Deep Learning with Minimal Supervision

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE

Faculty of Engineering

BY

Gaurav Pandey

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

August, 2018

Acknowledgements

It would have been impossible for me to undertake this arduous journey without the help and support of several individuals in the past few years. I am indebted with gratitude to my advisor Prof. Ambedkar Dukkipati, who introduced me to the world of research and guided me through every step of the Ph.D. program. His acumen for identifying interesting unsolved problems solvable in a reasonable amount of time, has proved to be extremely useful during my Ph.D. Furthermore, his help and support, that often extended beyond the realm of research, made it possible for me to get through the tough times with relative ease.

During the past few years, I had the opportunity to learn from several professors that have devoted themselves to research and teaching. It was with Prof. Jayant Haritsa, that I first learned the dedication involved in conducting meaningful research. Prof. Shivani Agarwal, Prof. Chiranjib Bhattacharyya, and Prof. M. Narasimha Murty instilled an interest towards machine learning within me. Prof. Ambedkar Dukkipati, Prof. Kaushal Verma, Prof. R. Vittal Rao and Prof. Manjunath Krishnapur introduced me to the rigorous nature of mathematics. I am grateful to all the above professors for their effort in making their courses interesting and useful. In addition, I am grateful to Prof. Y. Narahari, for his efforts in ensuring the well-being of every student in the department.

In addition to brilliant professors, I also had the opportunity of interacting with several graduate students whose insights into machine learning, mathematics, and life in general, has left a lasting impression on me. Akshay, Siddharth, Mahesh, Annervaz, and Shubham kept me updated about the latest research in the field of machine learning in general and deep learning in particular. Moreover, the presence of Maria, Abhishek, and Nithish in my research lab, ensured that there was never a dull moment. The lessons that I have learned from Maria will stay with me for the rest of my life.

I was lucky to be born into a loving family, that would go to any extent to ensure the well being of other members of the family. When I faced a health crisis, my mother chose to leave her job and stay with me instead, despite the fact that this involved living away from my father. I will forever be indebted to my parents for the sacrifices that they undertook to remove any

obstacles from my path. I am grateful to my sisters for ensuring that I never had any dearth of emotional as well as economic support. They were the first friends that I ever had, and the years apart have not diminished the mutual affection that we share with each other.

Lastly, I am grateful to my amazing wife Neha, for supporting me wholeheartedly in every decision that I have made with her. Without her patience and invigoration, it would have been impossible for me to navigate the rough seas, and reach this far. Her faith in me and dedication to my well being serves as a major encouragement for me in every endeavor that I undertake.

Abstract

In recent years, deep neural networks have achieved extraordinary performance on supervised learning tasks. Convolutional neural networks (CNN) have vastly improved the state of the art for most computer vision tasks including object recognition and segmentation. However, their success relies on the presence of a large amount of labeled data. In contrast, relatively fewer work has been done in deep learning to handle scenarios when access to ground truth is limited, partial or completely absent. In this thesis, we propose models to handle challenging problems with limited labeled information.

Our first contribution is a neural architecture that allows for the extraction of infinitely many features from an object while allowing for tractable inference. This is achieved by using the 'kernel trick', that is, we express the inner product in the infinite dimensional feature space as a kernel. The kernel can either be computed exactly for single layer feedforward networks, or approximated by an iterative algorithm for deep convolutional networks. The corresponding models are referred to as stretched deep networks (SDN). We show that when the amount of training data is limited, SDNs with random weights drastically outperform fully supervised CNNs with similar architectures.

While SDNs perform reasonably well for classification with limited labeled data, they can not utilize unlabeled data which is often much easier to obtain. A common approach to utilize unlabeled data is to couple the classifier with an autoencoder (or its variants) thereby minimizing reconstruction error in addition to the classification error. We discuss the limitations of decoder-based architectures and propose a model that allows for the utilization of unlabeled data without the need of a decoder. This is achieved by jointly modeling the distribution of data and latent features in a manner that explicitly assigns zero probability to unobserved data. The joint probability of the data and the latent features is maximized using a two step EM-like procedure. Depending on the task, we allow the latent features to be one-hot or real-valued vectors and define a suitable prior on the features. For instance, one-hot features correspond to class labels and are directly used for the unsupervised and semi-supervised classification tasks. For real-valued features, we use hierarchical Bayesian models as priors over the latent features.

Hence, the proposed model, which we refer to as discriminative encoder (or DisCoder), is flexible in the type of latent features that it can capture. The proposed model achieves state-of-the-art performance on several challenging datasets.

Having addressed the problem of utilizing unlabeled data for classification, we move to a domain where obtaining labels is a lot more expensive, that is, semantic image segmentation. Explicitly labeling each pixel of an image with the object that the pixel belongs to, is an expensive operation, in terms of time as well as effort. Currently, only a few classes of images have been densely (pixel-level) labeled. Even among these classes, only a few images per class have pixel-level supervision. Models that rely on densely-labeled images, can not utilize a much larger set of weakly annotated images available on the web. Moreover, these models can not learn the segmentation masks for new classes, where there is no densely labeled data.

Hence, we propose a model for utilizing weakly-labeled data for semantic segmentation of images. This is achieved by generating fake labels for each image, while simultaneously forcing the output of the CNN to satisfy the mean-field constraints imposed by a conditional random field. We show that one can enforce the CRF constraints by forcing the distribution at each pixel to be close to the distribution of its neighbors. The proposed model is very fast to train and achieves state-of-the-art performance on the popular VOC-2012 dataset for the task of weakly supervised semantic image segmentation.

Publications and Preprints

1. Pandey, G. and Dukkipati, A.(2017). Unsupervised feature learning with discriminative encoder. In *Proceedings of the IEEE 17th International Conference on Data Mining (ICDM'17) (accepted).*

2. Pandey, G., & Dukkipati, A. (2014). Learning by stretching deep networks. In *Proceedings of* 31^{st} *International Conference on Machine Learning (ICML'14)*, pp. 1719-1727, 2014.

3. Pandey, G., & Dukkipati, A. (2014). To go deep or wide in learning? In *Proceedings of* 17^{th} *International Conference on Artificial Intelligence and Statistics (AISTATS'14), JMLR W&CP 33:724-732*, 2014.

4. Pandey, G. and Dukkipati, A.(2017). Training CNNs with CRFs for weakly supervised semantic image segmentation. Submitted.

5. Pandey, G. and Dukkipati, A.(2017). Deep discriminative probabilistic models. To be submitted.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, there has been a growing amount of research to map the raw representation of input (primarily, images, text and speech signals) to a feature representation suitable for the final supervised learning task. Feature extraction algorithms, such as SIFT (Lowe, 2004) and SURF (Bay et al., 2006), keep the representation learning and prediction stages separate and have been shown to obtain good performance for object recognition tasks. Furthermore, many feature learning algorithms introduced in the past decade, such as sparse coding (Lee et al., 2007), restricted Boltzmann machines (Hinton, 2002), deep Boltzmann machines (Salakhutdinov and Hinton, 2009), autoencoders (Bengio et al., 2007), $k-$means (Coates et al., 2011), also learn features independently of the classification stage. Several results have shown that using one or more layers of these feature learning algorithms is sufficient to obtain reasonable performance for many object recognition tasks (Bo et al., 2013, Coates et al., 2011).

On the other end of the spectrum, there are learning models that support the coupling of the representation learning stage with the prediction stage. Neural networks form the primary examples of such models. While multi-layer neural networks were introduced more than three decades back, the lack of a proper way to initialize their weights, the extensive time and memory required for training these models, and the lack of efficient optimization algorithms have kept these models dormant for a long time. In recent years, the development of unsupervised feature learning algorithms, new weight-initialization methods, adaptive optimization methods, and inexpensive GPUs have brought back neural networks into the main stream of machine learning research. The state-of-the-art for object recognition has been vastly improved using deep convolutional neural networks (Krizhevsky et al., 2012, Szegedy et al., 2015).

Deep networks have millions of parameters. Hence, success in deep learning often relies on the presence of a large amount of labeled training data. However, generation of labeled data can be expensive and time consuming. In contrast, the web provides an almost unlimited source

for unlabeled data. Furthermore, learning in humans often occurs without explicit supervision. Therefore, the intent of this thesis is to present methods based on deep architectures, that can either handle limited data (Chapter 3) or incorporate unlabeled and partially-labeled data into learning (Chapters 4,5 and 6). In this introductory chapter, we will motivate the necessity of deep discriminative models that can perform reasonably well even in the absence of a large amount of labeled data.

1.1 Generative vs discriminative models

One of the most distinctive feature of machine learning is its reliance on training data. In particular, the model building phase in machine learning (also known as the training phase), uses the training data to learn a model that can predict some aspects of the data. For instance, if the training data consists of sentences and their corresponding sentiments, one may be interested in learning a classifier that can predict the sentiment of an arbitrary sentence. A classical approach to address this problem, will rely on the occurrence of selected words or patterns in the sentence for determining the sentiment. In contrast, a machine learning based approach will represent the sentences as well as the sentiments as vectors (or structured collection of vectors), and then learn a mapping from the sentence vectors to the sentiment vectors.

Assume that the training data that one is interested to model, consists of pairs of input and target vectors $(\mathbf{x}^{(i)}, \mathbf{z}^{(i)})_{i=1}^{n}$. The target vectors $\mathbf{z}$ can be the category of the object present in an image or the labels of all the pixels in an image or the parse tree of a sentence or the French translation of an English sentence. A generative approach to learn a mapping from the input to target vectors will proceed by defining a parametrized probability distribution p_θ over pairs of target and input vectors $(\mathbf{x}, \mathbf{z})$. The parameters of the distribution θ can then be learnt so as to maximize the probability of the observed data. Such a procedure is known as maximum likelihood estimation. Alternatively, one can choose the parameters to minimize some distance measure between the true data distribution p_d and the modeling distribution p_θ. Since the true data distribution p_d is unknown, the empirical distribution over the observed pairs $(\mathbf{x}, \mathbf{z})$, is often chosen as the data distribution. For specific choice of distance measures, this approach coincides with maximum likelihood estimation.

Since one can sample from the modelling distribution $p_\theta(\mathbf{x}, \mathbf{z})$ to generate new pairs of $\mathbf{x}$ and $\mathbf{z}$, the model discussed above is referred to as generative model. The target vectors for new data can then be inferred using the Bayes' rule:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}\,.$$

In contrast, a discriminative statistical approach to address the same problem will proceed by directly defining a parametrized probability distribution for the target vectors given the input vectors, that is, $p_\theta(\mathbf{z}|\mathbf{x})$ is modeled directly. The conditional likelihood of the target vectors given the input vectors is then maximized. Other discriminative models try to learn a parametrized mapping f_θ from the input $\mathbf{x}$ to the targets $\mathbf{z}$ by minimizing a loss function between the predicted and the observed target vectors. For instance, if $\mathbf{z}$ is a real-valued vector, and the loss function is the squared loss, the total loss ℓ for a given set of observed training data $(\mathbf{x}^{(i)}, \mathbf{z}^{(i)})_{i=1}^n$ is

$$\ell(\theta) = \sum_{i=1}^{n} ||f_\theta(\mathbf{x}^{(i)}) - \mathbf{z}^{(i)}||^2 .$$

One of the advantages of discriminative models over their generative counterparts, is that one need not make any assumption about the parametric form for the distribution of input vectors. For most supervised learning tasks, discriminative models consistently achieve asymptotically superior performance as compared to their generative counterparts. In contrast, generative models perform well with limited labeled data (Ng and Jordan, 2002). Furthermore, it is straightforward to incorporate unlabeled data in generative models, by maximizing the marginal probability of the observed input. For instance, if $\mathbf{z}$ is unlabeled, one can choose to maximize

$$\log p_\theta(\mathbf{x}) = \log \sum_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) .$$

Despite the simplicity of the approach, generative models are rarely used for incorporating unlabeled data. This is primarily since the specification of the distribution over $\mathbf{x}$ can be quite challenging, especially when $\mathbf{x}$ is a text, image or speech signal. Rather, most successful methods for incorporating unlabeled data for classification rely on heuristics, such as label propagation (Chapelle et al., 2009), minimum entropy (Grandvalet and Bengio, 2005) etc., in a semi-supervised setting. Hence, in Chapters 4 and 5 of this thesis, we discuss a principled approach for incorporating unlabeled data into the model without resorting to a generative approach.

1.2 From classical machine learning to deep learning

For concreteness, let us focus on the problem of object recognition. For several years, we have known that the visual cortex in mammals is organized into several regions in a hierarchical manner (Hubel and Wiesel, 1962). The primary visual cortex (V1) receives signals from the retina (via lateral geniculate nucleus). Each neuron in this region has a small receptive field, and hence the mapping between the locations in the V1 region and the actual visual field is

very precise. The neurons in this region are highly tuned to small changes in orientation, color and spatial frequency.

Traditional approaches for object recognition in computer vision such as histogram of oriented gradients (HOG) (Dalal and Triggs, 2005) and SIFT descriptors (Lowe, 2004) mimic the functionality of the V1 region of the visual cortex by describing the orientation of gradient in localized regions of the image. These descriptors are then binned, locally normalized and fed to a classifier such as support vector machine. Classification of speech and text proceeds similarly with the sole difference in the mechanism used in extracting features from the raw data.

The feature extraction stage in recognition is domain-specific and demands the expertise of domain-specific researchers. From the perspective of machine learning, the interesting part of the recognition stage is the final classification stage. To keep matters simple, let us focus on linear classification. Let $\mathbf{x}$ be the input and $\phi(\mathbf{x})$ be the features extracted by a domain-specific feature extractor. A linear classifier assumes that the final class label of the instance $\mathbf{x}$ can be obtained from a linear function of the extracted features, as given below

$$\mathbf{z} = \underset{1 \leq \ell \leq L}{\operatorname{argmax}} \, w_\ell^{\mathrm{T}} \phi(\mathbf{x}) + b_\ell \,, \tag{1.1}$$

where $w_1, b_1, \ldots, w_L, b_L$ are class-specific parameters that need to be determined by a learning algorithm.

A linear classifier puts the burden on the feature extraction stage to extract features such that the features representation of objects from two different classes can be separated by a linear classifier. Hence, the success of the entire model depends on the features extracted by the model. It is worthy to point out that the feature extraction stage is often agnostic of the final classification stage. Unfortunately, the features that are useful for classification for one set of classes may prove to be completely worthless for a different set of classes. For instance, the features that may prove to be useful to differentiate between men and women may not be useful to differentiate between dogs and cats.

A possible approach that allows the agnostic nature of the feature extraction stage, is to extract every possible feature that may prove to be useful in the final classification stage. Alternatively, the feature extraction stage may focus only on extracting simple features such as edge orientations, that may prove to be useful irrespective of the class labels. The first approach is not very well defined, since the possible classes present in the data may not be known in advance. The second approach was commonly employed by computer vision researchers for achieving successful object recognition.

The biggest revolution introduced by deep learning lies in the demolition of the boundary

between feature extraction and classification. The success of convolutional neural networks (CNNs) (LeCun et al., 1989, Rumelhart et al., 1986, Krizhevsky et al., 2012) on raw pixels, implies that we do not need computer vision experts to determine the features that will allow for successful classification. Similarly, the success of end-to-end models for speech recognition (Amodei et al., 2016, Hannun et al., 2014) implies that it is not necessary anymore to engineer features from raw speech signals that allow for successful identification of the corresponding label. In particular, the function ϕ which was determined by computer vision experts can now be learnt directly using the chain rule for derivatives, implemented efficiently as back-propagation (Rumelhart et al., 1986).

Neural networks assume that the final class label can be obtained from the instance $\mathbf{x}$ as follows

$$\mathbf{z} = \underset{1 \leq \ell \leq L}{\operatorname{argmax}} \, w_\ell^{\mathrm{T}} \phi(\mathbf{x}) + b_\ell \,. \tag{1.2}$$

Note the similarity in the form of the equations in (1.1) and (1.2). However, the difference lies in the fact that while the function ϕ was fixed in (1.1), it is a parametrized differentiable function of $\mathbf{x}$ in (1.1). Furthermore, the function is often defined hierarchically, that is,

$$\phi(\mathbf{x}) = \phi_K(\phi_{K-1}(\dots \phi_1(\mathbf{x}) \dots)) \,,$$

where ϕ_k are simple non-linear functions of the input. This hierarchical definition of the function partially mimics the hierarchical organization of the visual cortex. Similar to the visual cortex, the layers closer to the input compute simple class-agnostic features such as edge color and orientations whereas the layers further away from the input compute more abstract features that are class-specific.

1.3 Deep learning with minimal supervision

Deep neural networks have been immensely successful for several AI tasks, such as object recognition (Krizhevsky et al., 2012, Szegedy et al., 2015) and segmentation (Long et al., 2015, Zheng et al., 2015). However, the presence of a large number of trainable parameters in the model implies that they require a large amount of labeled data to achieve low generalization error. While researchers have collected labeled data for several interesting tasks over the past few years (Russakovsky et al., 2015, Everingham et al., 2012), for most of the tasks a negligible amount of labeled data is available. Hence, a machine learning practitioner working with limited labeled data is faced with a dilemma, one can either spend several hours to generate labeled data for the task, or one can design a model that allows one to handle limited labeled data as

well as unlabeled data. The problem of generating labeled data becomes even more severe for structured prediction tasks, such as semantic segmentation of images. Explicitly labeling each pixel of an image in known to be very time consuming. In such scenarios, models that rely on limited supervision (for instance, image-level labels vs pixel-level labels) are essential. For segmentation tasks, image-level labels are referred to as weak labels.

The above problems of learning with limited supervision can be handled either using generative or discriminative models. As discussed in Section 1.1, generative models have a straightforward mechanism for incorporating data with missing labels. For instance, if a generative model is defined over pairs of input and target variables $(\mathbf{x}, \mathbf{z})$, the model can incorporate missing target variables by maximizing the marginal log-likelihood of the input $\mathbf{x}$ only. Furthermore, the amount of labeled data required by generative models for achieving their asymptotic error is much lesser than their discriminative counterparts (Ng and Jordan, 2002). Hence, generative models can address the two main problems that this thesis attempts to address.

Over the years, several deep generative models have been proposed. Of particular importance are restricted Boltzmann machines (Hinton, 2002), deep Boltzmann machines (Salakhutdinov and Hinton, 2009) and variational autoencoders (Kingma and Welling, 2013). While all these three models were initially designed to capture the marginal distribution over the input variables $\mathbf{x}$ only, extensions of these models have been proposed to incorporate the target variables $\mathbf{z}$ as well (Kingma et al., 2014, Goodfellow et al., 2013). In particular, when the target variables are absent, these models maximize the marginal log-likelihood of $\mathbf{x}$ only. However, these models have largely been replaced by discriminative models that employ heuristics for handling unlabeled data.

A commonly employed heuristic in discriminative models for classification with unlabeled data is the minimum entropy heuristic (Grandvalet and Bengio, 2005). This heuristic is based on the idea that the boundaries between the classes occur where there is scarcity of data (both labeled and unlabeled). Hence, the boundaries should be chosen in such a way, that the class for each point is predicted with high certainty. This implies that every point (whether labeled or unlabeled) must be sufficiently far from the boundary. Variations of this heuristic have been employed in deep neural networks as well (Springenberg, 2015), and have been shown to achieve better performance as compared to generative models for the same task.

Other than the minimum entropy heuristic, several task-agnostic heuristics for unsupervised learning in deep architectures have also been proposed. Most of these heuristics fall under the domain of self-supervised learning, that is, the corresponding models generate supervision without any manual involvement. For instance, Exemplar-CNNs (Dosovitskiy et al., 2014) generate multiple transformations of a patch in an image, and train a classifier that ensures

that all the transformations of a single patch are classified to the same class. Wang and Gupta (2015) use visual tracking to generate supervision. Noroozi and Favaro (2016) jumble the patches in an image, and train a CNN to predict the correct spatial ordering of the patches. In order to predict the correct ordering, the network needs to capture global information such as the shape of the jumbled object, as well as local information such as the alignment of edges. However, since these heuristics are task-agnostic, the features learnt by the corresponding model may not be suitable for the task at hand.

Finally, to address semantic segmentation of weakly-labeled images, several researchers have proposed the use of saliency masks (Kolesnikov and Lampert, 2016, Wei et al., 2016, Saleh et al., 2016, Shimoda and Yanai, 2016). A saliency mask (Itti et al., 1998) captures the importance of a pixel in an object. They can either be generated automatically (Zhou et al., 2016) for a specific task or manually obtained (Bylinskii et al., 2015) in a task-agnostic manner. Manually obtaining the saliency masks is as expensive as generating segmentation masks, and hence, the methods that automate the generation of saliency masks are desirable. Another successful heuristic for semantic segmentation of images, groups the pixels in a video that move together into a single object (Pathak et al., 2016).

1.4 Our contributions

In this thesis, we attempt to address the problems of learning with limited, unlabeled and weakly-labeled data. The first two problems occur in the domain of classification, where the problem is to assign an object to a set of predefined object categories. In this thesis, we restrict ourselves to problems in which each object must be assigned to exactly one category. The last problem of learning with weakly-labeled data occurs in the domain of semantic segmentation of images, where one is required to label each pixel present in an image with the category of the object that the pixel belongs to.

Our first contribution is a neural architecture that allows for the extraction of infinitely many features from an object while allowing for tractable inference. This is achieved by using the 'kernel trick', that is, we express the inner product in the infinite dimensional feature space as a kernel. The kernel can either be computed exactly for single layer feedforward networks, or approximated by an iterative algorithm for deep convolutional networks. The corresponding models are referred to as stretched deep networks (SDN). We show that when the amount of training data is limited, SDNs with random weights drastically outperform fully supervised CNNs with similar architectures.

To handle unlabeled data, we propose to jointly model the observed and the latent features (one can think of target variables as latent features for semi-supervised classification tasks)

using a unique distribution. In this sense, the proposed model bears similarity to generative models. However, unlike generative models, the model is completely specified by the encoding distribution of the latent features given the observed features $p_\theta(\mathbf{z}|\mathbf{x})$ and the prior distribution on the latent features $p_\theta(\mathbf{z})$. In particular, we do not model the distribution of input features $\mathbf{x}$. In order to learn the latent features, we maximize the joint log-likelihood with respect to the parameters of the encoding and prior distributions as well as the latent features themselves. We propose an adversarial regularization scheme to learn categorical latent features for unsupervised and semi-supervised classification tasks.

Next, we explore the possibility of incorporating hierarchical Bayesian priors on the latent features. In particular, we employ Gaussian mixture and Gaussian topic models as priors on the real-valued latent features and use it for training the encoding network in a discriminative encoder. The proposed models, which are referred to as GMM-DisCoder and GTM-DisCoder respectively, are capable of discovering meaningful clusters and topics in the latent space. Furthermore, GMM-DisCoder can also be used for discovering unlabeled classes in challenging datasets, when few of the classes are labeled.

