`

[4]Δ-GAN architecture differs only in the two discriminators, which match conditionals/joints instead of marginals.

Quantitative Results. First, we evaluate conditionals learned by each model by measuring the ability of the model of generating a specific edge-shoe pair from a test set. We follow the same evaluation methodology adopted in (Metz et al., 2016; Xiang and Li, 2017), which opt for an inference-via-optimization approach to estimate the reconstruction error of a specific shoe given an edge. Specifically, given a trained model with mapping G_{AB} and an edge-shoe pair (a,b) in the test set, we solve the optimization task $z_b^* = \arg\min_{z_b} \|G_{AB}(a,z_b) - b\|_1$ and compute reconstruction error $\|G_{AB}(a,z_b^*) - b\|_1$. Optimization is done with RMSProp as in (Xiang and Li, 2017). We show the average errors over a predefined test set of 200 samples in Table 1 for: AugCGAN (unsupervised and semi-supervised with 10% paired data), unsupervised CycleGAN and StochCGAN, and a semi-supervised Δ-GAN, all sharing the same architecture. Our unsupervised AugCGAN model outperforms all baselines including semi-supervised Δ-GAN, which indicates that reconstruction-based cycle-consistency is more effective in learning conditionals than the adversarial approach of Δ-GAN. As expected, adding 10% supervision to AugCGAN improves shoe predictions further. In addition, we evaluate edge predictions given real shoes from test set as well. We report mean squared error (MSE) similar to (Gan et al., 2017), where we normalize over all edge pixels. The Δ-GAN model with our architecture outperforms the one reported in (Gan et al., 2017), but is outperformed by our unsupervised AugCGAN model. Again, adding 10% supervision to AugCGAN reduces MSE even further.

Qualitative Results. We qualitatively compare the mappings learned by our model AugCGAN and StochCGAN. Fig. 5 shows generated images of shoes given an edge $a \sim p_d(a)$ (row) and $z_b \sim p(z_b)$ (column) from both model, and Fig. 4 shows cycles starting from edges and shoes. Note that here the edges are sampled from the data distribution and not produced by the learnt stochastic mapping G_{BA}. In this case, both models can (i) generate diverse set of shoes with color variations mostly defined by z_b, and (ii) perform reconstructions of both edges and shoes.

While we expect our model to achieve these results, the fact that StochCGAN can reconstruct shoes perfectly without an inference model may seem at first surprising. However, this can be explained by the "steganography" behavior of CycleGAN (Chu et al., 2017): the model hides in the generated edge $\tilde{a}$ imperceptible information about a given shoe b (e.g., its color), in order to satisfy cycle-consistency without being penalized by the discriminator on A. A good model of the true conditionals $p(b|a)$, $p(a|b)$ should reproduce the hidden joint distribution and consequently the marginals by alternatively sampling from conditionals. Therefore, we examine the behavior of the models when edges are generated from the model itself (instead of the empirical data distribution). In Fig. 6, we plot multiple generated shoes given an edge generated by the model, i.e., $\tilde{a}$, and 5 different z_b sampled from $p(z_b)$. In StochCGAN, the mapping $G_{BA}(\tilde{a}, z_b)$ collapses to a deterministic function generating a single shoe for every z_b. This distinction between behaviour on real and synthetic data is

undesirable, e.g., regularization benefits of using unpaired data may be reduced if the model slips into this "regime switching" style. In AugCGAN, on the other hand, the mapping seem to closely capture the diversity in the conditional distribution of shoes given edges. Furthermore, in Fig. 7, we run a Markov chain by generating from the learned mappings multiple times, starting from a real shoe. Again AugCGAN produces diverse samples while StochCGAN seems to collapse to a single mode.

We investigate "steganography" behavior in both AugCGAN and StochCGAN using a similar approach to (Chu et al., 2017), where we corrupt generated edges with noise sampled from $\mathcal{N}(0,\epsilon^2)$, and compute reconstruction error of shoes. Fig. 8 shows L_1 reconstruction error as we increase ϵ. AugCGAN seems more robust to corruption of edges than in StochCGAN, which confirms that information is being stored in the latent codes instead of being completely hidden in generated edges.

9.5.2. Male-to-Female

We study another image translation task of translating between male and female faces. Data is based on CelebA dataset (Liu et al., 2015) where we split it into two separate domains using provided attributes. Several key features distinguish this task from other image-translation tasks: (i) there is no predefined correspondence in real data of each domain, (ii) the relationship is many-to-many between domains, as we can map a male to female face, and vice-versa, in many possible ways, and (iii) capturing realistic variations in generated faces requires transformations that go beyond simple color and texture changes. The architecture of stochastic mappings are based on U-NET conditional image generators of (Isola et al., 2017), and again with noise injected to all intermediate layers. Fig. 9 shows results of applying our model to this task on 128×128 resolution CelebA images. We can see that our model depicts meaningful variations in generated faces without compromising their realistic appearance. In Fig. 10 we show 64×64 generated samples in both domains from our model ((a) and (b)), and compare them to both: (c) our model but with noise injected noise only in last 3 layers of the G_{AB}'s network, and (d) StochCGAN with the same architecture. We can see that in Fig. 10-(c) variations are very limited, which highlights the importance of processing latent code with multiple layers. StochCGAN in this task produces almost no variations at all, which highlights the importance of proper optimization of cycle-consistency for capturing meaningful variations. We verify these results quantitatively using LPIPS distance (Zhang et al., 2018), where we average distance between 1000 pairs of generated female faces (10 random pairs from 100 male faces). AugCGAN (Fig. 10-(b)) achieves highest LPIPS diversity score with 0.108 ± 0.003, while AugCGAN with z in low-level layers (Fig. 10-(c)) gets 0.059 +/- 0.001, and finally StochCGAN (Fig. 10-(d)) gets 0.008 +/- 0.000, i.e., severe mode collapse.

Model	P@10 / NDCG@10	
	$s = 1\%$	$s = 10\%$
Triple-GAN†	40.97 / 50.74	62.13 / 73.56
Δ-GAN†	53.21 / 58.39	63.68 / 75.22
Baseline Classifier	63.36 / 79.25	67.34 / 84.21
AugCGAN	**64.38 / 80.59**	**68.83 / 85.51**

Table 3. CelebA semi-supervised attribute prediction with supervision $s = 1\%$ and 10% . † From (Gan et al., 2017).

9.5.3. Attributes-to-Faces

In this task, we make use of the CelebA dataset in order map from descriptive facial attributes A to images of faces B and vice-versa. We report both quantitative and qualitative results. For the quantitative results, we follow (Gan et al., 2017) and test our models in a semi-supervised attribute prediction setting. We let the model train on all the available data without the pairing information and only train with a small amount of paired data as described in Sec. 3.2. We report Precision (P) and normalized Discounted Cumulative Gain (nDCG) as the two metrics for multi-label classification problems. As an additional baseline, we also train a supervised classifier (which has the same architecture as G_{BA}) on the paired subset. The results are reported in Table 3. In Fig. 11, we show some generation obtained from the model in the direction attributes to faces. We can see that the model generates reasonable diverse faces for the same set of attributes.

9.6. Conclusion

In this chapter, we have introduced the Augmented CycleGAN model for learning many-to-many cross-domain mappings in unsupervised fashion. This model can learn stochastic mappings which leverage auxiliary noise to capture multi-modal conditionals. Our experimental results verify quantitatively and qualitatively the effectiveness of our approach in image translation tasks. Furthermore, we apply our model in a challenging task of learning to map across attributes and faces, and show that it can be used effectively in a semi-supervised learning setting.

(a) AugCGAN **(b)** StochCGAN

Fig. 4. Given an edge from the data distribution (leftmost column), we generate shoes by sampling five $z_b \sim p_z(z_b)$. Models generate diverse shoes when edges are from the data distribution.