Finally, we address the challenging problem of segmenting images from weakly-annotated images only. In particular, we need to predict the distribution over the classes at each pixel location. We achieve this by forcing the distribution at each pixel to be close to the neighborhood distribution. We devise two efficient schemes for computing the neighborhood distribution at each pixel location, and show that one of the schemes occurs as a consequence of using a conditional random field prior on the segmentation masks. The proposed model achieves the state-of-the-art for semantic segmentation of images when no pixel level information is available.

1.5 Outline

The rest of the thesis is organized as follows:

Chapter 2: Preliminaries and background

In this chapter, we discuss the architecture and training of feedforward neural networks and cover the background material on deep models for unsupervised feature learning. Rather than discussing all possible layers in a neural network, we limit ourselves to the layers that are used in the networks employed in our experiments. The optimization algorithms used for training these networks are discussed thereafter. Next, we discuss the unsupervised learning algorithms commonly employed in training deep architectures and focus on the limitations of decoder based architectures such as autoencoders and variational autoencoders. In particular, we motivate the need of models that do not employ pixel-level distance measures for learning representations

from unlabeled data. Finally, we review the mechanism by which this problem is handled by generative adversarial networks (Goodfellow et al., 2014).

Chapter 3: Stretched deep networks for learning with limited data

In this chapter, we propose a model inspired by deep architectures that can be used for modeling when the amount of labeled data is limited. We introduce the concept of stretched neural networks, whereby the weights are obtained by taking random linear combinations of the weight vectors in a neural network. We use the derivation of arc-cosine kernels in (Cho and Saul, 2009), to compute the inner product between hidden representations of a stretched neural network, for arbitrary input pairs $(\mathbf{x}, \mathbf{y})$. The resultant inner product corresponds to the covariance arc-cosine kernel (Pandey and Dukkipati, 2014b) between the input pairs. In order to incorporate translational invariance in the kernel, we replace the linear layers in the stretched network by convolution and pooling layers. The resultant kernel can be approximated by iterating through the stretched network multiple times. We show that the resultant kernel drastically outperforms CNNs with similar architecture when the amount of labeled data is limited. Finally, we discuss the limitations of stretched networks as compared to standard neural networks.

Chapter 4: Discriminative encoders for learning with unlabeled data

In this chapter, we propose a model, referred to as discriminative encoder (DisCoder) for learning latent features from unlabeled data. When these latent features are categorical random variables, they correspond to the missing class labels in a semi-supervised or unsupervised classification problem. Alternatively, these latent features may be real-valued with hierarchical Bayesian priors defined on these features. In this chapter, we focus on the first scenario, and assume the latent features to be categorical random variables, that represent the missing class labels.

To motivate discriminative encoders, we briefly review the limitation of generative models in learning meaningful latent representations. We derive the unique distribution of discriminative encoders by imposing a set of constraints on the desired model. We parameterize the discriminative encoder using a neural network. Thereafter, we propose a minibatch-based optimization strategy for training discriminative encoders. For learning categorical latent features, we propose the use of adversarial regularization in discriminative encoders. Our experiments for semi-supervised and unsupervised classification on several challenging datasets, demonstrate the effectiveness of the proposed model.

Chapter 5: Discriminative encoding with hierarchical Bayesian priors

In this chapter, we learn real-valued latent features using discriminative encoders by imposing hierarchical Bayesian priors on the latent features. In particular, we focus on Gaussian mixture models (GMM) and Gaussian topic models (GTM) as priors on the latent features. The resultant models are referred to as GMM-DisCoder and GTM-DisCoder. We briefly review the concepts of Gaussian mixture and Gaussian topic modeling. Thereafter, we employ these distributions as priors on the latent features of a discriminative encoder. We propose a learning algorithm for jointly inferring the parameters of the hierarchical Bayesian prior, as well as learning the weights of the neural network that parameterizes the discriminative encoder. We compare the performance of GMM-DisCoder against a categorical DisCoder for clustering on simple datasets. Furthermore, on challenging datasets, we compare the clustering performance of discriminative encoders against other deep architectures, assuming that some of the classes have been labeled. To check the sanity of the proposed GTM-DisCoder, we employ it for learning topics from the bag of words representation of documents in a corpus. Finally, we create a collection of 'bag of images', and train the GTM-DisCoder to identify the topic of each image.

Chapter 6: semantic segmentation of images with weakly-labeled data

In this chapter, we focus on the task of semantic segmentation of images, when only image-level labels are available for the images. We briefly review the architecture of pairwise conditional random fields used in semantic segmentation of images. We propose a model to generate fake pixel-level labels from image-level labels based on the output of the CNN. We use the heuristic that pixels with similar color that lie close to each other should have the same label. Hence, we force the output distribution at each pixel to be close to the distribution computed from its neighbors. The resultant approach is shown to be equivalent to enforcing mean-field constraints, when the prior on the segmentation masks is a pairwise conditional random field. The model achieves the state-of-the-art for semantic segmentation of images, when no pixel-level information is available.

Chapter 2

Preliminaries and Background

In this chapter, we will review the fundamentals of deep neural networks and deep unsupervised learning algorithms used in this thesis.

2.1 Feedforward neural networks

A deep architecture is a computing system composed of several processing layers organized in a hierarchical manner. Deep neural networks are examples of deep architectures, where each processing layer is composed of neurons. Each neuron in a neural network computes a parametrized non-linear function of the input data. Several variations of neural networks have been proposed in literature. In this thesis, we will limit ourselves to feedforward neural networks.

2.1.1 Layers of the network

A convenient way of representing a neural network is to divide its architecture into layers. The layers are organized in a hierarchy. Each layer in a feedforward neural network computes a differentiable function of output of the previous layer. The most commonly employed layers of a feedforward neural network are discussed below.

Fully connected layer: This layer computes a linear function of the input. It is parameterized by a weight matrix and a bias vector. The input to this layer is expected to be a vector $\mathbf{x}$. If W is the weight matrix and b is the bias vector, the output of this layer is given by

$$\mathbf{h} = W^{\mathrm{T}}\mathbf{x} + b\,.$$

Non-linearity layer: This layer applies an elementwise non-linearity to the input. The popular non-linearities are given below:

1. **Sigmoid**: The output of a layer computing the sigmoid non-linearity is given by

$$h_k = \frac{1}{1 + \exp(-x_k)}\,,$$

where h_k and x_k are the respective components of $\mathbf{h}$ and $\mathbf{x}$.

2. **Tanh**: The output of a layer computing the tanh non-linearity is given by

$$h_k = \frac{\exp(x_k) - \exp(-x_k)}{\exp(x_k) + \exp(-x_k)}\,.$$

3. **ReLU**: The output of a layer computing the rectified linear non-linearity is given by

$$h_k = \max(0, x_k)\,.$$

4. **Leaky ReLU:** While the ReLU non-linearity completely blocks the negative part of the input, a leaky ReLU non-linearity scales the negative values by a scaling factor less than 1. In particular, if the scaling factor is ℓ, the output of this layer is given by

$$h_k = \begin{cases} x_k, \text{ if } x_k > 0\,, \\ \ell x_k \text{ if } x_k \leq 0\,. \end{cases}$$

Convolution layer: This layer is motivated by the neurons in the V1 area of the visual cortex. The neurons in a convolution layer are organized in a grid fashion. Each neuron has a small receptive field which can be mapped to the corresponding area of the visual field. Here, the receptive field of a neuron constitutes all the locations in the input that are used in the computation of the output of the neuron. The neurons in this layer are highly tuned to the spatial distribution of the input. The input typically constitutes the feature maps of an image, and has dimensions $M \times W \times H$, where M is the number of feature maps, and W and H are the width and height of the feature maps respectively. Hence, if the location of a certain pattern in the input changes, the spatial distribution of the output of the neurons will change in an equivalent fashion. This property is referred to as equivariance.

This layer is parameterized by K filters $u_1, \ldots, u_K$. The output of this layer is computed by convolving the filters with the input.

$$\mathbf{h}_k = \mathbf{x} * u_k,\ 1 \leq k \leq K\,.$$

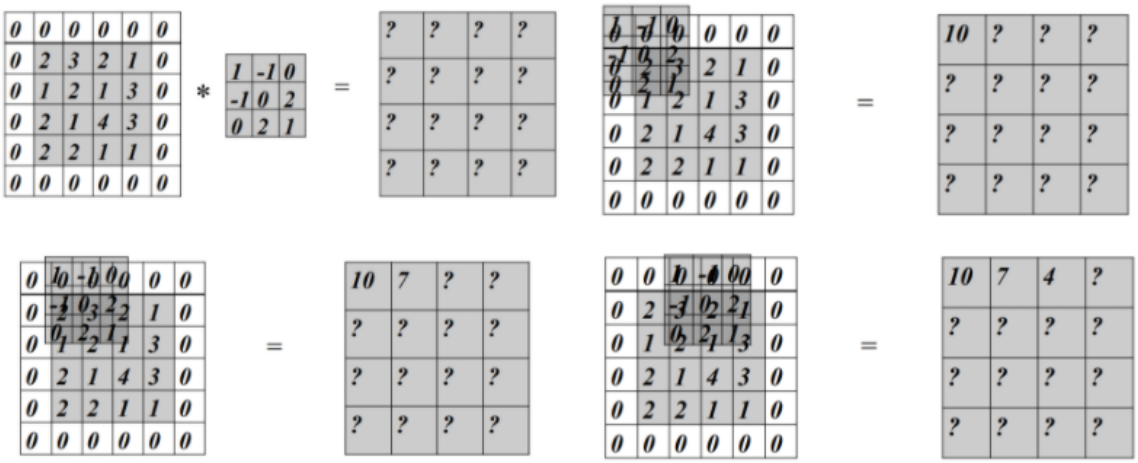


Figure 2.1: A pictorial representation of the convolution operation.

Note that each neuron in this layer computes a linear function of the input that lies within its receptive field. Furthermore, the neurons within a grid share all the parameters. A pictorial representation of the convolution operation is given in Figure 2.1.

Strided Convolution layer: Convolution operation preserves the spatial location of an input pattern. In general, it is desirable for the model to be invariant to small perturbation in the spatial location of patterns. This can be achieved by using a strided convolution operation. While in a convolution operation, the filter is applied to every adjacent overlapping region, a strided convolution operation is applied to regions that are as distant from each other as the size of the stride. A pictorial representation of strided convolution is given in Figure 2.2.

Transposed Convolution layer: The transposed convolution layer can be thought of as an attempt to reconstruct the input of a convolution layer from its output. Each entry of the input is multiplied by the filter and the resultant overlapping entries are summed together. It is implemented as the backward step of a convolution layer, and is commonly employed when the output of a CNN needs to be upsampled.

Max pooling layer: This layer pools the results of possibly overlapping patches, by computing the maximum value within the receptive field and assigning this value to the output. As in the case of convolution layers, the input is expected to be a tensor of size $M \times W \times H$, where M is the number of feature maps, and W and H are the width and height of the feature maps respectively. The result is computed individually for each feature map. Hence, the number of feature maps in the output remains the same as in the input.

Average pooling layer: This layer is similar to the max-pooling layer with the exception that the output is the average of the values within its receptive field rather than the maximum.

Weight normalization layer: This layer reparametrizes the weights of the convolution or

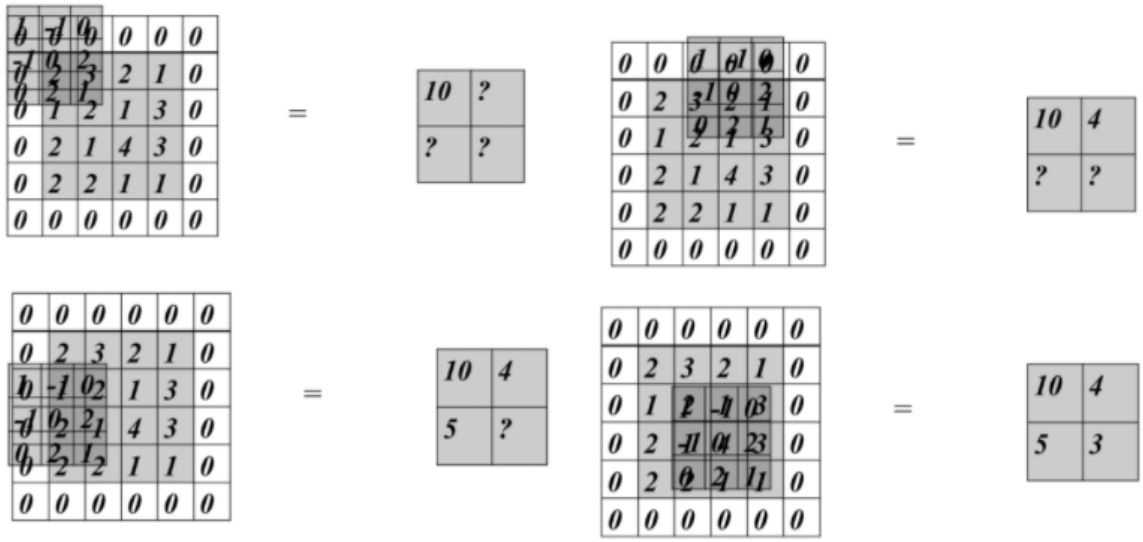


Figure 2.2: A pictorial representation of the strided convolution operation with stride 2.

fully connected layer that precedes this layer (Salimans and Kingma, 2016). In particular, if the preceding layer is a convolution layer with filters $u_1, \ldots, u_K$, this layer expresses the filters as shown below:

$$u_k = \frac{g_k}{||v_k||} v_k \,.$$

For convolutional layers closer to the input $\mathbf{x}$, g_k is often held fixed during training (Salimans et al., 2016) and only the normalized weights $\mathbf{v}_k$ are trained. This allows the network to focus on the shape of the filter rather than its magnitude, and hence, prevents the model from overfitting to the data.

Batch normalization layer: This layer normalizes the output of the previous layer over a minibatch (Ioffe and Szegedy, 2015). In particular, if the input to this layer is a minibatch of vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$, this layer computes the elementwise mean μ and the variance σ^2 of the vectors over the minibatch, where $\mu = (\mu_1, \ldots, \mu_K)$ and $\sigma^2 = (\sigma_1^2, \ldots, \sigma_K^2)$. The output of the layer is computed from the input as follows:

$$\hat{x}_k = \frac{x_k - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}} \,,$$
$$h_k = \gamma \hat{x}_k + \beta \,.$$

where ϵ is a small constant, and γ and β are parameters that can either be held fixed or trained with the rest of the parameters.

The batch normalization layer ensures that the distribution of the input at each layer remains

approximately unchanged during training, even though the exact values of the input may change by a wide margin. This speeds up the training of the network.

2.1.2 Training the network

Loss functions

In order to train the network, the output of the last layer of the network is compared with the desired output using a loss function. Depending upon the task, several loss functions can be used. In particular, let $\mathbf{y}$ be the target and $\hat{\mathbf{y}}$ be the output predicted by the network. The different loss functions used for comparing the target and the predicted values are given below:

1. **Mean squared error**: This loss is used when the target and the predicted values are real-valued vectors. The loss is given by

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = ||\mathbf{y} - \hat{\mathbf{y}}||^2 .$$

2. **Binary Cross Entropy**: This loss is used in binary classification. The predicted value is a number that lies between 0 and 1. The target belongs to the set $\{0, 1\}$. The loss is given by

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = \mathbf{y} \log \hat{\mathbf{y}} + (1 - \mathbf{y}) \log(1 - \hat{\mathbf{y}}) .$$

3. **Cross Entropy**: This loss is used in multi-class classification. The predicted value is a vector of probabilities $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_L)$. The target is a one-hot vector of the same dimension as the predicted vector $\mathbf{y} = (y_1, \ldots, y_L)$. The loss is given by

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{i=1}^{L} y_i \log \hat{y}_i ,$$

   where L is the number of labels.

Optimization algorithms

In a neural network with one or more layer of non-linearities, the loss function is a non-convex function of the weights of the network. Hence, training of a neural network almost always results in a local optima or a saddle point. The most commonly used optimization algorithm for a neural network is minibatch stochastic gradient descent (SGD), whereby the derivative of the loss is computed with respect to each parameter, and the weights of the network are updated accordingly. Hence, if θ_t are the weights of the network after t iterations of training, and $\nabla_{\theta_t} \ell$

is the gradient of the loss with respect to the parameters at time t, the update equation is given by

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \ell \,,$$

where η is the learning rate that determines the rate of convergence. If the learning rate is too small, the network will train very slowly, and a lot of iterations will be needed for reaching the optima. In contrast, if the learning rate is too high, the network may oscillate without ever reaching the local optima. Generally, the learning rate is selected using a grid search. These methods apply the same learning rate to all the parameters, which may be undesirable while traversing a region where the slope is much steeper along one of the coordinates. Hence, several modifications to standard SGD have been proposed in literature, and are listed below:

1. **Momentum**: Momentum is used to accelerate the descent in the direction across which the sign of the gradient remains unchanged over several iterations. At the same time, momentum dampens the descent in the direction where the sign of the gradient oscillates. This is obtained by computing a weighted average of the gradient of the loss with respect to the parameters over several iterations, and adding it to the current gradient. The average cancels out the directions for which the gradient oscillates whereas it strengthens the directions for which the sign of the gradient remains unchanged. This can be achieved succinctly as follows:

$$\begin{aligned}
\Delta\theta_1 &= \eta \nabla_{\theta_1} \ell \,, \\
\Delta\theta_t &= \eta \nabla_{\theta_t} \ell + \alpha \Delta\theta_{t-1} \,, \\
\theta_{t+1} &= \theta_t - \Delta\theta_t \,,
\end{aligned}$$

   where α is the momentum rate, that lies between 0 and 1. When $\alpha = 0$, this approach reduces to standard SGD. An expansion of $\Delta\theta_t$ in terms of the gradients of the previous iteration, shows that it is a weighted average of the gradients, where the weights decrease exponentially with time.

2. **Adagrad**: Adagrad (Duchi et al., 2011) is a learning strategy that assigns a different learning rate to each parameter depending upon the frequency of update of the parameter. Hence, if a parameter is updated frequently, it is assigned a low learning rate. In contrast, if a parameter is updated rarely, it is assigned a high learning rate. The algorithm keeps track of the sum of squares of the gradient with respect to each parameter. The update

equations are given below:

$$G_t = G_{t-1} + (\nabla_{\theta_t}\ell)^2\,,$$
$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}}\nabla_{\theta_t}\ell\,,$$

where ϵ is an arbitrarily small value.

3. **RMSProp**: In Adagrad, the learning rate for each weight decreases to 0 as the learning proceeds. To fix this issue, RMSProp computes an exponentially decaying weighted sum of the squares of the gradients.

$$G_t = \rho G_{t-1} + (1-\rho)(\nabla_{\theta_t}\ell)^2\,.$$

Furthermore, an exponentially decaying weighted sum of the gradients is computed as well. The square of this quantity is subtracted from G_t while computing the learning rate for the parameters

$$M_t = \rho M_{t-1} + (1-\rho)(\nabla_{\theta_t}\ell)\,,$$
$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t - M_t^2 + \epsilon}}\nabla_{\theta_t}\ell\,.$$

4. **Adam**: This method combines the momentum term with the Adagrad term in a multiplicative manner (Kingma and Ba, 2014). As in the case of RMSprop, an exponential decaying sum of the gradients and their squares is computed. These quantities are then used to compute the final update as follows:

$$G_t = \rho_1 G_{t-1} + (1-\rho_1)(\nabla_{\theta_t}\ell)^2\,,$$
$$M_t = \rho_2 M_{t-1} + (1-\rho_2)(\nabla_{\theta_t}\ell)\,,$$
$$\hat{G}_t = \frac{G_t}{1-\rho_1^t}\,,$$
$$\hat{M}_t = \frac{M_t}{1-\rho_2^t}\,,$$
$$\theta_t = \theta_{t-1} - \frac{\eta\hat{M}_t}{\sqrt{\hat{G}_t + \epsilon}}\nabla_{\theta_t}\ell\,.$$

2.2 Deep unsupervised learning

In this section, we will review the commonly used models for unsupervised learning and also study their limitations.

2.2.1 Restricted Boltzmann Machine (RBM)

An RBM (Hinton, 2002) is a complete bipartite Markov random field with a layer of visible units ($\mathbf{x}$) and another layer of finitely many latent units ($\mathbf{z}$). The visible units correspond to the features of the observed sample, for instance, pixels in an image. Every visible unit is connected to every latent unit by an edge. Since the graph is bipartite, the cliques of the model correspond to the edges and have size 2. The potential function of an edge (x_i, z_j) is given by $-(w_{ij}x_iz_j + a_ix_i + b_jz_j)$, where $\theta = \{w_{ij}, a_i, b_j,\ 1 \leq i \leq d,\ 1 \leq j \leq m\}$ form the parameters of the model. The energy function which is the sum of potential function across all edges, is given by

$$\begin{aligned} E_\theta(\mathbf{x}, \mathbf{z}) &= -\sum_{i,j} w_{ij}x_iz_j + a_ix_i + b_jz_j \\ &= -(\mathbf{x}^T W \mathbf{z} + a^T\mathbf{x} + b^T\mathbf{z})\,. \end{aligned}$$

The corresponding marginal probability of an instance $\mathbf{x}$ is given by

$$p_\theta(\mathbf{x}) = \frac{\sum_{\mathbf{z}} \exp(-E_\theta(\mathbf{x}, \mathbf{z}))}{\sum_{\mathbf{x}', \mathbf{z}} \exp(-E_\theta(\mathbf{x}', \mathbf{z}))}\,. \tag{2.1}$$

In order to maximize the log-likelihood of an RBM for a sequence of observations, one can use stochastic gradient descent. The gradient of the log-likelihood for a fixed observation $\mathbf{x}$ with respect to the weight matrix W is given by

$$\nabla_W \mathcal{L} = \frac{1}{\#\text{training}} \sum_{\mathbf{x} \in \text{training}} \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})}\mathbf{x}\mathbf{z}^T - \mathbb{E}_{p_\theta(\mathbf{x},\mathbf{z})}\mathbf{x}\mathbf{z}^T\,. \tag{2.2}$$

The gradient with respect to other parameters can be computed similarly. The first quantity

in the RHS is straightforward to compute from the following equations.

$$\mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})} z_j = \frac{\sum_{\mathbf{z}} \exp(\mathbf{x}^T W \mathbf{z} + a^T \mathbf{x} + b^T \mathbf{z}) z_j}{\sum_{\mathbf{z}} \exp(\mathbf{x}^T W \mathbf{z} + a^T \mathbf{x} + b^T \mathbf{z}))} ,$$
$$= \frac{1}{1 + \exp(-(\mathbf{x}^T w_j + b_j))} ,$$

where w_j is the j^{th} column in W. In order to compute the second quantity, we need the expected value of $\mathbf{x}\mathbf{z}^T$ for the current choice of W. This can be obtained by using Gibbs sampling. In practise, a small number (r) of iterations of Gibbs sampling is run to get $\mathbf{x}_r$ and $\mathbf{z}_r$ and plugged in (2.2). This method, also known as contrastive divergence (CD) (Hinton, 2002), has been shown to give a good approximation to the actual gradient. The corresponding update equation is given by

$$W^{t+1} = W^t + \eta(\mathbf{x}_0 \mathbf{z}_0^T - \mathbf{x}_r \mathbf{z}_r^T) , \tag{2.3}$$

where η is the learning rate. For more details about training an RBM, we encourage the reader to refer to (Fischer and Igel, 2012).