(a) AugCGAN **(b)** StochCGAN

Fig. 5. Cycles from both models starting from a real edge and a real shoe (left and right respectively in each subfigure). The ability for StochCGAN to reconstruct shoes is surprising and is due to the "steganography" effect (see text).

(a) AugCGAN **(b)** StochCGAN

Fig. 6. Given a shoe from the data distribution (leftmost column), we generate an edge using the model (second column). Then, we generate shoes by sampling five $z_b \sim p(z_b)$. When edges are generated by the model, StochCGAN collapses to a single mode of the shoes distribution and generate the same shoe.

(a) AugCGAN **(b)** StochCGAN

Fig. 7. We perform multiple generation cycles from the model by applying the learned mappings in turn. StochCGAN cycles collapse to the same shoe at each step which indicates that it doesn't capture the data distribution.

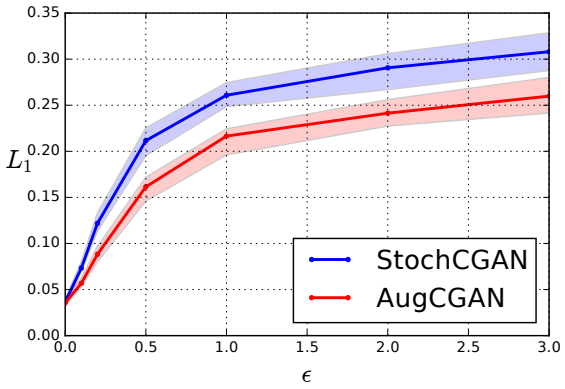


Fig. 8. Shoes reconstruction error given a generated edge as a function of the Gaussian noise ϵ injected in the generated edge.

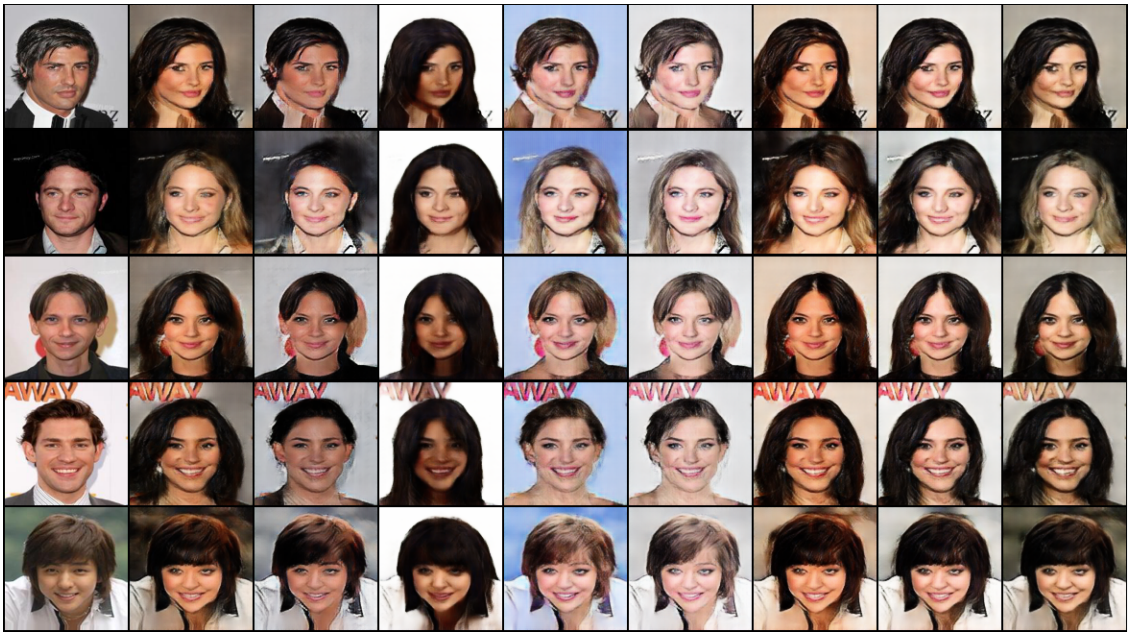
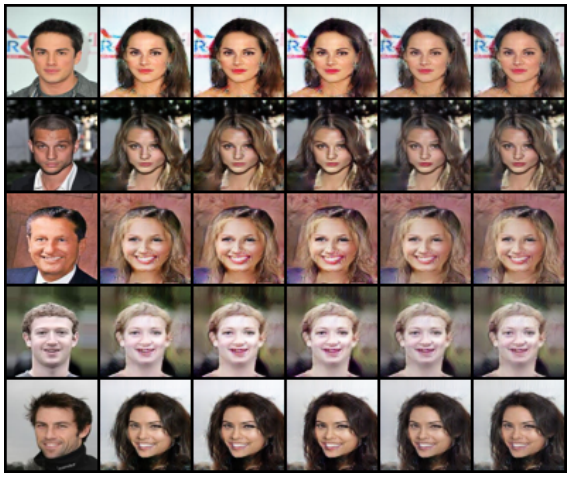


Fig. 9. Given a male face from the data distribution (leftmost column), we generate 8, 128×128 female faces with AugCGAN by sampling $z_b \sim p(z_b)$.

(a) AugCGAN Female-to-Male

(b) AugCGAN Male-to-Female

(c) AugCGAN – z in last 3 layers only



(d) StochCGAN

Fig. 10. Generated 64×64 faces given a real face image from the other domain and multiple latent codes from prior.

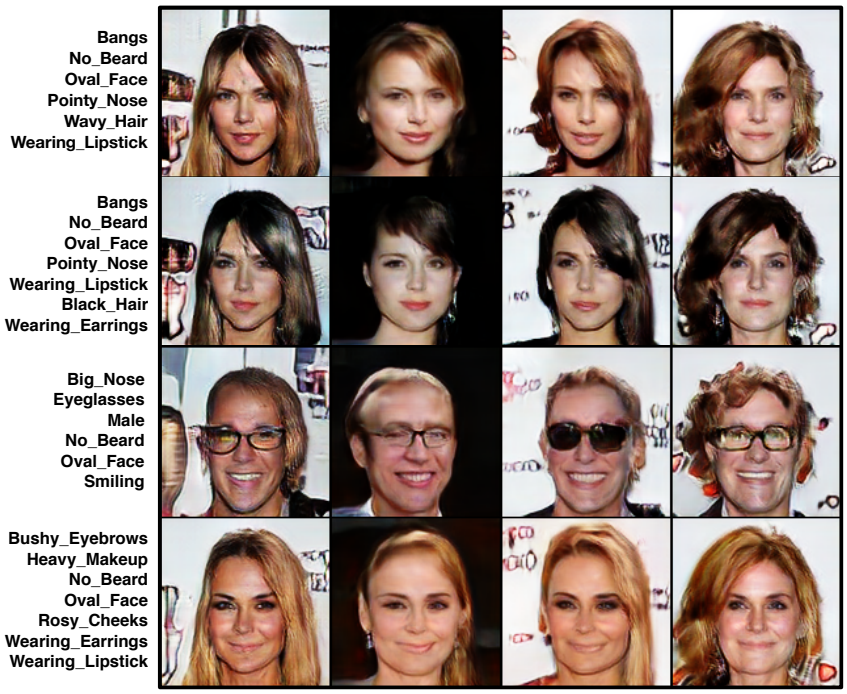


Fig. 11. Conditional generation given attributes learned by our model in the Attributes-to-Faces task. We sample a set of attributes from the data distribution and generate 4 faces by sampling latent codes from $z_b \sim p(z_b)$.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA. USENIX Association.

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM.

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., and Penn, G. (2012). Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4277–4280. IEEE.

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., et al. (2016). Theano: A python framework for fast computation of mathematical expressions. *arxiv preprint*.

Almahairi, A., Ballas, N., Cooijmans, T., Zheng, Y., Larochelle, H., and Courville, A. (2016). Dynamic capacity networks. In *International Conference on Machine Learning*, pages 2549–2558.