A commonly used variant of RBM has rectified linear units rather than stochastic binary units. Training of RBMs with rectified linear units is very similar to that with stochastic binary units (Nair and Hinton, 2010). The update equation for an RBM with rectified linear units is exactly similar to that of an RBM with stochastic binary units except that the latent units are sampled uniformly from the normal distribution with mean $\max(0, W^T \mathbf{x})$ and identity covariance matrix.

The energy function of an RBM can be modified to incorporate real-valued input as follows:

$$E_\theta(\mathbf{x}, \mathbf{z}) = -\left[\sum_{i,j} w_{ij} x_i z_j - \frac{(a_i - x_i)^2}{2\sigma^2} + b_j z_j\right] .$$

This model, referred to as Gaussian-Bernoulli RBM, can be used to learn features from patches of images. In order to extend the model to cover the entire image, one can create a grid of latent features. The latent features at grid location (i, j) must be connected to the visible units within its receptive field. Such a model is referred to as convolutional RBM (Lee et al., 2009). Layers of convolutional RBMs can be interleaved with probabilistic max-pooling to create a convolutional deep belief network (DBN) (Lee et al., 2009). The training of convolutional DBN involves layerwise training of several convolutional RBMs with probabilistic pooling and hence,

is quite expensive. Since these models are not trained end-to-end, convolutional DBNs are now rarely employed for extracting features from general images. These models have largely been replaced by models that can use backpropagation for training.

2.2.2 Autoencoders

An autoencoder is a neural network that compresses the input $\mathbf{x}$ to a latent representation $\mathbf{z}$ and attempts to reconstruct the input back from the latent representation. The chief components of an autoencoder include the encoder E that compresses $\mathbf{x}$ to obtain a latent representation, and a decoder D that reconstructs the input back from the compressed representation. The loss of an autoencoder is computed from the reconstructed representation $\hat{\mathbf{x}}$ and the original input $\mathbf{x}$, and is minimized over the training data. For instance, if the loss is the mean squared loss, the objective can be written as

$$\ell(E, D) = \sum_{\mathbf{x} \in \text{ training}} ||\mathbf{x} - D(E(\mathbf{x}))||^2 .$$

The encoded representation $\mathbf{z}$ must have dimensions smaller than the original input $\mathbf{x}$ to prevent the network from learning the identity transform. The model relies on the following principle: More the data has been compressed, more is the pattern extracted from the data. One can allow the encoded representation to be larger than the input by using a sparsity regularizer on the encoded representation. The idea behind using the sparsity regularizer is as follows: Although the possible patterns present in the data are vast, a single image contains a very small subset of those patterns.

Since autoencoders use backpropagation for training, it is straightforward to extend them for general images by using convolutional layers in the encoder and transposed convolutional layers in the decoder. The model can then be trained end-to-end. However, the universal approximation theorem implies that a sufficiently powerful decoder can reconstruct the input back from the encoded representation, irrespective of the encoding (as long as the codes are distinct). Hence, if we make the decoder powerful enough, there is no guarantee that the learnt representations will be meaningful at all.

To address this issue, layerwise training is used in autoencoders (Bengio et al., 2007), whereby the encoder as well as the decoder have a single layer only. Hence, the encoder is forced to learn a representation, from which the input can be reconstructed by a linear operation followed by a non-linearity. Once the layerwise autoencoder has been trained for the first layer, the encoded representation is then fed back to another autoencoder with simple encoding and decoding layers. Finally, after all the autoencoders have been trained, all the encoding

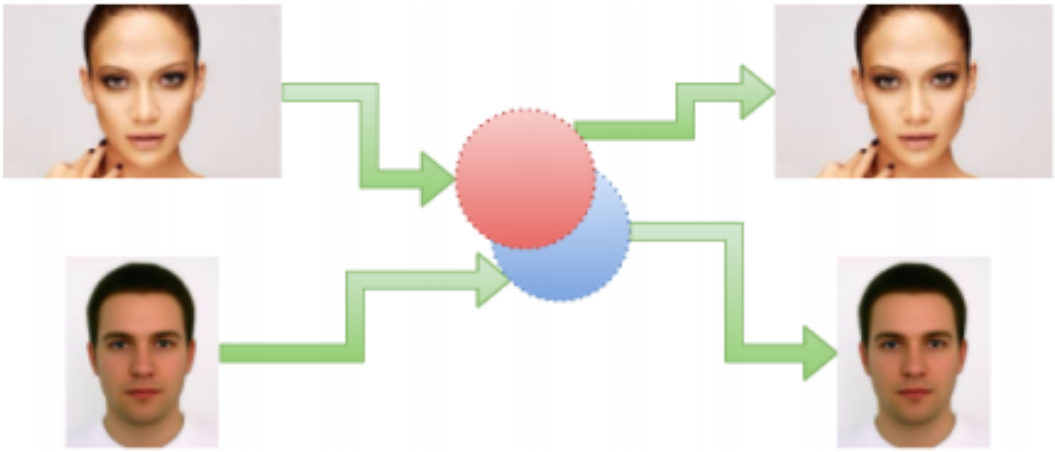


Figure 2.3: Probabilistic embeddings indicated as spheres in the latent space. The overlapping region must correspond to both the input images. Hence, if the probabilistic embeddings of two inputs overlap, the corresponding inputs must be close together in the pixel space.

layers are combined together to form the final encoder, whereas all the decoding layers are combined together to form the final decoder.

Despite their simplicity, autoencoders suffer from a fatal flaw. Two images of the same object under different illumination (due to different camera settings) are very far in the pixel space. In contrast, two images of two different objects under same illumination can be quite close in the input space. Hence, an autoencoder will attempt to push the representation in the first scenario to be far apart, whereas it will attempt to bring the representations in the second scenario to be closer together. Clearly, this is undesirable. Preprocessing the images may improve the quality of the embeddings. However, the presence of a decoder need not help in learning meaningful representations.

2.2.3 Variational autoencoders

While we used the term 'latent representation' for indicating the embeddings learnt by an autoencoder, these embeddings are often deterministic functions of the input. A variational autoencoder (Kingma and Welling, 2013) induces randomness in the representations by adding a noise term to the deterministic embeddings. Hence, the embeddings are probability distributions in the latent space as shown in Figure 2.3.

Assume that the probabilistic embeddings of two distinct inputs overlap to a great extent. This implies that the reconstruction from the overlapping region must correspond to both the

inputs. This is only possible if the two inputs are 'similar', where the definition of similarity depends on the loss function used for comparing the input with the reconstructed representation. In case of mean squared error, this implies that the Euclidean distance between the two images must be small in the pixel space.

Due to the probabilistic nature of the embeddings, variational autoencoders need not resort to layerwise pretraining for learning meaningful embeddings. Let $q_\phi(\mathbf{z}|\mathbf{x})$ be the encoding distribution, and $p_\theta(\mathbf{x}|\mathbf{z})$ be the decoding distribution. The negative of the reconstruction term of a variational autoencoder can be written as follows:

$$\mathcal{L}_1(\theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})$$

If we minimize the reconstruction error only, the probabilistic embeddings will be forced to be as far apart from each other as possible, thereby minimizing the overlap between all the embeddings. Hence, a prior constraint is forced upon the embeddings, that forces the embeddings to overlap. In particular, the encoding distribution is forced to be close to a prior distribution $p_\theta(\mathbf{z})$. The resultant objective is given by

$$\mathcal{L}_2(\theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$$

The above quantity can be shown to be a lower bound of the marginal log-likelihood $\log p_\theta(\mathbf{x})$. Hence, maximizing the above quantity guarantees that the marginal log-likelihood $\log p_\theta(\mathbf{x})$ will atleast be greater than the above quantity. Note that variational autoencoders suffer from the same problem as discussed in the last paragraph for autoencoders.

2.2.4 Generative adversarial networks

The problem with autoencoders and variational autoencoders is that they use the distances computed in pixel-space for learning the representations. As has been mentioned in previous paragraphs, distances in pixel-space are often not very meaningful. Hence, one may wonder if there is a better space for comparing the input and reconstructed image, and how can one identify such a space.

The distinctive feature of generative adversarial networks (Goodfellow et al., 2014) is that they allow the space for comparing images to be learnable. In particular, they train a discriminator to distinguish between real and generated images. Simultaneously, the generator (or decoder) is trained to generate images that are indistinguishable from real images for the classifier. The training of the generator alternates with the training of the discriminator. This creates an adversarial game for the generator and the discriminator as shown in Figure 2.4.

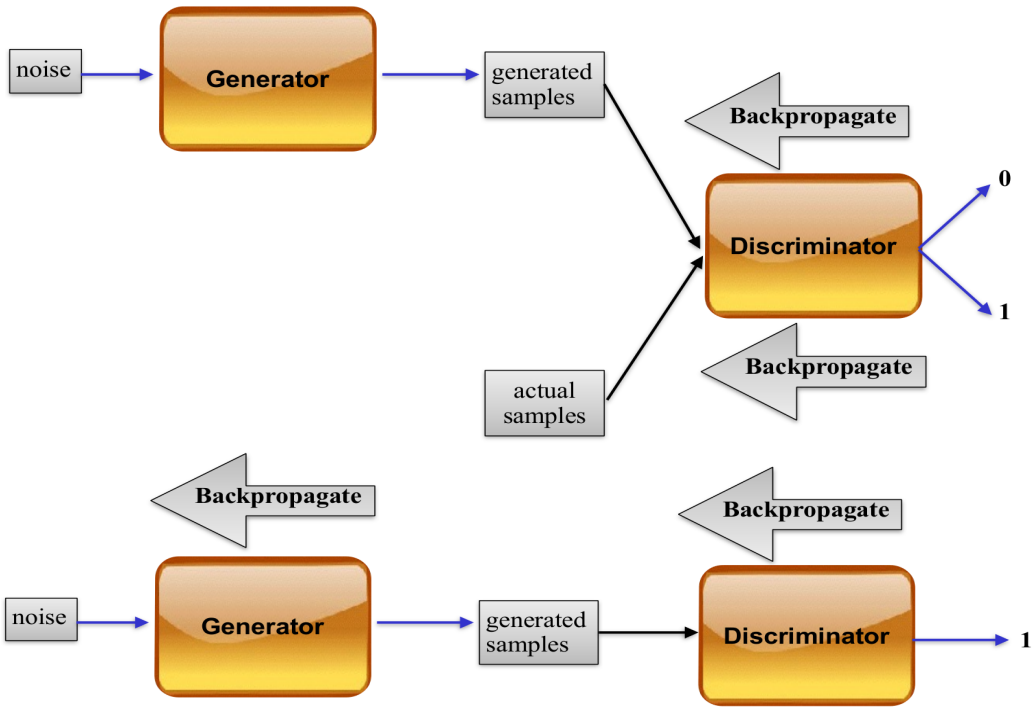


Figure 2.4: The steps involved in training a discriminator (above) and a generator (below) in a generative adversarial network.

It can be shown that when the discriminator is sufficiently powerful, and the training of the discriminator proceeds until the global optima is reached in every iteration, the resultant adversarial game optimizes the Jensen-Shannon divergence between the output distribution of the generator and the distribution of the real images.

The original definition of generative adversarial networks does not employ an encoder. Hence, given an input, it is not possible to obtain the encoded representation of the input. However, extensions of the model that incorporate an encoder have been proposed in literature (Donahue et al., 2016, Dumoulin et al., 2016). In our subjective opinion, these models provide the best unsupervised approach for learning task-agnostic embeddings for data. However, these models suffer from the limitation that they require the input to be real-valued.

Chapter 3

Stretched Deep Networks for Learning with Limited Data

3.1 Introduction

Traditional models for machine learning rely on a single layer of feature extraction, which is followed by a classification layer. The feature extraction can either be explicit, as in the case of SIFT (Lowe, 2004) and HOG (Dalal and Triggs, 2005) or implicit as in the case of kernel machines (Hofmann et al., 2008). These models, which are often referred to as shallow models, have largely been outperformed by deep neural networks that extract multiple layers of features from the data. This is particularly true in computer vision, where very deep neural networks currently hold the state of the art for most supervised learning tasks.

However, deep neural networks often have millions of parameters and require a large amount of labeled data to achieve reasonable performance. The lack of sufficient labeled data can cause deep neural networks to overfit, resulting in poor generalization. In fact, commonly employed neural networks can easily overfit a random labeling of the training data using stochastic gradient, as shown in (Zhang et al., 2017).

Hence, in this chapter, we propose a model that extracts multiple layers of features from the data without the introduction of extra trainable parameters. The model allows for extraction of an unbounded number of features from the data using the 'kernel trick'. Hence, we refer to the model as stretched neural network. The kernel can be computed exactly for single layer stretched networks, and can be approximated using an iterative algorithm for deep convolutional networks. We show that the model achieves reasonable performance for several simple classification tasks. Furthermore, for realistic images of medium size, the proposed model outperforms fully supervised neural networks with similar architectures, when the amount of

labeled data is limited.

Most of the results in this chapter also appear in our paper (Pandey and Dukkipati, 2014a).

3.2 Single layer threshold networks and corresponding kernels

In neural networks, one can sample the weights randomly from a fixed distribution and feed the output of the hidden units directly to a linear classifier such as SVM. Contrary to intuition, it has been observed that when the weights have been sampled from standard normal distribution and the number of hidden units is much greater than the number of visible units, the resultant classifier gives good performance on many classification tasks (Rahimi and Recht, 2009). Furthermore, the performance improves as the number of hidden units increase.

It is possible to perform learning tractably, when the number of hidden units in a randomly weighted neural networks tend to ∞ by using the kernel trick. Cho and Saul (2009) showed that for threshold neural networks, inner products of the randomly sampled hidden units for two instances becomes deterministic as the number of hidden units tend to ∞. The corresponding kernels are termed as arc-cosine kernels. Hence, learning a linear classifier in the original infinite dimensional space is same as learning a kernel machine using the arc-cosine kernel.

In particular, when the hidden units are given by

$$h(\mathbf{x}) = H(w^T\mathbf{x})(w^T\mathbf{x})^n, \qquad w \sim \mathcal{N}(0,1)\,,$$

where H is the Heavyside step function, the corresponding kernel is given by

$$K_n(\mathbf{x},\mathbf{y}) = \frac{1}{2\pi}\|\mathbf{x}\|^n\|\mathbf{y}\|^n J_n(\theta)\ , \tag{3.1}$$

where θ is the angle between $\mathbf{x}$ and $\mathbf{y}$, and $J_n\theta$ is given by

$$J_n(\theta) = (-1)^n(\sin\theta)^{2n+1}\left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\right)^n\left(\frac{\pi-\theta}{\sin\theta}\right),\ \ n\in\mathbb{N}\,.$$

As a special case, when $n = 1$, the hidden units are termed as rectified linear units and the corresponding kernel function is given by

$$K_1(\mathbf{x},\mathbf{y}) = \frac{1}{2\pi}\|\mathbf{x}\|\|\mathbf{y}\|(\sin\theta + (\pi-\theta)\cos\theta)\ . \tag{3.2}$$

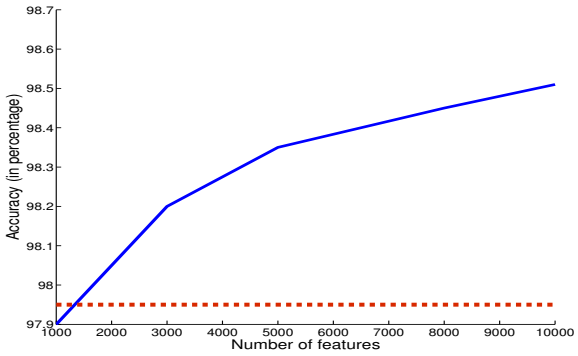


Figure 3.1: The plot depicts the improvement in classification performance for MNIST data, as the weight matrix learnt by an RBM for the dataset is stretched to increase the number of features. The dotted red line indicates the classification performance achieved by an RBM without stretching.

3.3 Single layer stretched networks

A single layer stretched architecture comprises of an input layer, a 'stretched feature layer' (which is defined below) and an output layer. To be precise, let $\mathbf{x}$ be an instance and A be a matrix whose columns indicate the weight vectors learnt by a feature learning algorithm. Let $W \in \mathbb{R}^{K \times L}$ be a random matrix whose entries have been sampled from the standard normal distribution. The encoding of a node in the stretched feature layer is then given by

$$h_j = \frac{1}{\sqrt{L}} f(\mathbf{x}, Aw_j) \, , 1 \leq j \leq L \ , \tag{3.3}$$

where f is a non-linear activation function. As is obvious from the above expression, we have just replaced the original weight vectors by their linear combinations.

In order to evaluate the effect of stretching on the performance of the subsequent classifier, we compare the unstretched model with the stretched model for ReLU RBM for increasing values of L on MNIST dataset. The classifier used at the output layer is a linear SVM. No fine-tuning is performed for either of the models. In order to account for the randomness introduced by the matrix W, we average the results over 10 iterations. The weight matrix used for the stretched and the unstretched model is the same and is obtained after training the model for 5 epochs ($\sim 150s$). The results are plotted in Figure 3.1.

It can be observed from the figure that stretching affects the performance of the subsequent classifier. Furthermore, the performance of the classifier improves as the number of units in the stretched layer increase.

Hence, it is particularly interesting to study the case when the number of weight vectors in the stretched matrix A_{cst}, tend to infinity. This means that A_{cst} consists of infinitely many linear combinations of the original weight vectors, where the scalars for these linear combinations are sampled independently from the standard normal distribution. In matrix notation, one can rewrite the above statement as $A_{cst} = A \times W$, where $A \in \mathbb{R}^{D \times M}$ is the original weight matrix, and $W \in \mathbb{R}^{\mathbb{M} \times \infty}$ is the random matrix whose entries have been sampled independently from the standard normal distribution.

Learning can be made feasible for infinitely stretched networks by using the kernel trick for the special case, when the activation function is of the form $f(\mathbf{x}, w) = (\mathbf{x}^{\mathrm{T}} w)_+$, where

$$x_+ = \begin{cases} x, \text{ if } x > 0 \\ 0, \text{ otherwise} \end{cases} . \tag{3.4}$$

In particular, we note that learning a linear classifier is equivalent to learning a kernel machine with the linear kernel. Here, the linear kernel corresponds to an inner product in the feature space, where the feature representation of an instance $\mathbf{x}$ is obtained by using (3.3). The next theorem allows us to compute the inner product in the feature space as $L \to +\infty$.

Proposition 1 *Let A be fixed weight matrix. Let $W \in \mathbb{R}^{K \times L}$ be a random matrix whose entries have been sampled independently from standard normal distribution. Let $\mathbf{x}$ and $\mathbf{y}$ be two instances whose feature representations are given by*

$$\begin{aligned} h_\ell(\mathbf{x}) =& \frac{1}{\sqrt{L}} (\mathbf{x}^{\mathrm{T}} A w_\ell)_+, \;\; 1 \leq \ell \leq L \;\; \textit{and} \\ h_\ell(\mathbf{y}) =& \frac{1}{\sqrt{L}} (\mathbf{y}^{\mathrm{T}} A w_\ell)_+, \;\; 1 \leq \ell \leq L \, . \end{aligned}$$

Let $\mathbf{h_x} = (h_\ell(\mathbf{x}))^{\mathrm{T}}_{1 \leq \ell \leq L}$ and $\mathbf{h_y} = (h_\ell(\mathbf{x}))^{\mathrm{T}}_{1 \leq \ell \leq L}$. Then the inner product between the feature representation of $\mathbf{x}$ and $\mathbf{y}$ as $L \to \infty$ is given by

$$\lim_{L \to \infty} \mathbf{h}_{\mathbf{x}}^{\mathrm{T}} \mathbf{h}_{\mathbf{y}} = \frac{1}{2\pi} \|A^{\mathrm{T}} \mathbf{x}\| \|A^{\mathrm{T}} \mathbf{y}\| (\sin \theta_A + (\pi - \theta_A) \cos \theta_A) \;\; , \tag{3.5}$$

where $\theta_A = \cos^{-1} \frac{\mathbf{x}^{\mathrm{T}} A A^{\mathrm{T}} \mathbf{y}}{\|A^{\mathrm{T}} \mathbf{x}\| \|A^{\mathrm{T}} \mathbf{y}\|}$.

Proof: When L is finite, the inner product between the feature representation is given by

$$\mathbf{h}_{\mathbf{x}}^{\mathrm{T}} \mathbf{h}_{\mathbf{y}} = \frac{1}{L} (W^{\mathrm{T}} A^{\mathrm{T}} \mathbf{x})_+^{\mathrm{T}} (W^{\mathrm{T}} A^{\mathrm{T}} \mathbf{y})_+$$

$$= \frac{1}{L}(\mathbf{x}^{\mathrm{T}}AW)_+(W^{\mathrm{T}}A^{\mathrm{T}}\mathbf{y})_+$$
$$= \frac{1}{L}\sum_{l=1}^{L}(\mathbf{x}^{\mathrm{T}}Aw_l)_+(\mathbf{y}^{\mathrm{T}}Aw_l)_+ \,.$$

Since w_l, $1 \leq l \leq L$ is a random vector, the above quantity is the empirical mean of L random variables, $z_1, ..., z_L$, where $z_l = (\mathbf{x}^{\mathrm{T}}Aw_l)_+(\mathbf{y}^{\mathrm{T}}Aw_l)_+$. Furthermore since w_l's are independent and identically distributed, z_l's being functions of w_l's are also independent and identically distributed. Hence, by law of large numbers, as $L \to +\infty$, the empirical mean of z_l's converges to the true mean, that is,

$$\lim_{L\to\infty} \mathbf{h}_{\mathbf{x}}^{\mathrm{T}}\mathbf{h}_{\mathbf{y}} = \lim_{L\to+\infty} \frac{1}{L}\sum_{l=1}^{L}(\mathbf{x}^{\mathrm{T}}Aw_l)_+(\mathbf{y}^{\mathrm{T}}Aw_l)_+ \tag{3.6}$$
$$= \int_{w\in\mathbb{R}^d} (\mathbf{x}^{\mathrm{T}}Aw)_+(\mathbf{y}^{\mathrm{T}}Aw)_+ p(w)\mathrm{d}w \tag{3.7}$$
$$= \frac{1}{(2\pi)^{\frac{d}{2}}} \int_{w\in\mathbb{R}^d} (\mathbf{x}^{\mathrm{T}}Aw)_+(\mathbf{y}^{\mathrm{T}}Aw)_+ e^{-\frac{\|w\|^2}{2}}\mathrm{d}w \tag{3.8}$$
$$= \frac{1}{2\pi}\|A^{\mathrm{T}}\mathbf{x}\|\|A^{\mathrm{T}}\mathbf{y}\|(\sin\theta_A + (\pi - \theta_A)\cos\theta_A) \,, \tag{3.9}$$

where $\theta_A = \cos^{-1}\frac{\mathbf{x}^{\mathrm{T}}AA^{\mathrm{T}}\mathbf{y}}{\|A^{\mathrm{T}}\mathbf{x}\|\|A^{\mathrm{T}}\mathbf{y}\|}$. Here, the last equation follows from the derivation of arc-cosine kernel by Cho and Saul (2009). □

If we replace $\Sigma = \frac{1}{M}AA^{\mathrm{T}}$ in (3.5), we get

$$k_\Sigma(\mathbf{x}, \mathbf{y}) = \lim_{L\to\infty} \mathbf{h}_{\mathbf{x}}^{\mathrm{T}}\mathbf{h}_{\mathbf{y}} = \frac{M}{2\pi}\|\mathbf{x}\|_\Sigma\|\mathbf{y}\|_\Sigma(\sin\theta_\Sigma + (\pi - \theta_\Sigma)\cos\theta_\Sigma) \,,$$

where $\|\mathbf{x}\|_\Sigma = \sqrt{\mathbf{x}^{\mathrm{T}}\Sigma\mathbf{x}}$ and $\theta_\Sigma = \cos^{-1}\frac{\mathbf{x}^{\mathrm{T}}\Sigma\mathbf{y}}{\sqrt{\mathbf{x}^{\mathrm{T}}\Sigma\mathbf{x}}\sqrt{\mathbf{y}^{\mathrm{T}}\Sigma\mathbf{y}}}$. Hence, we refer to the kernel as covariance arc-cosine kernel (Pandey and Dukkipati, 2014b) and denote it as $k_\Sigma(\mathbf{x}, \mathbf{y})$.