Almahairi, A., Kastner, K., Cho, K., and Courville, A. (2015). Learning distributed representations from reviews for collaborative filtering. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 147–154. ACM.

Almahairi, A., Rajeswar, S., Sordoni, A., Bachman, P., and Courville, A. (2018). Augmented cyclegan: Learning many-to-many mappings from unpaired data. In *International Conference on Machine Learning*.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223.

Artetxe, M., Labaka, G., Agirre, E., and Cho, K. (2018). Unsupervised neural machine translation. *ICLR*.

Ba, J., Mnih, V., and Kavukcuoglu, K. (2014). Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Bao, Y., Fang, H., and Zhang, J. (2014). TopicMF: Simultaneously exploiting ratings and reviews for recommendation. In *AAAI*.

Barkan, O. and Koenigstein, N. (2016). Item2vec: neural item embedding for collaborative filtering. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*, pages 1–6. IEEE.

Bengio, E., Bacon, P.-L., Pineau, J., and Precup, D. (2015). Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*.

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and trends in Machine Learning*, 2(1):1–127.

Bengio, Y. (2013). Deep learning of representations: Looking forward. In *Statistical Language and Speech Processing*, pages 1–37. Springer.

Bengio, Y. and Bengio, S. (2000). Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems*, pages 400–406.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Bucilua, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541. ACM.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

Campos, V., Jou, B., i Nieto, X. G., Torres, J., and Chang, S.-F. (2018). Skip RNN: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*.

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

Catherine, R. and Cohen, W. (2017). Transnets: Learning to transform for recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 288–296. ACM.

Chen, W., Zhang, Z., Li, Z., and Zhang, M. (2016a). Distributed representations for building profiles of users and items from text reviews. In *COLING*.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016b). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180.

Chen, Y., Guan, T., and Wang, C. (2010). Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Chu, C., Zhmoginov, A., and Sandler, M. (2017). Cyclegan: a master of steganography. *arXiv preprint arXiv:1712.02950*.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). *ICLR*.

Dai, Z., Yang, Z., Yang, F., Cohen, W. W., and Salakhutdinov, R. R. (2017). Good semi-supervised learning that requires a bad gan. In *Advances in Neural Information Processing Systems*, pages 6510–6520.

Denil, M., Bazzani, L., Larochelle, H., and de Freitas, N. (2012). Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8):2151–2184.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, pages 1269–1277.

Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., and Courville, A. (2016). Adversarially learned inference. *arXiv preprint arXiv:1606.00704*.

Dumoulin, V., Shlens, J., and Kudlur, M. (2017). A learned representation for artistic style.

Fedus, W., Rosca, M., Lakshminarayanan, B., Dai, A. M., Mohamed, S., and Goodfellow, I. (2018). Many paths to equilibrium: Gans do not need to decrease a divergence at every step. *ICLR*.

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., and Salakhutdinov, R. (2017). Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1039–1048.

Gan, Z., Chen, L., Wang, W., Pu, Y., Zhang, Y., Liu, H., Li, C., and Carin, L. (2017). Triangle generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 5253–5262.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. (2014). Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115.

Gonzalez-Garcia, A., van de Weijer, J., and Bengio, Y. (2018). Image-to-image translation for cross-domain disentanglement. In *Advances in Neural Information Processing Systems*, pages 1287–1298.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*.

Graves, A. (2013). Generating sequences with recurrent neural networks. Technical report, arXiv:1308.0850.

Graves, A. (2016). Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *ICML)*.

Gregor, K., Danihelka, I., Graves, A., and Wierstra, D. (2015). Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777.

Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A. A., Visin, F., Vazquez, D., and Courville, A. (2016). Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2016). Session-based recommendations with recurrent neural networks. In *ICLR*.