3.4 Experiments on simple datasets

Before we move any further, let's evaluate whether the inclusion of feature matrix in the arc-cosine kernel results in improvement in performance. In particular, we compare the performance of single layer stretched network against other classifiers on the dataset of simple binary images used in (Larochelle et al., 2007). The details of datasets that are used for the experiments, are given in Table 3.1.We normalize the pixel values to lie between 0 and 1. Except that, we do not use any preprocessing for the dataset. A standard Bernoulli RBM with 1000 hidden units

Table 3.1: Datasets used in our evaluation of single layer stretched networks.

	#training	#test	#classes
MNIST original	60,000	10,000	10
MNIST rotated	12,000	50,000	10
Rectangles	1,200	50,000	2
Convex	8,000	50,000	2

is trained on the raw pixel values using contrastive divergence with a fixed momentum of 0.5. In our experiments for MNIST dataset, we found that fixing the bias to zero does not affect the performance of the final model.

We use the weight matrix learnt by RBM to compute the kernel matrix of the stretched network. We use kernel spectral regression with the default hyperparameter settings (Cai et al., 2011) for all our experiments on stretched networks. Kernel spectral regression is a fast algorithm for performing discriminant analysis in the reproducing kernel Hilbert space in which the data points are mapped. The classifier was chosen due to its stability to the choice of hyperparameters. The results are given in Table 3.2. As can be observed from the table, a single layer stretched network outperforms other deeper architectures for the above datasets. Furthermore, the inclusion of a learnt weight matrix indeed results in improvement in performance as compared to arc-cosine kernels.

Table 3.2: Comparison of single layer stretched network against other classifiers using all the labels. Some of the results have been borrowed from (Larochelle et al., 2007), but only the best results are kept. The best result for each dataset is indicated in bold. For arc-cosine kernel, the numbers in parenthesis indicate the depth of the kernel.

	MNIST original	MNIST rotated	Rectangles	Convex
SVM-rbf	1.6%	10.38%	2.15%	19.13%
SVM-poly (Decoste and Schölkopf, 2002)	1.22%	13.61%	2.15%	19.82%
Neural Network (one hidden layer)	1.9%	17.62%	7.16%	32.25%
DBN (Hinton et al., 2006)	1.25%	12.30%	2.60%	18.63%
Stacked Autoencoder (Bengio et al., 2007)	1.4%	11.43%	2.41%	18.41%
Arc-cosine kernel (Cho and Saul, 2009)	1.38%(5)	11.22%(5)	2.27%(15)	17.15%(5)
Stretched Network	**0.95%**	**8.11%**	**1.55%**	**17.02%**

3.5 Incorporating translational invariance in SDNs

As was shown in our preliminary experimental results with single layer, infinitely stretched networks allow us to obtain good performance for object recognition tasks when the images are all centred and have few pose variations, for instance, MNIST data set. However, for complex real world tasks such as scene labelling, the above technique can not be expected to perform equally well, since no prior information about the data (for instance, the information that objects can be present at any location in the model) is incorporated in the model. The usual way to incorporate translational invariance in neural networks is to divide the image into overlapping patches, where each patch is a $d \times d$ square of pixels in an image. The patches are flattened from matrices to vectors and the same weight matrix is then learnt using each flattened patch in either supervised or unsupervised mode. The features are then pooled over a region in the image, where the region may be fixed *a priori* (LeCun et al., 1998), or chosen stochastically (Zeiler and Fergus, 2013). The above process is then repeated by treating the pooled feature maps from the previous layer as input. Such an architecture is called a convolutional neural network.

3.5.1 Convolutional stretched networks

Assuming that one prefers to use average pooling and a single pooling layer, there is a straightforward method to incorporate translational invariance in infinitely stretched networks. Let $\{x_1, .., x_N\}$ be the flattened patches in a region of an image $\mathbf{x}$, and $\{y_1, ..., y_N\}$ be the flattened patches of the corresponding region in $\mathbf{y}$. Let A be the weight matrix learnt by some feature learning algorithm. If no feature learning algorithm is used, A can be assumed to be an identity matrix. Then, the inner product between the stretched and pooled feature representation of $\mathbf{x}$ and $\mathbf{y}$ for the chosen region is given by

$$\mathbf{h}_{\mathbf{x}}^{\mathrm{T}}\mathbf{h}_{\mathbf{y}} = \frac{1}{LN^2}\left(\sum_{i=1}^{N}(x_i^{\mathrm{T}}AW)_+\right)\left(\sum_{i=1}^{N}(y_i^{\mathrm{T}}AW)_+\right)^{\mathrm{T}} .$$

As $L \to \infty$, the above inner product can be rewritten as

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) = \lim_{L\to\infty} \mathbf{h}_{\mathbf{x}}^{\mathrm{T}}\mathbf{h}_{\mathbf{y}} &= \lim_{L\to\infty} \frac{1}{LN^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\ell=1}^{L}(x_i^{\mathrm{T}}Aw_\ell)_+(y_j^{\mathrm{T}}Aw_\ell)_+ \\ &= \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}k_\Sigma(x_i, y_j) . \end{aligned}$$

Several important points can be noted about the manner in which the above kernel has been defined. Firstly, the above approach can be applied only if the pooling mechanism used is sum pooling or average pooling. Currently, we are not aware of any other approach to incorporate more general pooling mechanisms exactly in infinitely stretched networks. Secondly, the computation of a single entry of the pooled kernel matrix involves $\mathcal{O}(N^2)$ kernel computations, where N is the number of patches in the region. In general, the number of patches can be as high as the number of pixels in the region. Assuming a region of size 100×100, this means that computation of one entry of the pooled kernel matrix may require 10^8 kernel computations, clearly a prohibitively large number.

However, the main drawback of the above architecture is not its practical infeasibility, but the inherent limitation of the model itself. In a general convolutional architecture (LeCun et al., 1998), the first pooling layer pools the features over a smaller region, with successive layers pooling over larger and larger regions. The possible exception is the architecture in (Coates et al., 2011), where the features are pooled over a large region in a single pooling layer. However, in the approach described above, once we obtain the kernel matrix after first pooling layer, all information about location of pixels in the original image is lost. Hence, there is no way to apply a second pooling layer to the model. Hence, we are limited to using single pooling layer only.

3.5.2 Approximating convolutional SDNs

Because of the limitations of using pooling in infinitely stretched networks, we need to resort to an approximation of the model. In particular, we restrict our network to have finitely many filters, say L. Let A be fixed weight matrix learnt by a feature learning algorithm from the patches. Let W be a random weight matrix whose entries have been sampled from a fixed distribution. We compute the convolved and pooled feature representation for each image as given below:

$$\mathbf{h_x} = \frac{1}{N\sqrt{L}} \sum_{i=1}^{N} ((WA)^{\mathrm{T}} x_i)_+ \ ,$$

where x_i are the patches in $\mathbf{x}$. These representations are then used to update the kernel. This entire process is repeated until the difference between kernel entries in successive iterations reaches below a threshold. This approach is described in detail in Algorithm 2.

This architecture addresses the first two issues associated with a convolutional SDN mentioned in Section 3.5.1. The issue of multiple pooling layers is addressed in Section 3.5.3. In order to convey the most important points of the algorithm, we have assumed that each image has a single pooling region in the first pooling layer. It is quite straightforward to extend the

model for the case where each image has multiple pooling regions in the first pooling layer.

Algorithm 1: Iterative computation of the convolved kernel matrix

1 **Input:** A set of training instances
2 **Output:** The pooled kernel matrix

- Initialize the kernel matrix k_{ap} to all zeros.
- Learn the weight matrix A from patches of images using an unsupervised feature learning algorithm.
- Repeat the following steps till convergence
  1. Sample the entries of the random matrix $W \in \mathbb{R}^{D\times L}$ independently from standard normal distribution
  2. Compute the pooled feature representation of each instance using the random matrix W. In particular, the pooled feature representation of $\mathbf{x}$ is given by

$$\mathbf{h}_{\mathbf{x}} = \frac{1}{N\sqrt{L}} \sum_{i=1}^{N} ((WA)^{\mathrm{T}} x_i)_+ \ ,$$

     where x_i are the patches in $\mathbf{x}$.
  3. Update the approximate kernel matrix by using the following equation

$$k_{ap}^{(t+1)}(\mathbf{x}, \mathbf{y}) = \frac{1}{t+1}(t \times k_{ap}^{(t)}(\mathbf{x}, \mathbf{y}) + \mathbf{h}_{\mathbf{x}}^{\mathrm{T}} \mathbf{h}_{\mathbf{y}}) \,. \tag{3.10}$$

- Return the kernel matrix k_{ap}.

The proposed algorithm gives an iterative, memory-efficient method of computing the pooled kernel matrix. We show below that if the algorithm is run long enough, the resulting pooled kernel matrix indeed converges to the true pooled kernel matrix, that we obtained in the previous section. Here, we have shown the result for average pooling. For max-pooling, we need to use the inequality $\mathrm{var}(\max_{i=1}^{N} X_i) \leq \sum_{i=1}^{N} \mathrm{var}(X_i)$, where $\mathrm{var}(X)$ denotes the variance of the random variable X. Note that, one can not obtain the mean ($k(\mathbf{x}, \mathbf{y})$ in equation (3.11)) in closed forms for max-pooling. However, the result will still remain valid.

Proposition 2 *The pooled kernel matrix of approximately stretched network converges in probability to the pooled kernel matrix of infinitely stretched network. Furthermore, let T be the number of iterations for which Algorithm 2 has been run. Then, for fixed instances $\mathbf{x}$ and $\mathbf{y}$,*

with probability at least 1-δ,

$$|k_{ap}(\mathbf{x},\mathbf{y}) - k(\mathbf{x},\mathbf{y})| \leq \sqrt{\frac{2}{TL\delta}} \sqrt{\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} \|A^{\mathrm{T}}x_i\|^2 \|A^{\mathrm{T}}y_j\|^2} \ , \tag{3.11}$$

where $k(\mathbf{x},\mathbf{y})$ is the exact value of the kernel entry and $k_{ap}(\mathbf{x},\mathbf{y})$ is the approximate kernel value after T iterations.

Proof: After T iteration of Algorithm 2, the approximate kernel matrix $k_{ap}(\mathbf{x},\mathbf{y})$ is given by

$$\begin{aligned} k_{ap}(\mathbf{x},\mathbf{y}) &= \frac{1}{T}\sum_{t=1}^{T}\frac{1}{LN^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\ell=1}^{L}(x_i^{\mathrm{T}}Aw_{t\ell})_+(y_j^{\mathrm{T}}Aw_{t\ell})_+ \\ &= \frac{1}{TLN^2}\sum_{t,l}\sum_{i,j}(x_i^{\mathrm{T}}Aw_{t\ell})_+(y_j^{\mathrm{T}}Aw_{t\ell})_+ \ . \end{aligned}$$

Furthermore, the mean and variance of the approximate kernel after T iterations is given by

$$\begin{aligned} m(\mathbf{x},\mathbf{y}) &= \frac{1}{TLN^2}\sum_{t,l}\sum_{i,j}\mathbb{E}[(x_i^{\mathrm{T}}Aw_{t\ell})_+(y_j^{\mathrm{T}}Aw_{t\ell})_+] \\ &= \frac{1}{TLN^2}\sum_{t,l}\sum_{i,j}k_{\Sigma}(x_i,y_j) \\ &= \frac{1}{N^2}\sum_{i,j}k_{\Sigma}(x_i,y_j) = k(x,y). \\ v(\mathbf{x},\mathbf{y}) &= \frac{1}{T^2L^2}\sum_{t,l} var\left(\frac{1}{N^2}\sum_{i,j}(x_i^{\mathrm{T}}Aw_{t\ell})_+(y_j^{\mathrm{T}}Aw_{t\ell})_+\right) \\ &\leq \frac{1}{T^2L^2N^2}\sum_{t,l}\sum_{i,j} var((x_i^{\mathrm{T}}Aw_{t\ell})_+(y_j^{\mathrm{T}}Aw_{t\ell})_+) \\ &= \frac{1}{TLN^2}\sum_{i,j} var((x_i^{\mathrm{T}}Aw)_+(y_j^{\mathrm{T}}Aw)_+)\,. \end{aligned}$$

Since $var(X) = \mathbb{E}X^2 - (\mathbb{E}X)^2 \leq \mathbb{E}X^2$, $var((x_i{}^{\mathrm{T}}Aw)_+(y_j{}^{\mathrm{T}}Aw)_+) \leq \mathbb{E}[(x_i^{\mathrm{T}}Aw)_+^2(y_j^{\mathrm{T}}Aw)_+^2]$. From (3.1), the same quantity can be rewritten as

$$\begin{aligned} \mathbb{E}[(x_i^{\mathrm{T}}Aw)_+^2(y_j^{\mathrm{T}}Aw)_+^2] =& \|A^{\mathrm{T}}x_i\|^2\|A^{\mathrm{T}}y_j\|^2\frac{(3\sin\theta\cos\theta + (\pi-\theta)(1+2\cos^2\theta))}{2\pi} \\ &\leq 2\|A^{\mathrm{T}}x_i\|^2\|A^{\mathrm{T}}y_j\|^2 \ . \end{aligned}$$

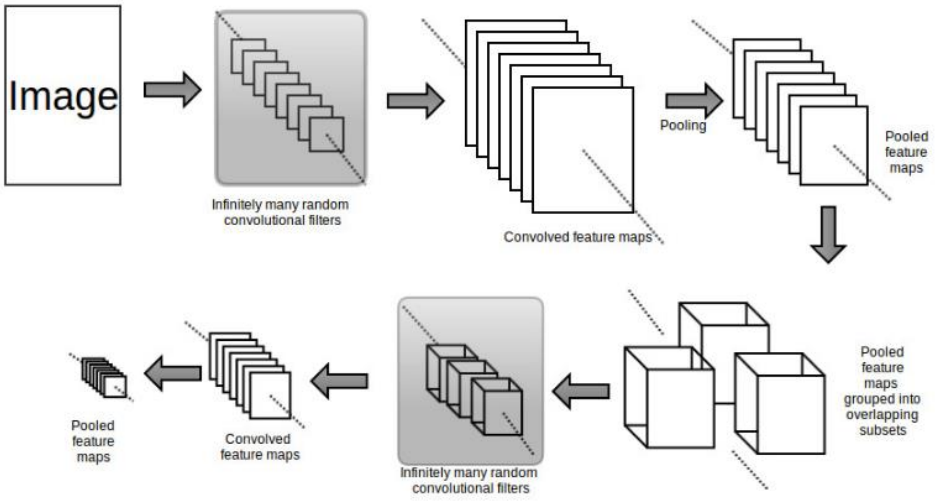


Figure 3.2: A convolutional SDN with two pooling layers.

Hence, by applying Chebyshev's inequality to the quantity $k_{ap}(\mathbf{x}, \mathbf{y})$, we get

$$P(|k_{ap}(\mathbf{x}, \mathbf{y}) - k(\mathbf{x}, \mathbf{y})| \geq \epsilon) \leq 2\frac{1}{TLN^2\epsilon^2} \sum_{i=1}^{N}\sum_{j=1}^{N} \|A^{\mathrm{T}}x_i\|^2\|A^{\mathrm{T}}y_j\|^2 \,.$$

Hence, as $T \to \infty$, approximate kernel entry converges in probability to the exact kernel entry. Rewriting the above statement, we can say that with probability at least $1 - \delta$.

$$|k_{ap}(\mathbf{x}, \mathbf{y}) - k(\mathbf{x}, \mathbf{y})| \leq \sqrt{\frac{2}{TL\delta}}\sqrt{\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} \|A^{\mathrm{T}}x_i\|^2\|A^{\mathrm{T}}y_j\|^2} \,.$$

□

The above result implies that, if the difference between a kernel entry in approximate and exact kernel matrix after 1 iteration is ϵ with probability at least $1 - \delta$, one can say that with the same probability, we obtain a 10 fold reduction in estimation error by scaling L ten times and repeating the experiment for ten iterations.

By using the above approximation, one can incorporate any non-linearity in the model architecture. One disadvantage of the model is that one may need to run the algorithm multiple number of times to obtain the pooled kernel matrix. However, since the entries of the random matrix W are sampled independently, it is trivial to parallelize the model on multiple nodes or adapt it to any distributed system. Furthermore, in our experiments, we found that 20 iterations were often enough to obtain good results for many object recognition tasks.

3.5.3 Extension to multiple layers

Since the computation of the exact kernel for an SDN with a single convolution and pooling layer is very expensive, we restrict ourselved to the iterative algorithm for computing the kernel by randomly sampling the weights of various layers per iteration. We also attempted to learn the weights that can best approximate the kernel for each layer by minimizing

$$\left| k_\Sigma(x_i, y_j) - \frac{1}{L} \sum_{\ell=1}^{L} (x_i^{\mathrm{T}} w_\ell)_+ (y_j^{\mathrm{T}} w_\ell)_+ \right|^2$$

over $w_1, \ldots, w_L$ for randomly selected pairs of patches. Here, k_Σ is the covariance arc-cosine kernel. However, to our surprise, the network obtained using this approach performs no better than a network whose weights have been sampled from the normal distribution. Hence, in the rest of section, we limit ourselves to convolutional SDNs with weights sampled from the normal distribution.

Instead of using the output of the first pooling layer to compute the kernel matrix, one can feed it to another layer of convolution + pooling. Specifically, let $\mathbf{x}$ be an image of size $m \times m$ and U be a tensor of size $d \times d \times L_1$ whose entries are sampled from $\mathcal{N}(0, 1)$. Let $u_1, ..., u_{L_1} \in \mathbb{R}^{d \times d}$ be the sub-tensors of U obtained by fixing the last dimension of U. For the sake of clarity, we assume that the matrix A learnt from flattened patches of $\mathbf{x}$, is an identity matrix. It is straightforward to merge the matrix A with tensor U, when it is not an identity matrix.

In the first convolutional layer, $\mathbf{x}$ is convolved with $u_1, ..., u_{L_1}$ (followed by rectification) to obtain L_1 feature maps of size $(m-d+1) \times (m-d+1)$, which are then pooled using any of the possible pooling methods. Let the resultant size of each feature map be $m_1 \times m_1$. Let us denote the feature maps at the output of the first pooling layer as $f_1, ..., f_{L_1}$. Hence, $f_i = \mathrm{pool}(x * u_i)$, where * denotes the 2D convolution operator and pool(.) denotes the pooling operation.

Next, the L_1 feature maps at the output of the first pooling layer are divided into L_2 overlapping subsets, where each subset consists of Q feature maps. One can represent each subset as a tensor of size $m_1 \times m_1 \times Q$. Let us denote them as $\hat{x}_1, ..., \hat{x}_{L_2}$. Let $V \in \mathbb{R}^{d \times d \times Q \times L_2}$ be a tensor whose entries have been sampled from $\mathcal{N}(0, 1)$. Let $v_1, ..., v_{L_2} \in \mathbb{R}^{d \times d \times Q}$ denote the sub-tensors of V obtained by fixing the last dimension. Each subset $\hat{x}_1, ..., \hat{x}_{L_2}$ is convolved with its corresponding sub-tensor to obtain L_2 feature maps, which are then pooled. We denote the resultant feature maps as $g_1, ...g_{L_2}$, where $g_i = \mathrm{pool}\,(\hat{x}_i * v_i)$.

One can add further layers in a similar manner. The output of the last pooling layer is then fed to a linear kernel. This entire procedure corresponds to one single iteration of kernel

computation. A pictorial representation of SDN is given in Figure 3.2. The entire algorithm for computing the kernel is informally described in Algorithm 2.

Algorithm 2: Iterative computation of the convolved kernel matrix for multiple stages of convolution + pooling

Input: A set of training instances

Output: The pooled kernel matrix

- Initialize the kernel matrix k_{ap} to be all zeros.
- Repeat until the kernel matrix k_{ap} converges
  - Sample the entries of tensors U, V etc., from $\mathcal{N}(0,1)$.
  - Use the steps described in Section 3.5.3 to compute the feature maps at the output of the last pooling layer for each instance. Let they be denoted as $g_1, ...g_{L_2}$. Flatten it to get a single vector $\mathbf{g}$.
  - Update the kernel matrix as

$$k_{ap}^{(t+1)}(\mathbf{x},\mathbf{y}) = \frac{1}{t+1}(t \times k_{ap}^{(t)}(\mathbf{x},\mathbf{y}) + \mathbf{g}_{\mathbf{x}}^{\mathrm{T}}\mathbf{g}_{\mathbf{y}}) \,.$$

- Return the kernel matrix k_{ap}.

Finally, we would like to mention that in all our experiments using convolution SDNs, we assume an infinite fully-connected random layer as the final layer, while computing the final kernel matrix. Let k_{ap} be the kernel matrix obtained after multiple stages of convolution + pooling and k_{full} be the kernel matrix after the fully connected random layer. Furthermore, let $\mathbf{g_x}$ and $\mathbf{g_y}$ be the feature representation of two instances $\mathbf{x}$ and $\mathbf{y}$ after the the last pooling stage. Then, we have

$$\begin{aligned} k_{full}(\mathbf{x},\mathbf{y}) &= \frac{1}{2\pi}\|\mathbf{g_x}\|\|\mathbf{g_y}\|(\sin\theta + (\pi-\theta)\cos\theta) \\ &= \frac{1}{2\pi}\sqrt{k_{ap}(\mathbf{x},\mathbf{x})}\sqrt{k_{ap}(\mathbf{y},\mathbf{y})}(\sin\theta + (\pi-\theta)\cos\theta) \ , \end{aligned}$$

where θ is the angle between $\mathbf{g_x}$ and $\mathbf{g_y}$ and is given by

$$\theta = \cos^{-1}\left(\frac{k_{ap}(\mathbf{x},\mathbf{y})}{\sqrt{k_{ap}(\mathbf{x},\mathbf{x})}\sqrt{k_{ap}(\mathbf{y},\mathbf{y})}}\right) \ .$$

Hence, the kernel matrix after the fully connected stage only depends upon the kernel matrix

Table 3.3: Classification accuracies on Caltech-101 dataset using various algorithms. The numbers in parenthesis indicate the number of hidden units in each layer.

Algorithm	Accuracy
Conv. SDN-RBM (64-256-512)	74.1%
Conv. SDN-RBM (64-256)	**74.3**%
Conv. SDN-RBM (64)	66.2%
Conv. DBN (64-256) (not finetuned)	64.1%
Conv. DBN (64-256) (finetuned)	65.5%

after the last pooling stage and hence, it can be computed after Algorithm 2 stops.

Note: The law of large numbers implies that the approximate kernel entry derived in this section will converge to its mean as long as the mean is finite (since the kernel entry is non-negative). However, unlike the case of SDNs with single convolutional layer, the mean of this multi-layer approximate kernel will not be the same as the mean of a deep convolutional SDN with infinitely many weights in each layer.

3.6 Experimental results for convolutional SDNs

In this section, we present the results obtained by using convolutional SDNs for image recognition tasks. Whenever we refer to convolutional SDNs, we mean the approximated convolutional SDN derived using Algorithm 2. As before, we use kernel spectral regression with the default parameter settings (Cai et al., 2011) for all our experiments on stretched networks. We focus on 2 datasets with limited labeled data, that is, Caltech-101 and STL-10. Furthermore, to show that the performance of the model indeed depends on the amount of labeled data, we also perform experiments for varying amount of labeled data for NORB and CIFAR-10 dataset.

Caltech-101 dataset: The Caltech-101 dataset (Fei-Fei et al., 2007) consists of pictures of objects belonging to 101 categories with about 40 to 800 objects per category. For each object class, we used 30 samples for training and a maximum of 30 samples for testing. We use the same preprocessing as done in (Jarrett et al., 2009). Furthermore, we average the result over 5 draws from the training set.

Effectively, we use a similar architecture as used by Jarrett et al. (2009). We randomly extract 9×9 patches from the images and learn a weight matrix with 64 weight vectors from these patches using a ReLU RBM. The weight matrix is then stretched by multiplication with a random matrix of size 64×64. We use average pooling with a 10×10 boxcar filter and 5×5 down-sampling. The output of first stage of feature extraction stage is 64 feature maps of size 26×26. In the second stage, we randomly combine 30 feature maps from the previous layers

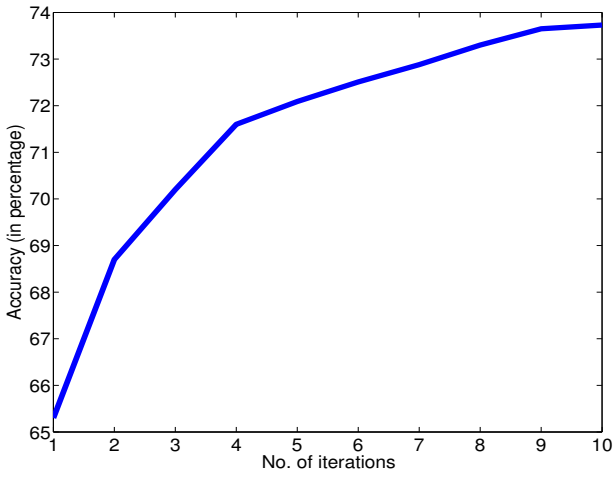


Figure 3.3: The plot depicts the improvement in classification performance for Caltech-101 dataset with respect to the no. of iterations of stretching.

using 256 9×9 kernels . Again, we use average pooling in this layer with a 6×6 boxcar filter with a 4×4 down-sampling step. The output of the second layer of feature extraction is 256 feature maps of size 4×4. In both stages, we use the $\max(0, x)$ non-linearity and local contrast normalization. In the third stage, we map the output of the second stage (256 feature maps of size 4×4) to infinitely many feature maps using an arc-cosine kernel. We repeat the entire process for 20 iterations to construct the final kernel matrix as discussed in Algorithm 2.

By iterating over the above architecture for 20 iterations, our best model achieves an accuracy of 74.3%. We plot the change in classification performance with the no. of iterations for Caltech-101 dataset in Figure 3.3 for the first 10 iterations. The result should be compared with accuracy achieved by an unstretched architecture of the same size, which is 64.1%. The accuracy achieved by finetuning the same architecture is 65.5% as mentioned in (Jarrett et al., 2009).

We also experimented using a deeper SDN of size $64 - 256 - 512$. However, as can be observed from Tabel 3.3, the resultant model did not achieve any improvement in performance. We attempted to use AlexNet as the base architecture for our convolutional SDN. However, the resultant performance was considerably worse than the performance achieved by our best model. Hence, it appears that the performance of a convolutional SDN does not improve with depth.

STL-10 dataset: STL-10 dataset (Coates et al., 2011) is an image recognition dataset with 10 classes and 500 training and 800 test images per class. We use the same preprocessing as in the previous case. Again, two layers of convolution, pooling and local contrast normalization

Table 3.4: Classification accuracies on STL-10 dataset using various algorithms.

Algorithm	Accuracy
Conv. SDN-RBM (64-256)	65.7%
Conv. SDN-RBM (64)	55.3%
Conv. DBN (64-256) (not finetuned)	53.9%
Conv. DBN (64-256) (finetuned)	55.3%

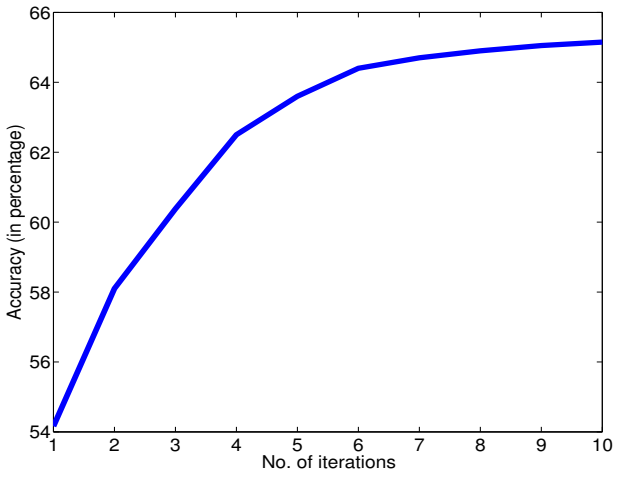


Figure 3.4: The plot depicts the improvement in classification performance for STL-10 dataset with respect to the no. of iterations of stretching.

are used. As in the previous case, we randomly extract 9×9 patches from the images and learn a weight matrix with 64 weight vectors from these patches using an ReLU RBM. The first pooling layer uses an 8×8 boxcar filter with 3×3 average sampling. The second pooling layer uses 5×5 boxcar filter with 4×4 average sampling. The rest of the architecture remains the same as in the previous case. The output of the second stage is mapped to infinitely many feature maps by using the arc-cosine kernel. After iterating over the above architecture for 20 iterations, our best model achieves an accuracy of 65.7%. The change in accuracy of an SDN with the number of iterations is given in Figure 3.4. The result should be compared with the accuracy achieved by an unstretched architecture of the same size, which is 53.9%. The accuracy achieved by finetuning the same architecture is 55.3%.

NORB dataset: The datasets presented in the previous experiments consist of a small number of images per class. Hence, SDNs with minimal supervised training were able to outperform fully supervised CNNs of similar architecture. In contrast, the NORB dataset consists of 5 object categories with 4860 images per class. Hence, NORB dataset (LeCun et al., 2004) provides an excellent testbed for comparing fully supervised CNNs with SDNs by varying the amount of labeled data.

We use exactly the same architecture as used in (Jarrett et al., 2009) for NORB dataset. In particular, the first convolution layer consists of 8 filters of size 5×5, which is followed by 5×5 average pooling. The second convolutional layer has 24 filters of size 6×6. Each filter combines 4 randomly chosen feature maps of the previous layer. Finally, the last pooling layer uses 3×3 average pooling.

We trained a fully supervised CNN and a convolutional SDN with 50,150,500,1500 and 4860 labeled images per class. The results are shown in Figure 3.5. As can be observed from the figure, the convolutional SDN outperforms a fully supervised CNN when the number of training samples are small. However, as the number of training samples increase, fully supervised CNN outperforms convolutional SDN.

CIFAR-10 dataset: The classes of the CIFAR-10 dataset are exactly the same as the classes in STL-10 dataset. However, the number of labeled images per class is much higher (5000). Hence, rather than designing a new architecture, we use exactly the same architecture as used for STL-10 dataset. Since the images in CIFAR-10 are half in width and height as compared to the images in STL-10, we use nearest neighbor upsampling to upsample the images to 64×64.

We trained a fully supervised CNN and a convolutional SDN with labeled images per class. The results are shown in Figure 3.6. As can be observed from the figure, the convolutional SDN outperforms a fully supervised CNN when the number of training samples are small. However, as the number of training samples increase, fully supervised CNN outperforms convolutional

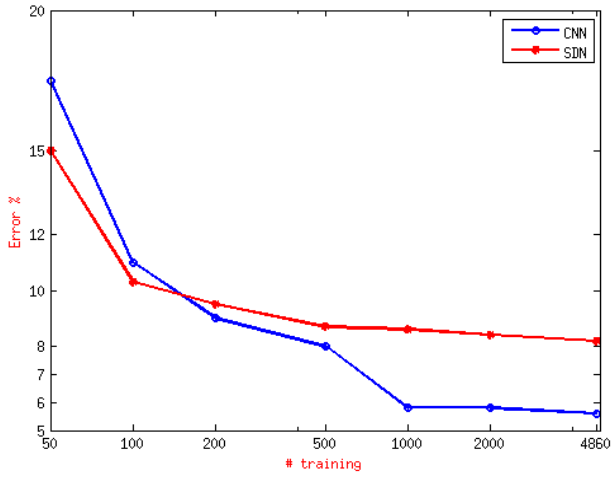


Figure 3.5: The plot compares the performance of an SDN with a fully supervised CNN on NORB dataset. As the number of labeled examples increase, SDNs are outperformed by fully supervised CNNs.

SDN.

3.7 Discussion

In this chapter, we proposed stretched networks that allow for extraction of infinitely many features from objects via the kernel trick. We show that for several simple datasets, single layer stretched networks achieve acceptable performance, often outperforming their fully supervised counterparts. Moreover, for realistic images, when the amount of labeled data is limited, convolutional SDNs outperform convolutional neural networks with similar architecture.

Our model is based on the work done by several authors for studying neural networks of infinite width. In particular, it was shown by Neal (2012), that Bayesian neural networks with infinitely many neurons converge to a Gaussian process prior for several prior distributions on the weights. The covariance functions of the Gaussian process for sigmoidal and rbf activation functions was derived by Williams (1997). Cho and Saul (2009) extended the family of known covariance functions for infinite neural networks to include one-sided polynomial activation functions. The arc-cosine kernel used in this chapter, is a special case when the degree of the polynomial activation function is set to 1.

The work done in this chapter appeared in (Pandey and Dukkipati, 2014a). More recently, kernels that explicitly incorporate translational invariance have been proposed (Mairal, 2016, Mairal et al., 2014) and have been shown to achieve performance comparable with CNNs. Another line of research uses the output of the neural networks to compute the covariance function

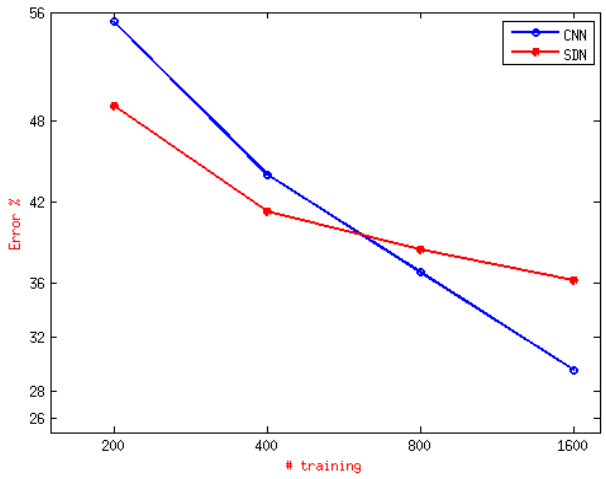


Figure 3.6: The plot compares the performance of SDN with fully supervised CNN on CIFAR-10 dataset. As in the case of NORB dataset, SDN are outperformed by fully supervised CNN with increase in labeled examples.

of a Gaussian process. The derivative of the log-likelihood with respect to the covariance function is then used to train the kernel (Wilson et al., 2016).

Despite the success of the proposed model in handling limited labeled data, SDNs have several limitations as compared to standard neural networks. The primary limitation of SDN is their inability to utilize labeled and unlabeled data to train all the layers of the network. As a consequence, the performance of SDNs does not improve with depth even in the presence of large amount of labeled data. The reason for this behavior is the fact that each layer in an SDN computes a random function of the previous layer. Hence, if the number of random layers in a convolutional SDN (and hence the depth) increases, most of the information present in the original image is lost, by the time it reaches the final layer. To be more specific, the output of a deep convolutional SDN is almost independent of the original image.

Another limitation of SDN is its inability to utilize pretrained networks. For instance, the networks that achieve the best performance on most datasets with limited labeled data are the ones that have been trained on the much larger ImageNet dataset. It is quite common to take a network pretrained on ImageNet, discard the last layer, and finetune it on a smaller dataset with limited number of labeled examples. In contrast, the weights of all the layers of an SDN except for the last layer, are randomly drawn from a fixed distribution. Hence, it makes no sense to transfer the weights from an SDN trained on a larger dataset to another SDN. Furthermore, there is no known mechanism to transfer the weights of a pretrained neural network to an SDN.

As a consequence of the above limitations, SDNs can not achieve the mainstream appeal

that is enjoyed by traditional neural networks. An architecture that allows a hierarchy of learnable layers will often be more useful than architectures with random layers. Hence, in the next chapter, we propose a model that utilized labeled as well as unlabeled data for training all the layers of the network. The new model will address all the concerns of an SDN while still being capable of learning with limited labeled data.

Chapter 4

Discriminative Encoders for Learning with Unlabeled Data

4.1 Introduction

Is it possible to learn meaningful representations using unlabeled data? The representation of interest might be a categorical vector representing the object present in a image, a real-valued vector representing the edges/shapes present in an image, the topics present in a document, the abstract speech patterns in a speech signal etc. Over the years, several models have been proposed to extract patterns from unlabeled images, speech and text.

Generative models address this problem by defining a joint distribution over the visible and the latent features. The marginal log-probability of the observed data is then maximized. Examples of such models include Gaussian mixture models, latent Dirichlet allocation (Blei et al., 2003), restricted Boltzmann machines (Hinton, 2002), variational autoecnoder (Kingma and Welling, 2013, Rezende et al., 2014) etc. However, these models require the specification of the parametric form for the distribution on the input $\mathbf{x}$. Unfortunately, successfully modeling the distribution of images and speech signals is incredibly challenging.

In this chapter, we propose a model for learning latent representations from unlabeled data that does not model the distribution of the input. The model, which we refer to as discriminative encoder (DisCoder), assumes that the empirical distribution of the input variables is the same as their true distribution. The model is completely specified by the encoding and the prior distribution on the latent features. In the experiments presented in this chapter, we restrict ourselves to categorical latent features only. In the next chapter, we will learn real-valued latent features by allowing the prior distribution over the latent features to be specified hierarchically.

The rest of the chapter is organized as follows: In Section 4.2, we motivate the choice of our

model. The steps involved in training the model are discussed in Section 4.3. In Section 4.4, we discuss an adversarial regularization strategy that prevents the model from getting stuck to its initial configuration. In Section 4.5, we present the results achieved by our model on clustering and semi-supervised learning tasks. Most of the results that appear in this chapter, also appear in our paper (Pandey and Dukkipati, 2017).

4.2 The proposed model

We denote the observed features as $\mathbf{x} = (x_1, \ldots, x_d)$, whereas the corresponding latent features are denoted by $\mathbf{z} = (z_1, \ldots, z_m)$. The distribution $p_\theta(\mathbf{z}|\mathbf{x})$ will be referred to as the encoding distribution, whereas $p_\theta(\mathbf{x}|\mathbf{z})$ will be referred to as the decoding distribution. The parameters of the distribution will be given by θ. In the later parts of the chapter, we will parameterize the distributions using neural networks. In particular, the neural networks that parameterize the encoding distribution will be denoted by E. The corresponding network will be referred to as encoding network. We will denote the number of samples in the training data by n, and the size of the latent features by m.

4.2.1 Motivation

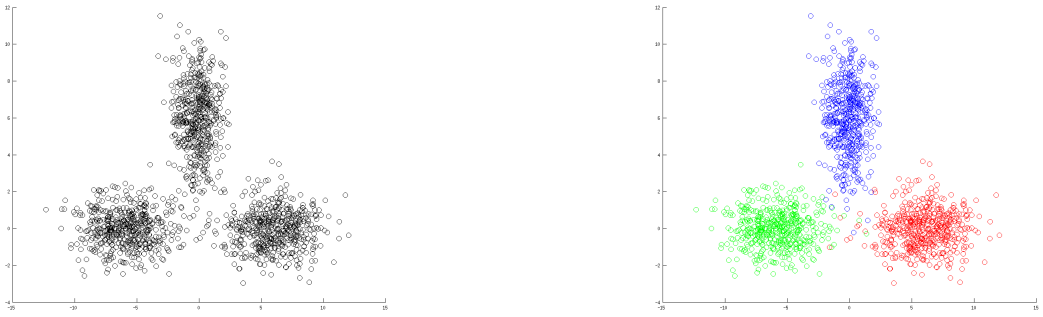
A common approach to address the problem of unsupervised learning with latent features is by defining a joint distribution over the observed and the latent features. Such a model is referred to as generative model in the sequel. One can then maximize the marginal distribution

$$p_\theta(\mathbf{x}) = \sum_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z})$$

of the observed features. The MAP estimate of the latent features given the observed features can then be used as a representation for the data. That is,

$$\hat{\mathbf{z}} = \operatorname*{argmax}_{\mathbf{z}} \log p_\theta(\mathbf{x}, \mathbf{z}) \,.$$

The generative approach is one of the most common approaches for modeling latent features, and has been successfully used in Gaussian mixture models (GMM), hidden Markov models (HMM), restricted Boltzmann machines (RBM), variational autoencoder (VAE), latent Dirichlet allocation (LDA) etc. Despite its almost ubiquitous success, this approach for learning latent features suffers from a fatal flaw: It is possible to maximize $p_\theta(\mathbf{x})$ without changing the encoding distribution $p_\theta(\mathbf{z}|\mathbf{x})$ at all. This is particularly true when the choice of decoding distribution $p_\theta(\mathbf{x}|\mathbf{z})$ is flexible enough to fit the data distribution. In such a scenario, the model



(a) (b)

Figure 4.1: An artifical dataset of points (a) and the clusters detected by the GMM model (b).

can choose to ignore the latent features $\mathbf{z}$ in the decoding distribution $p_\theta(\mathbf{x}|\mathbf{z})$ by equating $p_\theta(\mathbf{x}|\mathbf{z})$ to $p_\theta(\mathbf{x})$. In this case,

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} \approx \frac{p_\theta(\mathbf{x})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} = p_\theta(\mathbf{z})\,.$$

Hence, the resultant latent features $\mathbf{z}$ are independent of the observed features $\mathbf{x}$.

To make the above idea more concrete, let us consider the simplest generative model with latent features, namely, a Gaussian mixture model (GMM). In the case of GMM, the latent features are one-hot vectors, that correspond to the cluster the corresponding datapoint belongs. Moreover, the decoding distribution $p(\mathbf{x}|\mathbf{z} = k)$ is a Gaussian distribution with mean μ_k and covariance matrix Σ_k. (Note that $\mathbf{z} = k$ implies that the k^{th} unit of the vector one-hot $\mathbf{z}$ is 1). In the current example, we assume that the prior distribution on the clusters is a uniform distribution. The marginal log-likelihood of the data can be written as

$$\mathcal{L}(\mu, \Sigma, \pi) = \sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(i)}; \mu_k, \Sigma_k)\,,$$

where $\pi_k, 1 \leq k \leq K$ are the mixing coefficients and sum up to 1. We maximize the marginal log-likelihood of the data shown in Figure 4.1a assuming the above GMM model with $K = 3$. As expected, the GMM model is easily able to detect the three clusters present in the data (see Figure 4.1b). The three clusters are colored in red, blue and green.

Next, we allow the decoding distribution $p(\mathbf{x}|\mathbf{z})$ to be as flexible as the marginal distribution

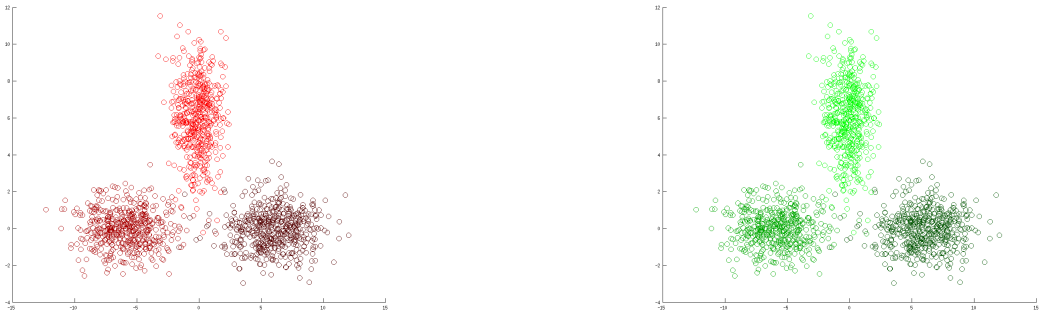


Figure 4.2: Two examples of clusters learnt when the encoding distribution is a 'mixture of Gaussians' rather than a Gaussian distribution. Note that all points are correctly assigned to different subclusters, but the latent representation of all the points is the same in both cases.

$p(\mathbf{x})$ itself. In particular, the distribution of a point in the k^{th} cluster is given by

$$p(\mathbf{x}|\mathbf{z}=k) = \sum_{l=1}^{K} \alpha_{kl} \mathcal{N}(\mathbf{x}; \mu_{kl}, \Sigma_{kl}) \,,$$

where α_{kl}, μ_{kl} and Σ_{kl} are parameters to be determined by the learning algorithm. Hence, each cluster is distributed according to a Gaussian mixture model. For visualization, we color the sub-clusters within the cluster by varying intensity of the same color. The new log-likelihood is given by

$$\bar{\mathcal{L}}(\mu, \Sigma, \pi, \alpha) = \sum_{i=1}^{n} \log \sum_{k,l=1}^{K} \pi_l \alpha_{kl} \mathcal{N}(\mathbf{x}^{(i)}; \mu_{kl}, \Sigma_{kl}) \,.$$

Note that the above problem is equivalent to learning a Gaussian mixture model with 9 Gaussian components, each of which can have a different mixing proportion. There are several globally optimal solution to the above problem. Two of the solutions are given in Figure 4.2. In both the cases, all the points have been assigned to the same cluster, and hence have the same latent representation. Note that the marginal distribution over the data $p(\mathbf{x})$ remains the same in all the cases.