Hill, F., Cho, K., and Korhonen, A. (2016). Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1367–1377. Association for Computational Linguistics.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hinton, G. E. (1999). Products of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN)*, volume 1, pages 1–6, Edinburgh, Scotland. IEE.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Distributed Representations, pages 77–109. MIT Press, Cambridge, MA, USA.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hoffman, J., Tzeng, E., Park, T., Zhu, J.-Y., Isola, P., Saenko, K., Efros, A. A., and Darrell, T. (2017). Cycada: Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv:1711.03213*.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Huang, X., Li, Y., Poursaeed, O., Hopcroft, J., and Belongie, S. (2017). Stacked generative adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1866–1875. IEEE.

Huang, X., Liu, M.-Y., Belongie, S., and Kautz, J. (2018). Multimodal unsupervised image-to-image translation. In *European Conference on Computer Vision*.

Ilin, A. and Raiko, T. (2010). Practical approaches to principal component analysis in the presence of missing values. *The Journal of Machine Learning Research*, 11:1957–2000.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *arXiv preprint*.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2015). Spatial transformer networks. *arXiv preprint arXiv:1506.02025*.

Jaderberg, M., Vedaldi, A., and Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions. In *BMVC*.

Jégou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE TPAMI*, 33(1):117–128.

Jernite, Y., Grave, E., Joulin, A., and Mikolov, T. (2016). Variable computation in recurrent neural networks. In *International Conference on Learning Representations*.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kim, T. and Bengio, Y. (2016). Deep directed generative models with energy-based probability estimation. *arXiv preprint arXiv:1606.03439*.

Kim, T., Cha, M., Kim, H., Lee, J., and Kim, J. (2017). Learning to discover cross-domain relations with generative adversarial networks. *arXiv preprint arXiv:1703.05192*.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kumar, R., Goyal, A., Courville, A., and Bengio, Y. (2019). Maximum entropy generators for energy-based models. *arXiv preprint arXiv:1901.08508*.

Lample, G., Denoyer, L., and Ranzato, M. (2018a). Unsupervised machine translation using monolingual corpora only. *ICLR*.

Lample, G., Ott, M., Conneau, A., Denoyer, L., and Ranzato, M. (2018b). Phrase-based & neural unsupervised machine translation. *EMNLP*.

Larochelle, H. and Hinton, G. E. (2010). Learning to combine foveal glimpses with a third-order boltzmann machine. In *Advances in neural information processing systems*, pages

1243–1251.

Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37.

Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. *CoRR*, abs/1405.4053.

Lee, H.-Y., Tseng, H.-Y., Huang, J.-B., Singh, M. K., and Yang, M.-H. (2018). Diverse image-to-image translation via disentangled representations. In *European Conference on Computer Vision*.

Li, C., Liu, H., Chen, C., Pu, Y., Chen, L., Henao, R., and Carin, L. (2017). Alice: Towards understanding adversarial learning for joint distribution matching. In *Advances in Neural Information Processing Systems*, pages 5495–5503.

Lin, J., Rao, Y., Lu, J., and Zhou, J. (2017). Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2178–2188.

Ling, G., Lyu, M. R., and King, I. (2014). Ratings meet reviews, a combined approach to recommend. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 105–112, New York, NY, USA. ACM.

Liu, L. and Deng, J. (2017). Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *arXiv preprint arXiv:1701.00299*.

Liu, M.-Y., Breuel, T., and Kautz, J. (2017). Unsupervised image-to-image translation networks. In *Advances in Neural Information Processing Systems 30*, pages 700–708.

Liu, M.-Y. and Tuzel, O. (2016). Coupled generative adversarial networks. In *Advances in neural information processing systems*, pages 469–477.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I. (2016). Adversarial autoencoders. In *International Conference on Learning Representations*.

McAuley, J. and Leskovec, J. (2013). Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 165–172, New York, NY, USA. ACM.

McGill, M. and Perona, P. (2017). Deciding how to decide: Dynamic routing in artificial neural networks. In *International Conference on Machine Learning*, pages 2363–2372.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2016). Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*.

Mikolov, T. (2012a). Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*.

Mikolov, T. (2012b). *Statistical Language Models based on Neural Networks*. PhD thesis, Brno University of Technology.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations: Workshops Track*.