The above example was a simple case, where it was easy to detect that the latent features have been mistakenly identified as subclusters. Another example of a model with a flexible decoding distribution is given in (Bowman et al., 2016), where the decoding distribution $p(\mathbf{x}|\mathbf{z})$ learns to ignore the latent features $\mathbf{z}$ during decoding. In general, the choice of decoding model is simple enough to prevent this from happening. For instance, in HMM, RBM, VAE and LDA, the decoding distribution factorizes completely, whereas in GMM, the decoding distribution is

a Gaussian distribution. This forces the model to utilize the latent features to effectively model the data distribution.

Hence, generative models rely on weak decoding distributions for learning useful representations. In general, a good generative model does not guarantee a useful latent representation. The problem lies in the fact, that generative models optimize $p_\theta(\mathbf{x})$, and may or may not choose to care about the encoding distribution $p_\theta(\mathbf{z}|\mathbf{x})$. To address this specific issue, we propose a model that specifically assigns zero probability to any point $\mathbf{x}$ that does not occur in the training data. One can see that such a modeling strategy is intinsically used by discriminative models for supervised learning tasks. In particular, if $(\mathbf{x}^{(1)}, \mathbf{z}^{(1)}), \ldots, (\mathbf{x}^{(n)}, \mathbf{z}^{(n)})$ are the observed points and their labels, the joint log-likelihood $\mathcal{L}$ can be written as

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)}) + \sum_{i=1}^{n} \log p_\theta(\mathbf{z}^{(i)}|\mathbf{x}^{(i)}) \,.$$

In discriminative models, the first term in the above equation is independent of the parameters in the second term, and hence can be maximized independently. The maximum occurs, when $p_\theta(\mathbf{x}) = \frac{1}{n}$ for all the observed points, and 0 for any point that does not occur in the training data.

Unfortunately, when the labels $\mathbf{z}$ are unknown, the same strategy can not be used, since the model collapses to a single $\mathbf{z}$ with $p_\theta(\mathbf{z}|\mathbf{x}) = 1$ for all $\mathbf{x}$. To prevent this from happening, we couple the encoding distribution $p_\theta(\mathbf{z}|\mathbf{x})$ with a function $f(\mathbf{z})$, and define a new distribution

$$q_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x}) f(\mathbf{z}) \,.$$

To obtain the value of f at $\mathbf{z}$, we force the distribution to satisfy the following constraints:

C1. The marginal distribution of $q_\theta(\mathbf{x}, \mathbf{z})$ over $\mathbf{z}$ is $p_\theta(\mathbf{z})$. This is done to prevent the model from collapsing to a single $\mathbf{z}$. This also allows us to incorporate the prior information about the latent features into our model. For instance, if we know that the components of $\mathbf{z}$ should form a Markov chain, we can incorporate that information easily into the model.

C2. The marginal distribution of q_θ over $\mathbf{x}$ assigns 0 probability to any point that does not occur in the training data. This ensures that the only way to improve $q_\theta(\mathbf{x}, \mathbf{z})$ is by improving the encoding distribution. This is in stark contrast with generative models, where the marginal distribution over $\mathbf{x}$ is the quantity of interest.

Proposition 3 *Let q_θ be a distribution of the form given in (4.1) that satisfies the constraints*

C1 and C2. Then

$$q_\theta(\mathbf{x}, \mathbf{z}) = \frac{p_\theta(\mathbf{z}|\mathbf{x})p_\theta(\mathbf{z})}{\sum_{j=1}^{n} p_\theta(\mathbf{z}|\mathbf{x}^{(j)})}, \tag{4.1}$$

for any $\mathbf{x}$ *observed in the training data and,* 0 *otherwise.*

Proof: The proof of the above proposition follows directly from the constraints C1 and C2. From the second constraint, we have $q_\theta(\mathbf{x}) = 0$ for all $\mathbf{x}$ not present in the training data. This can be rewritten as $\sum_{\mathbf{z}} q_\theta(\mathbf{x}, \mathbf{z}) = 0$ for all $\mathbf{x}$ not present in the training data. The non-negativity of $q_\theta(\mathbf{x}, \mathbf{z})$ implies that each of these quantities must be zero. From the first constraint, we have

$$\begin{aligned}
q_\theta(\mathbf{z}) = p_\theta(\mathbf{z}) &\implies \sum_{\mathbf{x}} q_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{z}) \\
&\implies \sum_{i=1}^{n} q_\theta(\mathbf{x}^{(i)}, \mathbf{z}) = p_\theta(\mathbf{z}) \\
&\implies \sum_{i=1}^{n} p_\theta(\mathbf{z}|\mathbf{x}^{(i)}) f(\mathbf{z}) = p_\theta(\mathbf{z}) \\
&\implies f(\mathbf{z}) = \frac{p_\theta(\mathbf{z})}{\sum_{i=1}^{n} p_\theta(\mathbf{z}|\mathbf{x}^{(i)})}.
\end{aligned}$$

□

We optimize the distribution q_θ for unsupervised feature learning in the rest of the chapter. Note that the model is completely specified by the choice of the prior $p_\theta(\mathbf{z})$ and the encoding distribution $p_\theta(\mathbf{z}|\mathbf{x})$.

4.2.2 Discriminative Encoder (DisCoder)

In the previous section, we motivated the choice of the model that can be used for unsupervised feature learning. The model is referred to as discriminative encoder (DisCoder) in the sequel. In this section, we discuss the model in further detail, and justify the name, *discriminative encoder*, for the model.

Given an *i.i.d* sequence of unlabeled samples $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$, the joint likelihood function of a DisCoder can be written as a function of the corresponding latent features $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}$, and

parameters of the model θ.

$$\mathcal{L}(\theta, \mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}) = \sum_{i=1}^{n} \log q_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) \,,$$

where $q_\theta(\mathbf{x}, \mathbf{z})$ is defined as in (4.1). In order to optimize the above objective, we use an EM-like procedure, that alternates between the selection of $\mathbf{z}^{(i)}$ and optimization with respect to θ.

In the selection step, for each $\mathbf{x}^{(i)}$ in the training data, we find the best $\mathbf{z}$, that is, the $\mathbf{z}$ that maximizes $\log q_\theta(\mathbf{x}^{(i)}, \mathbf{z})$. The quantity $\log q_\theta(\mathbf{x}^{(i)}, \mathbf{z})$ can be expanded as

$$\log q_\theta(\mathbf{x}^{(i)}, \mathbf{z}) = \log p_\theta(\mathbf{z}|\mathbf{x}^{(i)}) + \log p_\theta(\mathbf{z}) - \log \sum_{j=1}^{n} p_\theta(\mathbf{z}|\mathbf{x}^{(j)}) \,. \tag{4.2}$$

Hence, the optimal latent representation should be chosen to maximize $\log p_\theta(\mathbf{z}|\mathbf{x}^{(i)})$, but minimize $\log \sum_{j=1}^{n} p_\theta(\mathbf{z}|\mathbf{x}^{(j)})$. In other words, the latent representation should have high probability of being assigned to $\mathbf{x}^{(i)}$, but low probability of being assigned to any other $\mathbf{x}^{(j)}, j \neq i$.

Once we have found the optimal latent representation for $\mathbf{x}^{(i)}$, we equate it to $\mathbf{z}^{(i)}$. Next, we optimize the objective with respect to the parameters of the model. In expanded form, the objective as a function of θ can be written as

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \log p_\theta(\mathbf{z}^{(i)}|\mathbf{x}^{(i)}) - \sum_{i=1}^{n} \log \sum_{j=1}^{n} p_\theta(\mathbf{z}^{(i)}|\mathbf{x}^{(j)}) + \sum_{i=1}^{n} \log p_\theta(\mathbf{z}^{(i)}) \,. \tag{4.3}$$

For instance, when $p_\theta(\mathbf{z}|\mathbf{x})$ is normally distributed with mean $E(\mathbf{x})$ and variance $1/2$, and the prior on the latent features is standard normal, the objective can be written as

$$-\mathcal{L}(\theta) = \sum_{i=1}^{n} ||E(\mathbf{x}^{(i)}) - \mathbf{z}^{(i)}||^2 + \sum_{i=1}^{n} \log \sum_{j=1}^{n} \exp(-||E(\mathbf{x}^{(j)}) - \mathbf{z}^{(i)}||^2) + C \,, \tag{4.4}$$

where C contains the terms that do not have any trainable parameters. Since $\mathbf{z}^{(i)}$ are fixed for this step, the terms that depend on $\mathbf{z}^{(i)}$ alone have been removed from the objective. For a fixed $\mathbf{x}^{(i)}$, this step trains the model to maximize the probability of the $\mathbf{z}^{(i)}$ selected in the previous step while simultaneously lowering the probability of other $\mathbf{z}^{(j)}, j \neq i$.

Both the steps of training try to ensure that the learnt representations are as dissimilar to each other as possible, while simultaneously satisfying the requirement of the prior distribution. Hence, the representations learn to capture those features that vary among the samples, while completely ignoring the features common to all the samples. Hence, we name the model as

discriminative encoder or DisCoder. The name also stresses the fact, that we do not employ a decoder for reconstructing the input.

If we have access to a set of labeled samples, DisCoder can utilize those samples to improve upon the learnt embeddings. We achieve this by adding a term to the objective to maximize the conditional log-likelihood over the labeled samples. If we denote the set of labeled samples as $\{(\mathbf{x}_s^{(1)}, \mathbf{z}_s^{(1)}), \ldots, (\mathbf{x}_s^{(m)}, \mathbf{z}_s^{(m)})\}$, the new objective can be written as:

$$\mathcal{L}(\theta, \mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}) = \sum_{i=1}^{n} \log q_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) + \sum_{i=1}^{m} \log p_\theta(\mathbf{z}_s^{(i)} | \mathbf{x}_s^{(i)}) \,.$$

4.3 Optimization

We parametrize the encoding distribution $p_\theta(\mathbf{z}|\mathbf{x})$ using a neural network E. In particular, when the encoding distribution is a Gaussian, the mean of the distribution is the output of a network, while the variance is kept fixed. Similarly, when the encoding distribution is categorical, the mean of the categorical distribution (a probability distribution over the categories) is output by a neural network. As mentioned in the previous section, the optimization proceeds in the following two steps:

i. latent feature selection, and

ii. encoding network optimization.

The latent feature selection step differs depending on the choice of the latent features, while the encoding network optimization step remains essentially the same irrespective of the latent features. We first derive the two optimization steps for categorical encoding distribution, and then extend it for the case of continuous encoding distributions.

4.3.1 Categorical latent features

Categorical latent features are desirable when the data can be grouped into non-overlapping subsets, such as classes or clusters. Hence, modeling with categorical latent features, is often referred to as clustering or classification, and dealt differently from unsupervised feature learning. However, in this chapter, we assume clustering to be a special case of unsupervised feature learning, when the latent representation is forced to be a binary vector with exactly one entry set to 1.

For categorical latent features, the encoding network takes a sample $\mathbf{x}^{(i)}$ as input, and outputs the mean of the encoding categorical distribution, that is, a vector of non-negative

entries that sum up to 1. The latent feature selection step for categorically distributed latent features can be obtained by exponentiating (4.2)

$$\mathbf{z}^{(i)} = \operatorname*{argmax}_{\mathbf{z}} \frac{\prod_{k=1}^{m} E_k(\mathbf{x}^{(i)})^{z_k}}{\sum_{j=1}^{n} \prod_{k=1}^{m} E_k(\mathbf{x}^{(j)})^{z_k}} . \tag{4.5}$$

In particular, $z_\ell^{(i)} = 1$, if

$$\ell = \operatorname*{argmax}_{k \in \{1,\dots,m\}} \frac{E_k(\mathbf{x}^{(i)})}{\sum_{j=1}^{n} E_k(\mathbf{x}^{(j)})} .$$

The last equation follows from the fact that only one component of $\mathbf{z}$ can be non-zero at a time. The quantity within the argmax can be computed efficiently and independently for each component. Note that for categorical latent features, the latent feature selection step can be performed exactly, and hence no approximations are involved in this step.

Next, one needs to train the encoding network for the $\mathbf{z}$ selected in the previous step. For categorical latent features, the objective in (4.3) can be written as

$$\mathcal{L}(E) = \sum_{i=1}^{n} \sum_{k=1}^{m} z_{ik} \log E_k(\mathbf{x}^{(i)}) - \sum_{i=1}^{n} \log \sum_{j=1}^{n} \prod_{k=1}^{m} E_k(\mathbf{x}^{(j)})^{z_{ik}} .$$

Note that the objective is a differentiable function of the parameters of the neural network E. Hence, we use minibatch stochastic gradient descent to maximize the above objective. Given a minibatch of samples, we optimize the restriction of the objective to this minibatch, while ignoring the examples from other minibatches. In particular, the summation inside the log of second term in the above equation is evaluated only over the minibatch. This reduces, to a large extent, the computational complexity of optimization from $\mathcal{O}(n^2)$ to $\mathcal{O}(nb)$, where b is the batch size. Furthermore, the randomness introduced by the restriction serves to regularize the encoding network, and encourages it to explore other choices of $\mathbf{z}$.

4.3.2 Real-valued latent features

For real valued latent features $\mathbf{z}$, there are several possible choices for prior $p(\mathbf{z})$ and encoding distribution $p(\mathbf{z}|\mathbf{x})$. In the simplest scenario, one can assume the prior and encoding distributions to be normally distributed. Alternatively, if the data is known to form clusters in the latent space, a Gaussian mixture model with unknown means can be used as the prior.

Similarly, more complicated encoding distributions can be chosen as well.

Let us assume that the encoding distribution is Gaussian with a constant variance λ. In our experiments, we select the variance by cross-validation on a validation set for a secondary task. Furthermore, we assume that the prior distribution on the latent features is standard normal $\mathcal{N}(0, I)$. For a fixed $\mathbf{x}^{(i)}$, the mean of the encoding distribution $E(\mathbf{x}^{(i)})$ is the output of the network. The latent feature selection step for normally distributed features with a standard normal prior can be written as:

$$\mathbf{z}^{(i)} = \underset{\mathbf{z}}{\operatorname{argmin}} \frac{||E(\mathbf{x}^{(i)}) - \mathbf{z}||^2}{2\lambda} + \log \sum_{j=1}^{n} \exp\left(-\frac{||E(\mathbf{x}^{(j)}) - \mathbf{z}||^2}{2\lambda}\right) + \frac{||\mathbf{z}||^2}{2}. \tag{4.6}$$

The above objective is a differentiable function of $\mathbf{z}$. Hence, one can use stochastic gradient descent (SGD) or its extensions (Kingma and Ba, 2014) for optimizing the above objective with respect to $\mathbf{z}$. Unfortunately, the optimization needs to be done for each example individually in every iteration of training. That is, for each $\mathbf{x}$, one needs to find the corresponding $\mathbf{z}$ that minimizes the above objective in every iteration. Since this involves an optimization within an optimization algorithm, it can be quite expensive.

Alternatively, one can amortize the cost of learning the latent features over the entire dataset. Hence, instead of learning a $\mathbf{z}$ for each $\mathbf{x}$ individually, we can train a neural network that takes $\mathbf{x}$ as input and generate $\mathbf{z}$ as output. Let us refer to the neural network as $E_{\mathbf{z}}$ to signify the fact that the network encodes the latent features. Hence, the problem of learning a $\mathbf{z}$ for each $\mathbf{x}$ reduces to the problem of training the network $E_{\mathbf{z}}$ that computes the latent features for all $\mathbf{x}$. The new objective can be written as

$$\mathcal{L}(E, E_{\mathbf{z}}) = \sum_{i=1}^{n} \frac{||E(\mathbf{x}^{(i)}) - E_{\mathbf{z}}(\mathbf{x}^{(i)})||^2}{2\lambda} + \sum_{i=1}^{n} \log \sum_{j=1}^{n} \exp\left(-\frac{||E(\mathbf{x}^{(j)}) - E_{\mathbf{z}}(\mathbf{x}^{(i)})||^2}{2\lambda}\right) + \sum_{i=1}^{n} \frac{||E_{\mathbf{z}}(\mathbf{x}^{(i)})||^2}{2}.$$

Rather than training two independent networks E and $E_{\mathbf{z}}$, it is reasonable to allow E and $E_{\mathbf{z}}$ to share all the layers except for the last one. One can then train the two networks alternatively until convergence. Alternatively, if we equate the latent feature network $E_{\mathbf{z}}$ to the encoding network E completely, the objective gets further simplified:

$$\mathcal{L}(E) = \sum_{i=1}^{n} \log\left[1 + \sum_{j \neq i} \exp\left(-\frac{||E(\mathbf{x}^{(i)}) - E(\mathbf{x}^{(j)})||^2}{2\lambda}\right)\right] + \sum_{i=1}^{n} \frac{||E(\mathbf{x}^{(i)})||^2}{2}. \tag{4.7}$$

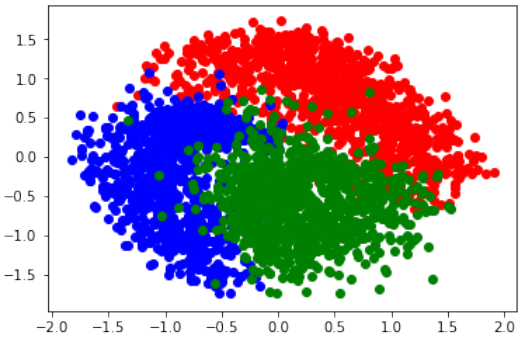


Figure 4.3: Two dimensional embeddings learnt by the DisCoder on the first three classes of MNIST dataset. The different colors correspond to different classes

Such an assumption is reasonable, since we expect the encoding network to output the latent features at the end of training, that is, the latent feature network $E_{\mathbf{z}}$ learn to mimic the encoding network E. Moreover, the new objective requires the training of a single network, which can be achieved with relative ease. Hence, in the rest of the thesis, we use the above objective for training a DisCoder with real valued latent features. Note that the objective is simply forcing all the encoded points to be as far away from each other as possible while simultaneously satisfying the prior, thereby forcing the encoding network to capture discriminatory information.

As in the case of categorical latent features, we use minibatch SGD and its adaptive extensions to minimize the above objective. Given a minibatch of samples, we optimize the restriction of the objective to this minibatch, while ignoring the examples from other minibatches. In particular, the summation inside the log term in (4.7) is evaluated over the minibatch only, thereby reducing the computational complexity of optimization from $\mathcal{O}(n^2)$ to $\mathcal{O}(nb)$, where b is the batch size.

As a toy experiment, we choose the first three classes of MNIST that correspond to the characters 0, 1 and 2 and learn two dimensional embeddings for these classes. The results are shown in Figure 4.3. These classes were chosen since they are easily separable in two dimensions. Note that the objective in (4.7) forces all the embeddings to be as far apart as possible while simultaneously satisfying the constraints imposed by the prior. A Gaussian prior on the latent features only serves to ensure that the embeddings form a Gaussian distribution. In order to learn interesting latent representations, the prior on the latent features needs to be more interesting. Hence, we limit ourselves to categorical latent features in this chapter. In the next chapter, we impose hierarchical Bayesian priors on the latent features, thereby forcing the model to learn interesting representations.

4.3.3 Complexity of training

As mentioned in the previous sections, the complexity of each step (latent feature selection as well as network optimization) is quadratic in the size of the batch, that is, $\mathcal{O}(b^2)$. However, when the batch size is small enough (< 2000 samples), the empirical running time depends linearly on the batch size rather than quadratically. This is because, most of the running time is spent in forward and backpropagating through the network, which needs to be done only once for the entire batch. Very little time is utilized in the computation of the distribution of the DisCoder given the output of the network, or the gradient of the objective function with respect to the network output.

4.4 Regularizing the model for categorical latent features

As mentioned in the previous section, the training of the model with categorical latent features alternates between latent feature selection and encoding network optimization. The categorical latent features (also referred to as labels) selected at the beginning of training are completely random. Now, if the encoding network is sufficiently powerful, and the encoding network optimization step proceeds for a sufficiently long time, the emodel may attempt to fit to the initial labels, despite the fact that they are completely random.

This is a problem faced by almost all the approaches for categorical representation learning. For instance, in clustering based approaches (Yang et al., 2016, Xie et al., 2016, Springenberg, 2015), if the encoding network that maps the samples to clusters is sufficiently powerful, and the training of encoding network proceeds for a sufficiently long time, keeping the clusters fixed, any choice of the clusters can be predicted with absolute certainty. This problem is addressed in (Yang et al., 2016) by first performing an overclustering of the data into a large number of clusters, and then merging the clusters as training proceeds. Having a large number of clusters ensures that two disparate points have lesser probability of ending up in the same cluster, when a good representation has not yet been learned by the encoding network.

Alternatively, Xie et al. (2016) addresses this problem by performing soft assignment rather than hard assignment. Specifically, the encoding distribution is modified in every step to be closer to a lesser entropy version of itself. Another approach used in (Springenberg, 2015) minimizes the entropy of the encoding distribution for real images, while simultaneously maximizing the entropy for fake images. Entropy maximization for fake images serves as an effective regularization scheme, allowing the model to achieve state of the art performance for clustering tasks. We use the following related scheme in our model for learning categorical latent features.

4.4.1 Adversarial regularization

The adversarial regularization works as follows: While training the DisCoder to maximize $\sum_{i=1}^{n} \log q_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i)})$ for real samples $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$, we simultaneously train it to be totally confused about a set of *fake* samples $\bar{\mathbf{x}}^{(1)}, \ldots, \bar{\mathbf{x}}^{(n)}$. We force this by maximizing the probability over all the latent features for fake images. Hence, the new objective becomes:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \log q_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) + \sum_{i=1}^{n} \sum_{\mathbf{z}} p(\mathbf{z}) \log p_\theta(\mathbf{z}|\bar{\mathbf{x}}^{(i)}) \,.$$

The above regularization forces the model to learn mappings from data to categories that are completely unsure about samples that do not occur in the training data. A similar scheme for regularization was also used in (Springenberg, 2015). Note that the adversarial regularization used in this chapter must not be confused with adversarial training used in (Goodfellow et al., 2014). We choose the term adversarial regularization for lack of a better name.

4.4.2 Adversarial regularization as label smoothing

Supervised classification proceeds by minimizing the cross-entropy loss between the true distribution over the labels for a given input $\mathbf{x}$, and the predicted distribution. In particular, if $p^{\text{real}}(\mathbf{x}, \mathbf{z})$ is the true distribution over the input $\mathbf{x}$ and targets $\mathbf{z}$, and $p_\theta(\mathbf{z}|\mathbf{x})$ be the distribution predicted by the model, the cross entropy loss is given by

$$\mathcal{L}(\theta) = -\sum_{\mathbf{z}} p^{\text{real}}(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{z}|\mathbf{x}) \,.$$

This forces the model distribution p_θ to learn to mimic the true distribution. In general, we do not have access to the true distribution, and hence, we use the empirical distribution as the true distribution. However, if the labels are noisy or if the training data contains very few samples, this may result in overfitting. The resultant model may not generalize well to new data. Hence, Szegedy et al. (2016) have proposed to replace the true distribution by a weighted sum in the cross entropy term, as follows:

$$\mathcal{L}(\theta) = -\sum_{\mathbf{z}} \left[(1-\epsilon)p^{\text{real}}(\mathbf{z}|\mathbf{x}) + \epsilon p(\mathbf{z})\right] \log p_\theta(\mathbf{z}|\mathbf{x}) \,,$$

where $p(\mathbf{z})$ is a prior distribution over the labels, and ϵ is a value that weighs the prior with the true distribution.

In this section, we show that the proposed adversarial regularization scheme achieves label-

smoothing where the weights for the prior and the true distribution are determined by the true distribution over the input $p^{\text{real}}(\mathbf{x})$, and the distribution of generated samples $p^{\text{fake}}(\mathbf{x})$. We analyze the regularization scheme in the presence of labeled data only. This analysis does not pertain to the DisCoder model. Rather we analyze supervised classification with the proposed adversarial regularization. The supervised classification objective with the proposed adversarial regularization can be written as given below:

$$\mathbb{E}_{(\mathbf{x},\mathbf{z})\sim p^{\text{real}}} \log p_\theta(\mathbf{z}|\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p^{\text{fake}}} \sum_{\mathbf{z}} p(\mathbf{z}) \log p_\theta(\mathbf{z}|\mathbf{x}) \,.$$

Maximizing the above objective with respect to $p_\theta(\mathbf{z}|\mathbf{x})$ under the constraint that $\sum_{\bar{\mathbf{z}}} p_\theta(\bar{\mathbf{z}}|\mathbf{x}) = 1$, yields

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p^{\text{real}}(\mathbf{x},\mathbf{z}) + p^{\text{fake}}(\mathbf{x})p(\mathbf{z})}{\sum_{\bar{\mathbf{z}}} p^{\text{real}}(\mathbf{x},\bar{\mathbf{z}}) + p^{\text{fake}}(\mathbf{x})p(\bar{\mathbf{z}})} \tag{4.8}$$

$$= \frac{p^{\text{real}}(\mathbf{x})p^{\text{real}}(\mathbf{z}|\mathbf{x}) + p^{\text{fake}}(\mathbf{x})p(\mathbf{z})}{\sum_{\bar{\mathbf{z}}} p^{\text{real}}(\mathbf{x})p^{\text{real}}(\bar{\mathbf{z}}|\mathbf{x}) + p^{\text{fake}}(\mathbf{x})p(\bar{\mathbf{z}})} \tag{4.9}$$

$$\propto p^{\text{real}}(\mathbf{x})p^{\text{real}}(\mathbf{z}|\mathbf{x}) + p^{\text{fake}}(\mathbf{x})p(\mathbf{z}) \tag{4.10}$$

Hence, the trained distribution p_θ learns a weighted linear combination of the true distribution of labels given the input $p^{\text{real}}(\mathbf{z}|\mathbf{x})$, and the prior distribution on the labels $p(\mathbf{z})$. The weights of the linear combination are the probabilities of the point $\mathbf{x}$ under the real and the fake distribution. When the point $\mathbf{x}$ has very low probability under p^{fake}, we get

$$p_\theta(\mathbf{z}|\mathbf{x}) \approx \frac{p^{\text{real}}(\mathbf{x},\mathbf{z})}{\sum_{\bar{\mathbf{z}}} p^{\text{real}}(\mathbf{x},\bar{\mathbf{z}})} = p^{\text{real}}(\mathbf{z}|\mathbf{x}) \,.$$

Hence, the trained distribution p_θ learns to mimic the true distribution p^{real} of labels given the data. Similarly, for any point that has very low probability under p^{real} (and hence $p^{\text{real}}(\mathbf{x}) \approx 0$), the model assigns the prior distribution on labels to the predictive distribution p_θ.

By choosing the fake distribution carefully, one can regularize the classifier. This regularization scheme is a data-dependent version of label-smoothing. The aim of this regularization scheme is to prevent the model from being confident about labels of samples that do not lie on the data manifold, and hence $p^{\text{real}}(\mathbf{x}) \approx 0$. This can be achieved by finding samples on which the trained model is confident, and then training the model to be confused about these samples.

Below, we list several methods for generating samples from the fake distribution.

Class-conditioned generation: In order to generate fake image samples, one can couple the encoder with a generator G, that takes Gaussian noise $\mathbf{n}$ and latent features $\mathbf{z}$ as input and generates fake samples $\bar{\mathbf{x}}$ as output, that is,

$$\bar{\mathbf{x}} = G(\mathbf{n}, \mathbf{z})$$

The generator is trained to generate samples $\bar{\mathbf{x}}$, such that the encoder can extract $\mathbf{z}$ back from $\bar{\mathbf{x}}$. We achieve this by training the generator to maximize $\log p_\theta(\mathbf{z}|\bar{\mathbf{x}} = G(\mathbf{n}, \mathbf{z}))$. Note that the discriminator is simultaneously being trained to be completely confused about the latent features of the fake examples. This creates an adversarial game akin to the adversarial game of generative adversarial networks (Goodfellow et al., 2014). The choice of the generative model allows us to visualize the latent features, and hence, determine if the model is learning anything meaningful. In particular, if we wish to determine the information captured by the i^{th} latent feature, we set it to 1 and the other latent features to 0, couple it with Gaussian noise and pass it through the generator.

Feature matching: Class-conditioned generation results in images that are sharp with the object immediately identifiable. Note that the encoding network is simultaneously being trained to be totally confused about the generated samples. However, if the underlying class of the fake image is immediately identifiable, training a network to be confused about the image results in unstable training. Hence, while class conditioned generation is useful when the underlying task is to generate sharp images, it is not really suitable for classification tasks.

In order to address this problem, we use feature matching for classification tasks. Noise is passed through the generator to generate a batch of fake images $\bar{\mathbf{x}}$, that is, $\bar{\mathbf{x}} = G(\mathbf{n})$. The statistics of the fake batch is then compared with the statistics of the real batch. As suggested in (Salimans et al., 2016), we pass the fake batch as well as the real batch through the encoding network, and compute the squared difference between the means of encodings of the real and fake batch for the penultimate layer. The generator is trained to minimize the distance between the average encodings for the penultimate layer. The resultant images are not very sharp. However, this approach works quite well for classification tasks as has been observed in (Salimans et al., 2016).

Negative Sampling: To generate fake documents, we randomly select words from the vocabulary, based on the frequency with which they occur in the corpus. The size of the document is Poisson distributed whose mean is the mean of the length of the documents that occur in the corpus.

Algorithm 3 combines all the above steps for training the model.

4.5 Experiments

In order to evaluate the capability of the model for learning meaningful representations, we evaluate the learnt categorical embeddings across several tasks. Here, the categorical latent feature indicates the class (for semi-supervised classification) or the cluster (for clustering) that a unlabeled sample belongs to. For each task, we evaluate our model against the state-of-the-art models for that task. The code for the experiments in this chapter is available at *https://github.com/gauravpandeyamu/DisCoder*. The real-valued embeddings are not very useful, unless coupled with hierarchical models such as mixture models and topic models. Hence, we defer our experiments on real-valued embeddings for the next chapter where discriminative encoders with hierarchical priors are discussed.

We evaluate the one-hot embeddings learnt by the model for the task of clustering on MNIST, and 20-newsgroup dataset. We also evaluate the embeddings for the task of semi-supervised classification on CIFAR-10 and SVHN dataset. In all our experiments, we use a batch size of 100 images, and Adam optimizer with a constant learning rate of 0.0002. Since neural networks are likely to get stuck in local optima, we repeat the experiment 10 times, and report the mean and the standard deviation.

4.5.1 MNIST

The MNIST (LeCun et al., 1998) is a relatively simple dataset of digits from 0 to 9. We normalize the images, by dividing the pixel intensities by 255. We evaluate DisCoder on this dataset for the clustering task into 20 clusters. We use generative adversarial regularization for regularizing the DisCoder as discussed in Section 4.4. The architecture of the generator and encoding network are provided in Table 4.1.

The clustering error achieved by the various models on MNIST dataset is shown in Table 4.2. The accuracy is computed as follows: For each cluster i, we identify the sample $\mathbf{x}$ that maximizes $p(\mathbf{z} = i|\mathbf{x})$. Next, the label of the selected $\mathbf{x}$ is assigned to all the samples in this cluster. The test error is computed based on the labels assigned to the points using this procedure. Such a scheme is also used for evaluating clustering in (Makhzani et al., 2015).

For the sake of completeness, we also show the results when adversarial regularization is not used. As can be observed, adversarial regularization is extremely crucial for achieving good performance when one-hot embeddings are used in DisCoder. Also noteworthy, is the high standard deviation of an unregularized DisCoder, which indicates that the model overfits to

Algorithm 3: Minibatch training of DisCoder

Input: Observed features $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$
Output: An encoder E and generator G (if adversarial regularization is enabled)

1 Repeat the following steps until convergence

1. Randomly select a batch of observed features B.
2. Forward propagate them through the encoder to obtain the encoding distribution $p_\theta(\mathbf{z}|\mathbf{x}), \mathbf{x} \in B$.
3. Select the latent features $\mathbf{z_x}$ for all $\mathbf{x} \in B$ by maximizing $\log q_\theta(\mathbf{x}, \mathbf{z})$ for each $\mathbf{x}$ in the batch.
4. Train the encoding network to maximize

$$\mathcal{L}(\theta) = \frac{1}{B} \sum_{\mathbf{x} \in B} \log q_\theta(\mathbf{x}, \mathbf{z_x})$$

5. If semi-supervised learning is enabled:
   (a) Randomly select a batch B_s of labeled samples.
   (b) Train the encoding network to maximize $\frac{1}{B_s} \sum_{(\mathbf{x},\mathbf{z}) \in B_s} \log p_\theta(\mathbf{z}|\mathbf{x})$.
6. If class conditioned generation is enabled:
   (a) Concatenate noise $\mathbf{n}$ with the latent features obtained in the previous step $\mathbf{z_x}$ and propagate them through the generator to get a batch of fake samples $\bar{B}$.
   (b) Train the encoding network to maximize

$$\frac{1}{\bar{B}} \sum_{\mathbf{x} \in \bar{B}} \sum_{\mathbf{z}} p(\mathbf{z}) \log p_\theta(\mathbf{z}|\bar{\mathbf{x}}^{(i)})$$

   (c) Train the generator to maximize $\log p_\theta(\mathbf{z}|\bar{\mathbf{x}} = G(\mathbf{n}, \mathbf{z}))$
7. If feature matching is enabled:
   (a) Forward propagate noise $\mathbf{n}$ through the generator to generate a batch of fake samples.
   (b) Train the encoding network to maximize

$$\frac{1}{\bar{B}} \sum_{\mathbf{x} \in \bar{B}} \sum_{\mathbf{z}} p(\mathbf{z}) \log p_\theta(\mathbf{z}|\bar{\mathbf{x}}^{(i)})$$

   (c) Let the mean of the penultimate layer of the encoder for the fake batch be $E_{\bar{B}}$ and for the real batch be E_B.
   (d) Train the generator to minimize the squared distance between the average encodings for the real and fake batch for the penultimate layer.
8. If negative sampling is enabled:
   (a) Create a batch of fake documents $\bar{B}$ by sampling words in the vocabulary based on their frequency with which they occur in the corpus.
   (b) Train the encoding network to maximize

$$\frac{1}{\bar{B}} \sum_{\mathbf{x} \in \bar{B}} \sum_{\mathbf{z}} p(\mathbf{z}) \log p_\theta(\mathbf{z}|\bar{\mathbf{x}}^{(i)})$$

Table 4.1: Network architecture for MNIST

Generator	Encoder
256 ConvT 4x4, relu	64 Conv 4x4, stride 2, leaky relu
128 ConvT 3x3, stride 2, bn, relu	128 Conv 4x4, stride 2, bn, lrelu
—	128 Conv 3x3, stride 1, bn, lrelu
64 ConvT 4x4, stride 2, bn, relu	256 Conv 3x3, stride 2, bn, lrelu
—	256 Conv 3x3, stride 1, bn, lrelu
1 ConvT 4x4, stride 2, sigmoid	nc Conv 4x4, stride 1, softmax

*ConvT=transposed convolution
*relu = rectified linear units
*lrelu=leaky rectified linear units
*bn=batch normalization
*nc=number of clusters

Table 4.2: Clustering error on MNIST dataset for various models

Model	Percentage error
DEC (10 clusters) (Xie et al., 2016)	15.6
CatGAN (20 clusters) (Springenberg, 2015)	4.27
Adversarial autoencoder (32 clusters) (Makhzani et al., 2015)	4.10±1.13
DisCoder (unregularized)	24.3±10.6
DisCoder (20 clusters)	**3.12±0.93**

the initial assignment.

The DisCoder achieves the state-of-the-art results for clustering on MNIST, significantly outperforming other methods. Since the generator is trained along with the encoder, we can visualize the images associated with each latent feature by setting the corresponding component of $\mathbf{z}$ to 1, concatenating it with noise and passing it through the generator. The resultant images are shown in Figure 4.4. As expected, each latent feature has learned to associate itself with images of a single class.

4.5.2 20 newsgroup

In order to ensure reproducibility, we use a preprocessed version of the dataset[1]. We use tf-idf representation for the documents. We train a DisCoder with negative sampling for clustering on this dataset as detailed in Section 4.4. The encoding network consists of a single hidden

[1]The pre-processed dataset is available at https://sites.google.com/site/renatocorrea02/textcategorizationdatasets

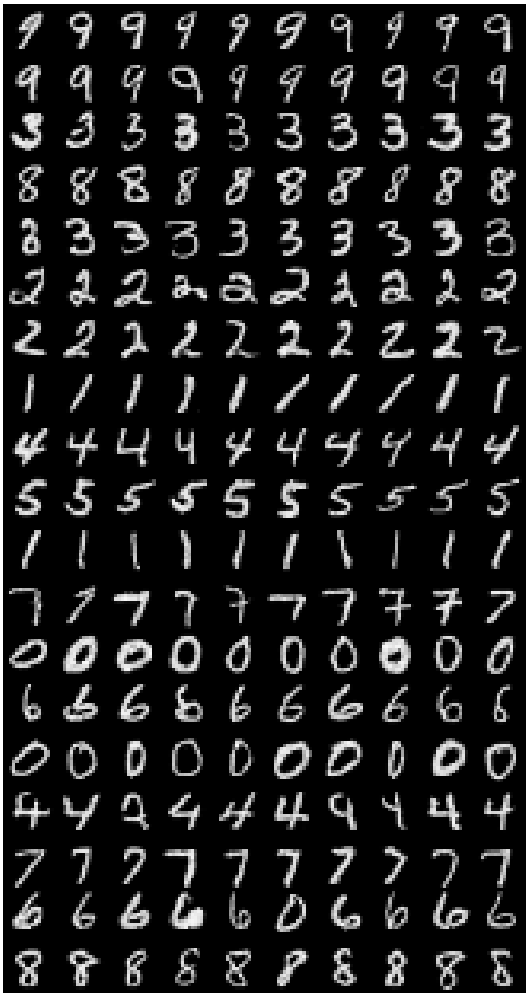


Figure 4.4: Generated samples from the unsupervised MNIST generator conditioned on $\mathbf{z}$. Each row corresponds to a different component of $\mathbf{z}$ set to 1. Note that despite the fact that training was done in an unsupervised fashion, the generator was successfully able to associate different clusters with different classes.

layer with 1000 units. In order to compute clustering accuracy, we use the same strategy that we had used for MNIST.

The clustering accuracy achieved by the various models on 20 newsgroup dataset is shown in Table 4.3. For LSA and LDA, we set the number of topics to the number of clusters. We assign each document to the topic with the highest probability for the given document. Again, as in the case of MNIST, a regularized DisCoder outperforms all the other models.

Table 4.3: Clustering accuracy on 20 newsgroup dataset for various models

Model	Percentage accuracy on 20 newsgroup
K-means	40.4
LSA (Landauer et al., 1998)	57.25
LDA (Blei et al., 2003)	55.6
DisCoder (unregularized)	35.11±8.13
DisCoder	**61.43±2.24**

4.5.3 CIFAR-10

Next, we evaluate the performance of the DisCoder on semi-supervised learning task for CIFAR-10 dataset. CIFAR-10 is a relatively complicated image dataset with 50000 training samples belonging to 10 classes. There is high variability within each class, making it almost impossible to perform clustering on this dataset. Hence, we use the dataset for semi-supervised classification. We normalize the images to lie between -1 and 1. We use 400 examples per class as labeled data, and use the rest of the images as unlabeled.

The architecture of the generator and the encoding network used for this task is given in Table 4.4. The architecture of the encoding network is essentially the same as the architecture of discriminator in Improved GAN (Salimans et al., 2016). In particular, the magnitude of weights in all the convolution layers of the encoding network are held fixed during training, and leaky rectified non-linearities with a slope of .2 follow every convolution layer. Batch normalization is used in the layers of the generator. Moreover, to allow head-to-head comparison with Improved GAN, we use feature matching to generate the fake images for our experiments on semi-supervised learning. For unlabeled images, the class labels are treated as latent features. Hence, DisCoder and Improved GAN only differ in the mechanism used for training the encoding network/discriminator for semi-supervised classification.

The classification error achieved by the various models on CIFAR-10 is shown in Table 4.5. DisCoder achieves the lowest classification error among all the models, significantly outperforming both Improved GAN and CatGAN. We use class-conditioned generation to visualize samples associated with each class learnt by the generator. To generate each row, we set the corresponding component of $\mathbf{z}$ to 1, concatenate it with noise, and pass it through the generator. The corresponding images for the 10 classes are shown in Figure 4.5. One can observe that for many classes, the generator is able to capture the shape of the objects quite effectively, which has been known to be quite challenging for CIFAR-10. For comparison, we have shown

Table 4.4: Network architecture for CIFAR-10

Generator	Encoder
512 ConvT 4x4, relu	dropout(.2)
—	96 Conv 3x3, stride 1, wn, lrelu
—	96 Conv 3x3, stride 1, wn, lrelu
—	96 Conv 3x3, stride 2, wn, lrelu
256 ConvT 4x4, stride 2, bn, relu	dropout(.5)
—	192 Conv 3x3, stride 1, wn, lrelu
—	192 Conv 3x3, stride 1, wn, lrelu
—	192 Conv 3x3, stride 2, wn, lrelu
128 ConvT 4x4, stride 2, bn, relu	dropout(.5)
—	192 Conv 3x3, stride 1, wn, lrelu
—	192 Conv 1x1, stride 1, wn, lrelu
—	192 Conv 1x1, stride 1, wn, lrelu
—	Global average pooling
3 ConvT 4x4, stride 2, tanh	10 linear,wn, softmax

*wn= weight normalization

the images generated by feature matching when trained on CIFAR-10 for the same task. As one can observed, most of the images are meaningless blobs.

Finally, note that any improvement in the architecture of neural networks or the loss function used for training the generator can be incorporated in the DisCoder, since these hyperparameters are complementary to the model definition of a DisCoder.

4.5.4 SVHN

We also evaluate the performance of the DisCoder for semi-supervised learning on the Street View House Numbers (SVHN) dataset (Netzer et al., 2011). The dataset consists of cropped images of digits extracted from house numbers in Google Street View images. As is the case with CIFAR-10, the dataset has high variability, and hence, requires a few labeled examples per class to achieve satisfactory classification result. We use 100 examples per class as labeled images, while the rest of the images are used as unlabeled images.

The architecture of the generator and the encoding network used for this task is given in Table 4.6. Again, the architecture of the encoding network is essentially the same as the architecture of discriminator in Improved GAN (Salimans et al., 2016). Feature matching

Table 4.5: Performance of various models on CIFAR-10 dataset for the task of semi-supervised classification.

Model	Percentage error using 4000 labeled samples
Ladder network (Rasmus et al., 2015)	20.4±0.47
CatGAN (Springenberg, 2015)	19.58±0.46
Improved GAN (Salimans et al., 2016)	18.63±2.32
DisCoder (unregularized)	31.33±4.4
DisCoder	**17.24±0.43**

is used to generate the fake images for our experiments on semi-supervised learning. The classification error achieved by the various models on SVHN is shown in Table 4.7. As can be observed, regularized DisCoder outperforms Improved GAN by a significant margin, despite using the same network architecture. The samples generated using feature matching are given in Figure 4.6.

4.5.5 Discovering new classes

Till now, we have considered experiments in which the labeled data and unlabeled data have been sampled from the same class. In practice, it is quite common to have access to a large labeled dataset (eg. ImageNet), whereas the prediction needs to be done on a much smaller set of examples whose labels do not occur in the large labeled dataset. A standard approach to deal with such problems is to pretrain the network using the large labeled dataset, throw away the final prediction layer, and finetune the network on the smaller set of examples. However, if the set of interest does not have any labeled examples, this approach can not be used.

If the intra-cluster variability in the unlabeled dataset mimic the intra-class variability of the large labeled dataset, it should be theoretically possible to utilize the labeled classes to cluster the unlabeled data. This is similar to how humans perceive images of objects. If we observe several images of two distinct objects that we have not observed before, we will be able to distinguish between the two sets of images, based on our knowledge of variability within images of objects of the same class.

To test this hypothesis, we use 4000 unlabeled images from the first two classes of CIFAR-10 and 4000 labeled images from the remaining classes. Note that we do not use any labeled images for the first two classes. We wish to cluster the first two classes using the remaining classes of CIFAR-10. We use the same network architecture and hyperparameters, that we used for semi-supervised learning on CIFAR-10. The training procedure remains exactly the same

Table 4.6: Network architecture for SVHN

Generator	Encoder
512 ConvT 4x4, relu	dropout(.2)
—	64 Conv 3x3, stride 1, wn, lrelu
—	64 Conv 3x3, stride 1, wn, lrelu
—	64 Conv 3x3, stride 2, wn, lrelu
256 ConvT 4x4, stride 2, bn, relu	dropout(.5)
—	128 Conv 3x3, stride 1, wn, lrelu
—	128 Conv 3x3, stride 1, wn, lrelu
—	128 Conv 3x3, stride 2, wn, lrelu
128 ConvT 4x4, stride 2, bn, relu	dropout(.5)
—	128 Conv 3x3, stride 1, wn, lrelu
—	128 Conv 1x1, stride 1, wn, lrelu
—	128 Conv 1x1, stride 1, wn, lrelu
—	Global average pooling
3 ConvT 4x4, stride 2, tanh	10 linear,wn, softmax

*wn= weight normalization

Table 4.7: Performance of various models on SVHN dataset for the task of semi-supervised classification.