Mnih, A. and Salakhutdinov, R. (2007). Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264.

Mnih, V., Heess, N., Graves, A., et al. (2014). Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212.

Mor, N., Wolf, L., Polyak, A., and Taigman, Y. (2018). A universal music translation network. *arXiv preprint arXiv:1805.07848*.

Nedelec, T., Smirnova, E., and Vasile, F. (2017). Specializing joint representations for the task of product recommendation. *arXiv preprint arXiv:1706.07625*.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5. Granada, Spain.

Ng, A. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670.

Nowozin, S., Cseke, B., and Tomioka, R. (2016). f-gan: Training generative neural samplers using variational divergence minimization. *arXiv preprint arXiv:1606.00709*.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2017). Film: Visual reasoning with a general conditioning layer. *arXiv preprint arXiv:1709.07871*.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Ranzato, M. (2014). On learning where to look. *arXiv preprint arXiv:1405.5488*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286.

Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. (2011). *Recommender systems handbook*, volume 1. Springer.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2014). Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*.

Roth, K., Lucchi, A., Nowozin, S., and Hofmann, T. (2017). Stabilizing training of generative adversarial networks through regularization. In *Advances in Neural Information Processing Systems*, pages 2018–2028.

Royer, A., Bousmalis, K., Gouws, S., Bertsch, F., Moressi, I., Cole, F., and Murphy, K. (2017). Xgan: Unsupervised image-to-image translation for many-to-many mappings. *arXiv preprint arXiv:1711.05139*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

Salakhutdinov, R. and Mnih, A. (2008). Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*.

Schaul, T., Zhang, S., and LeCun, Y. (2013). No more pesky learning rates. In *International Conference on Machine Learning*, pages 343–351.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

Shen, T., Lei, T., Barzilay, R., and Jaakkola, T. (2017). Style transfer from non-parallel text by cross-alignment. *arXiv preprint arXiv:1705.09655*.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*.

Tung, H.-Y. F., Harley, A. W., Seto, W., and Fragkiadaki, K. (2017). Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation from unpaired supervision. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 2.

Van Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756.

Vapnik, V. (1992). Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838.

Vasile, F., Smirnova, E., and Conneau, A. (2016). Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 225–232. ACM.

Wolf, L., Taigman, Y., and Polyak, A. (2017). Unsupervised creation of parameterized avatars. *arXiv preprint arXiv:1704.05693*.

Wu, C.-Y., Ahmed, A., Beutel, A., and Smola, A. J. (2016). Joint training of ratings and reviews with recurrent recommender networks.

Wu, C.-Y., Ahmed, A., Beutel, A., Smola, A. J., and Jing, H. (2017). Recurrent recommender networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 495–503. ACM.

Xiang, S. and Li, H. (2017). On the effects of batch and weight normalization in generative adversarial networks. *stat*, 1050:22.

Xu, B., Wang, N., Chen, T., and Li, M. (2015a). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015b). Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057.

Yang, Z., Chen, W., Wang, F., and Xu, B. (2018). Unsupervised neural machine translation with weight sharing. *ACL*.

Yi, Z., Zhang, H., Gong, P. T., et al. (2017). Dualgan: Unsupervised dual learning for image-to-image translation. *arXiv preprint arXiv:1704.02510*.

Yu, A. and Grauman, K. (2014). Fine-grained visual comparisons with local learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 192–199.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. (2018). The unreasonable effectiveness of deep networks as a perceptual metric. In *CVPR*.

Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

Zhao, J., Mathieu, M., and LeCun, Y. (2016). Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*.

Zheng, L., Noroozi, V., and Yu, P. S. (2017). Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 425–434. ACM.

Zhu, J.-Y., Krähenbühl, P., Shechtman, E., and Efros, A. A. (2016). Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, pages 597–613. Springer.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017a). Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*.

Zhu, J.-Y., Zhang, R., Pathak, D., Darrell, T., Efros, A. A., Wang, O., and Shechtman, E. (2017b). Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems*, pages 465–476.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438.