Model	Percentage error using 1000 labeled samples
SDGM (Maaløe et al., 2016)	16.61±0.24
ADGM (Maaløe et al., 2016)	22.86
Improved GAN (Salimans et al., 2016)	8.11±1.13
DisCoder (unregularized)	16.33±2.42
DisCoder	**5.22±0.12**

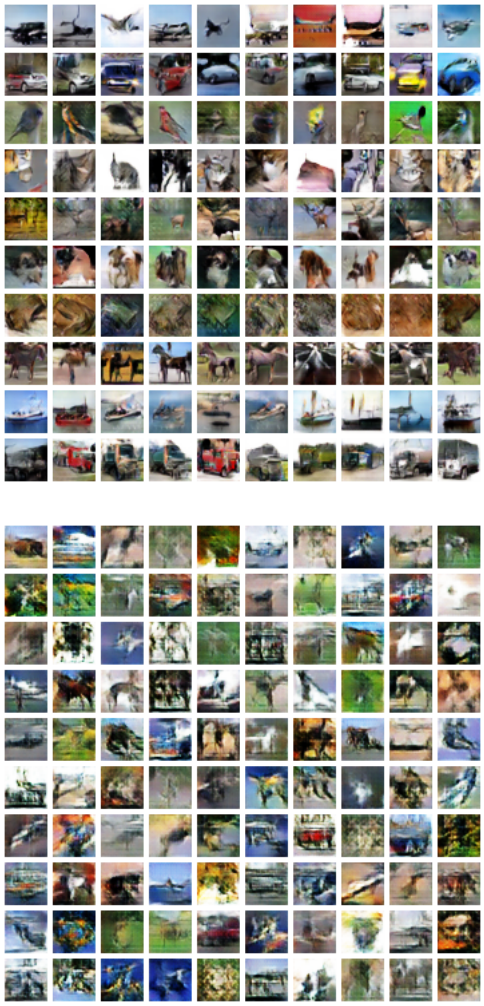


Figure 4.5: (Top) Generated samples from the semi-supervised CIFAR-10 generator using class-conditioned generation. Each row corresponds to a different component of $\mathbf{z}$ set to 1. Since this problem is semi-supervised, $\mathbf{z}$ is not exactly a latent feature. Note that for many classes, the generator is able to preserve the shape of the objects, which is known to be quite challenging for CIFAR-10 images. (Bottom) Samples generated by the generator using feature matching for the same task. Note that the shape of the objects is not preserved in the images.

as the procedure used for semi-supervised learning. In particular, for labeled data, the model is trained to predict the correct class. For unlabeled data, the model is trained using the training procedure for DisCoder.

We use the purity metric for evaluating the clusters. Let $c_1, \ldots, c_m$ be the true classes and $o_1, \ldots, o_n$ be the clusters predicted by a model. The purity of the clusters $o_1, \ldots, o_n$ is define as follows:

$$\text{purity} = \frac{\sum_{j=1}^{n} \max_{1 \leq i \leq n} |c_i \cap o_j|}{\sum_{j=1}^{n} |o_j|},$$

Figure 4.6: Samples generated by the generator using feature matching for semi-supervised learning on SVHN.

where $|A|$ is the number of points in the set A. We achieve a purity of 97.4% on clustering the first two classes of CIFAR-10 dataset. In contrast, if we attempt to cluster the two classes without access to any labeled data, we achieve an accuracy of 61.7%, which is better than random chance (50%), but considerably worse than the result obtained using the proposed model.

An alternative but more obvious strategy is to train the network using the labeled data, throw away the last prediction layer, get the feature representation from the last layer of the network for unlabeled data and apply a clustering algorithm on these representations. We use nearest neighbor spectral clustering to cluster the representations of unlabeled data into 2 clusters. Using this strategy, we achieve an accuracy of 88.5%. The results for spectral clustering using other kernels were considerably worse. The results are summarized in Table 4.8.

There are a few observations that can be made from the table. Firstly, clustering general images without any access to labeled data is extremely challenging. This is obvious, since there are several factors of variations (such as color, intensity etc.) that can be used for clustering the images. Secondly, the features learnt by a classifier on a large amount of labeled data, generalize well for discriminating the classes not observed in the training data. Finally, the best performance on unlabeled data is achieved when the labeled classes as well as unlabeled classes are used for training all the layers of the network. To our knowledge, this is the first result for clustering on the classes of CIFAR-10.

Table 4.8: Clustering error on the first two classes of CIFAR-10 using various levels of supervision. The classes used for training are indicated in paranthesis.

Data used for training the network	(1-Purity)%
Unlabeled data only (0-1)	39.3%
Labeled data only (2-9)	11.5%
Labeled + Unlabeled data	**2.6**%

4.5.6 Domain Adaptation and Transfer Learning

Domain adaptation and transfer learning are used when the data distribution during test differs from the data distribution during training. Let $\mathbf{x}$ be the input and $\mathbf{y}$ be the output, $p(\mathbf{x})$ be the distribution of the input and $p(\mathbf{y}|\mathbf{x})$ be the distribution of the output given the input.

Domain adaptation: Here $p(\mathbf{y}|\mathbf{x})$ remains the same during training as well as testing. However, the distribution of the input $p(x)$ changes. As an example, one can think of sentiment classification in Amazon reviews, whereby the training data consists of reviews about refrigerators whereas the test data consists of reviews about smartphones. The task is to identify the reviews about smartphones as positive or negative based on the labeled reviews about refrigerators.

Transfer learning: In contrast to domain adaptation, the definition of transfer learning is rather broad and includes all models and methods that transfer knowledge from one dataset to other. Hence, domain adaptation is a subset of transfer learning. Perhaps the most popular transfer learning approach is pretraining on a large labeled generic dataset and finetuning on a much smaller labeled task-specific dataset. For instance, in Chapter 6, we pretrain a VGG network on ImageNet with 1000 classes, transfer the weights to a modified fully-convolutional network for semantic segmentation, and finetune the modified network for weakly supervised semantic segmentation.

In this section, we want to study the effectiveness of transfer learning for semi-supervised learning tasks. Instead of initializing the network with random weights, we use the weights of networks pretrained on ImageNet for initialization. We use the network for semi-supervised learning on CIFAR-10 dataset using a VGG-16 model pretrained on ImageNet dataset. We discard the last classification layer and finetune the last two fully connected layers on the CIFAR-10 dataset as shown in the Figure 4.7. The rest of the layers are kept fixed during finetuning. Since CIFAR-10 images are of very low resolution (32x32), we use bicubic interpolation to upsample the images to 224x224 before feeding them to the VGG-16 network.

Using 4000 labeled samples as discussed in Section 4.5.3, we achieve an error rate of 15.23%,

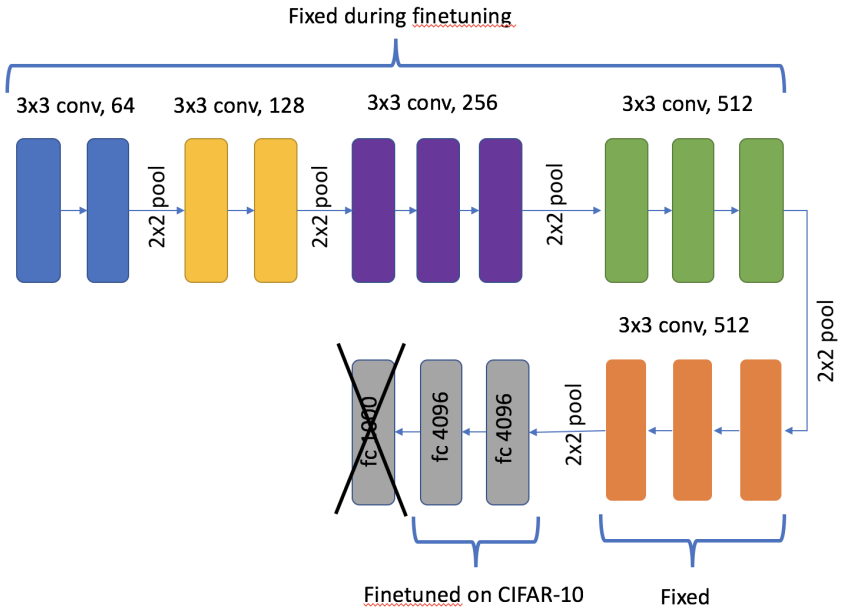


Figure 4.7: The VGG Network finetuned on CIFAR-10 dataset. Only the last two fully connected layers are finetuned.

which is much lower than the reported performance for semi-supervised learning on CIFAR-10. However, this result is not directly comparable, since we make use of a large dataset (ImageNet) for pretraining the network.

4.6 Discussion

In this chapter, we introduced a discriminative method for encoding the data in an unsupervised manner. We evaluated the categorical embeddings learnt by the model for unsupervised and semi-supervised classification tasks and achieved improved performance for most of the datasets. Furthermore, we used DisCoder to successfully discover unobserved classes on CIFAR-10 dataset. From the above experiments, it is quite clear that the categorical latent features learnt by a DisCoder are indeed meaningful.

Over the past few years, several approaches for unsupervised and semi-supervised classification with deep networks have been proposed. One of the first works in deep semi-supervised classification (Kingma et al., 2014) used variational autoencoders (Kingma and Welling, 2013) to learn hybrid (categorical+real) embeddings for data. In order to ensure that the categorical embeddings correspond to class labels, the network that maps data to categorical embeddings, is finetuned using labeled data. An extension of the model uses auxiliary variables to achieve improved performance for semi-supervised learning tasks (Maaløe et al., 2016).

Variational autoencoders attempt to learn embeddings from which the data can be reconstructed back using a decoder. In contrast, another set of models learn categorical embeddings without enforcing the constraints imposed by a decoder on the embeddings (Springenberg, 2015, Salimans et al., 2016). These models are based on the framework of generative adversarial networks (GAN) (Goodfellow et al., 2014). In a GAN, a discriminator is trained to distinguish between generated and real samples, while a generator is simultaneously trained to generate images from a latent distribution that fool the discriminator.

In particular, CatGANs (Springenberg, 2015) learn a categorical latent representation for the data by maximizing the mutual information between the data and the categorical labels, while simultaneously maximizing the entropy over the class labels for generated data. Improved GANs (Salimans et al., 2016) train a classifier to map the data to $m + 1$ labels, where m is the number of classes present in the data. The classifier is trained to predict the correct class for labeled data, while any of the m classes can be predicted for unlabeled data. Furthermore, the classifier is trained to predict the $(m + 1)^{st}$ class for generated data, while the generator is simultaneously trained to generate images that have the same features as real data. GAN based approaches consistently achieve superior results as compared to autoencoder based models. However, while autoencoder based models have a simple probabilistic interpretation in terms of maximization of log-likelihood, CatGANs and improved GANs are based on the heuristic that the data is sparsely distributed at the boundaries between classes.

For categorical latent features, DisCoder is similar in spirit to CatGANs and Improved GANs. However, DisCoder allows a straightforward interpretation in terms of maximization of log-likelihood of a certain distribution. Hence, several interesting models can be used as priors over the latent features of a DisCoder. In this chapter, we have focused on priors on latent features that have a single level of hierarchy. In the next chapter, we couple the DisCoders with hierarchical Bayesian priors for modeling the latent representation of images.

Chapter 5

Discriminative Encoding with Hierarchical Bayesian Priors

5.1 Introduction

In the previous chapter, we proposed discriminative encoders (DisCoder) for incorporating unlabeled data in discriminative models. In the experiments, we focused on treating the categorical latent features as cluster labels. With the incorporation of adversarial regularization, the proposed model achieved state of the art performance for clustering and related tasks on several interesting datasets.

Categorical representations are limited, since they represent an object using a one-hot representation. It is more logical to assume real valued representations for objects, where objects with similar abstract features, lie close in the latent space. Unfortunately, DisCoders with flat priors such as Gaussian or uniform distribution (in a compact space) attempt to force the representation of all the objects to be apart from each other. In general, it is more reasonable to assume that some of the objects are more similar to each other than others, and hence, their representations must lie closer to each other. For instance, the sentences within a document are more likely to be similar, than two sentences from two randomly chosen documents.

To incorporate these relationships, we use hierarchical Bayesian priors on the real valued latent representations. In particular, we focus on Gaussian mixture models and Gaussian topic models as priors on the latent features to allow for grouping among the representations. This work represents the first step in an intermarriage between deep learning and hierarchical Bayesian models for unsupervised learning. In particular, we use a discriminative encoder to extract latent representations from input, while simultaneously using Gaussian mixture and Gaussian topic priors to model the same latent features. This allows us to perform mixture

modeling and topic modeling in the representation space rather than the input space. Furthermore, unlike generative models, we do not need to worry about the parametric form of the input distribution at all.

Before, we describe the model, let us quickly review the main properties of discriminative encoders:

- DisCoders are completely specified by specifying the encoding distribution $p_\theta(\mathbf{z}|\mathbf{x})$ and the prior distribution on the latent features $p_\theta(\mathbf{z})$. Here θ encodes all the parameters of the distributions.

- The joint distribution of a DisCoder over the observed and the latent features is given by

$$q_\theta(\mathbf{x}, \mathbf{z}) = q_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})\,.$$

  Here, the distribution $q(\mathbf{x}|\mathbf{z})$ is computed from the encoding distribution as follows:

$$q_\theta(\mathbf{x}|\mathbf{z}) = \frac{p_\theta(\mathbf{z}|\mathbf{x})}{\sum_{i=1}^{n} p_\theta(\mathbf{z}|\mathbf{x}^{(i)})}\,,$$

  where $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ are the observed training samples.

- Discoders do not attempt to model the true distribution of the observed features $\mathbf{x}$. In fact, they explicitly assign 0 probability to any data point not observed in the training set.

- The marginal distribution of a DisCoder over the latent features is given by $p_\theta(\mathbf{z})$.

5.2 Mixture modeling

In this section, we consider the scenario, where the prior over the latent features is given by a mixture of Gaussians. We briefly review Gaussian mixture models in a Bayesian framework, before defining them as a prior for discriminative encoders.

5.2.1 Gaussian mixture models

Let us assume that we are interested in clustering $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}$ into K clusters. Hence, for this section $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}$ are the input. Let π be the mixing proportion for the clusters $1, \ldots, K$. Let $c^{(i)}$ be an indicator variable indicating the cluster that $\mathbf{z}^{(i)}$ belongs to. We will refer to $c^{(i)}$ as

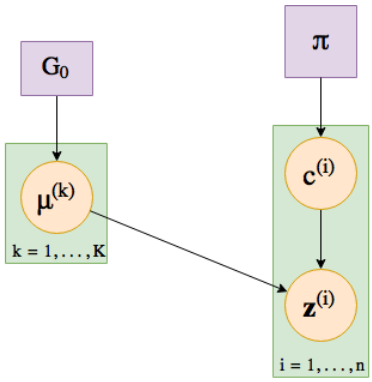


Figure 5.1: A graphical representation of the GMM model. Square nodes indicate deterministic variables, whereas circular nodes indicate random variables. Circles in a rectangle indicate that the variable must be sampled multiple times.

cluster assignment variable. Furthermore $Cat(.)$ denotes the categorical distribution, whereas $Dir(.)$ denotes the Dirichlet distribution. A Gaussian mixture model for sampling $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}$ can be represented as follows:

$$\begin{aligned} c^{(i)} &\sim Cat(\pi)\,, \\ \mathbf{z}^{(i)}|c^{(i)} &\sim \mathcal{N}(\mu_{c^{(i)}}, \Sigma_{c^{(i)}})\,. \end{aligned}$$

We assume that $\Sigma_k = \rho^{-1}I$ for all the clusters. Here, ρ is a hyperparameter, that is kept fixed during training. If we further define a prior G_0 on the means of the clusters, the new model can be represented as given below:

$$\begin{aligned} \mu_k &\sim G_0\,, \\ c^{(i)} &\sim Cat(\pi)\,, \\ \mathbf{z}^{(i)}|c^{(i)}, \{\mu_k\}_{k=1}^K &\sim \mathcal{N}(\mu_{c^{(i)}}, \rho^{-1}I)\,. \end{aligned}$$

The corresponding graphical model representation is given in Figure 5.1. For brevity, we denote the latent features $\{\mathbf{z}^{(i)}\}_{i=1}^n$ as $\{\mathbf{z}^{(i)}\}$, the means of the clusters $\{\mu_k\}_{k=1}^K$ as $\{\mu_k\}$, and the cluster indicator variables $\{c^{(i)}\}_{i=1}^n$ as $\{c^{(i)}\}$. The natural choice for G_0 is a normal distribution. In particular, we assume that G_0 is a normal distribution with mean 0 and fixed covariance matrix $\rho_0^{-1}I$. The joint distribution of the input, the cluster assignment variables, the parameters of

clusters, and the mixing proportions is given by

$$p(\{\mathbf{z}^{(i)}\}, \{c^{(i)}\}, \{\mu_k\}) = \left(\prod_{i=1}^{n} p(\mathbf{z}^{(i)}|c^{(i)}, \{\mu_k\})p(c^{(i)})\right)\left(\prod_{k=1}^{K} p(\mu_k)\right) . \tag{5.1}$$

Given the input $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}$, the aim is to infer the cluster assignment variables $c^{(i)}$ for each $\mathbf{z}^{(i)}$, and the parameters of each cluster. Conjugacy among the distributions allows for efficient inference using Gibbs sampling. Inference in a GMM alternates between the inference of the cluster assignment variables $c^{(i)}$, and the parameters of the clusters μ_k.

Inferring the cluster assignment variables $c^{(i)}$: As can be observed from the graphical representation of the model in Figure 5.1, the cluster assignment variables are independent random variables. Hence,

$$p(\{c^{(i)}\}) = \prod_{i=1}^{n} p(c^{(i)}) .$$

Hence, the posterior distribution is proportional to the product of the prior and likelihood.

$$p(c^{(i)} = k|\mathbf{z}^{(i)}, \{\mu_k\}) \propto p(c^{(i)})p\left(\mathbf{z}^{(i)}|c^{(i)} = k, \mu_k\right) \tag{5.2}$$

$$= \pi_k \mathcal{N}(\mathbf{z}^{(i)}|\mu_k, \rho^{-1}I) , . \tag{5.3}$$

Inferring the parameters of clusters μ_k: Given the cluster assignment variables, the inference scheme for the parameters of the clusters is quite straightforward. The following observations can be made from (5.1).

1. The means of different clusters are independent random variables.

2. Given the cluster assignment variables, the mean of cluster k depends only on the features that have been assigned to cluster k.

Hence,

$$p(\mu_k|\{c^{(i)}\}, \{\mathbf{z}^{(i)}\}) \propto G_0(\mu_k) \prod_{c^{(i)}=k} p\left(\mathbf{z}^{(i)}|c^{(i)} = k, \mu_k\right) \tag{5.4}$$

$$= \mathcal{N}(\mu_k|0, \rho_0^{-1}I) \prod_{c^{(i)}=k} \mathcal{N}(\mathbf{z}^{(i)}|\mu_k, \rho^{-1}I) . \tag{5.5}$$

The conjugacy of the normal distribution implies that the posterior distribution is also a normal distribution. We can compute the mean and variance of the posterior distribution by completing

the quadratic in the exponent. In particular, for fixed x, the negative exponent of Gaussian distribution with mean m and precision matrix S has the following form (The precision matrix is the inverse of the covariance matrix):

$$\frac{1}{2}x^{\mathrm{T}}Sx + \frac{1}{2}m^{\mathrm{T}}Sm - x^{\mathrm{T}}Sm\,.$$

In contrast, the negative exponent of the posterior distribution for fixed x has the following form (We have replaced μ_k from the posterior by x to allow comparison):

$$\begin{aligned}&\sum_{c^{(i)}=k} \rho\frac{||\mathbf{z}^{(i)} - x||^2}{2} + \rho_0\frac{||x||^2}{2} + constant\\ &= \frac{||x||^2}{2}\left(n_k\rho + \rho_0\right) - n_k\rho x^{\mathrm{T}}\left(\frac{\sum_{c^{(i)}=k}\mathbf{z}^{(i)}}{n_k}\right) + constant\end{aligned}$$

Equating the coefficeints of quadratic and linear terms of x, we get

$$\begin{aligned}S &= (n_k\rho + \rho_0)I\,,\\ m &= \frac{n_k\rho}{n_k\rho + \rho_0}\left(\frac{\sum_{c^{(i)}=k}\mathbf{z}^{(i)}}{n_k}\right)\,.\end{aligned}$$

The posterior distribution of the mean of k^{th} cluster is given by

$$p\left(\mu_k|\{c^{(i)}\},\{\mathbf{z}^{(i)}\}\right) = \mathcal{N}(m, S^{-1})\,.$$

Algorithm 4 lists the steps involved in inference in Gaussian mixture models.

5.2.2 Discriminative encoding with Gaussian mixture models

We use the Gaussian mixture model defined above as a prior $p_\theta(\mathbf{z})$ on the real-valued latent features of a discriminative encoder. Given the latent features $\mathbf{z}^{(i)}, 1 \leq i \leq n$, it is straightforward to run Algorithm 4 on the features to obtain the parameters of each cluster. Hence, in order to complete the description, we only need to describe the mechanism for obtaining the latent features $\mathbf{z}$ from the observed features $\mathbf{x}$.

In a DisCoder, the joint log-likelihood of the observed features $\mathbf{x}^{(i)}$ and the latent features $\mathbf{z}^{(i)}$ is given by

$$\log q_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) = \log p_\theta(\mathbf{z}^{(i)}|\mathbf{x}^{(i)}) - \log\sum_{j=1}^{n} p_\theta(\mathbf{z}^{(i)}|\mathbf{x}^{(j)}) + \log p_\theta(\mathbf{z}^{(i)})\,. \tag{5.7}$$

Algorithm 4: Gibbs inference in Gaussian mixture models.

Input: Input $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}$, hyperparameters ρ and ρ_0
Output: The parameters of the clusters $\{\mu_k\}$ and the assignment variables $\{c^{(i)}\}$
Initialization: Randomly assign the points to clusters to get an initial cluster assignment.

1 Repeat the following steps until the Markov chain has mixed thoroughly.

1. Repeat for $k = 1, \ldots, K$.

   (a) Compute the number of points, and the mean of each cluster as follows:

$$n_k = \sum_{c^{(i)}=k} 1$$
$$\bar{\mathbf{z}}_k = \frac{\sum_{c^{(i)}=k} \mathbf{z}^{(i)}}{n_k}$$

   (b) Compute the following quantity:

$$\bar{\mu}_k = \frac{n_k \rho}{\rho + n_k} \bar{\mathbf{z}}$$
$$S_k = (n_k \rho + \rho_0) I$$

   (c) Sample $\mu_k \sim \mathcal{N}(\bar{\mu}_k, S_k)$.

2. Repeat for $i = 1, \ldots, n$.

   (a) Sample $c^{(i)}$ from the following distribution

$$p\left(c^{(i)} = k | \mathbf{z}^{(i)}, \{\mu_k\}\right) \propto \pi_k \mathcal{N}(\mathbf{z}^{(i)} | \mu_k, \rho^{-1} I)\,. \tag{5.6}$$

The prior distribution on the latent features is a Gaussian mixture model. We condition the prior distribution on the cluster assignment variable, $c^{(i)}$, and the parameters of the clusters $\{\mu_k\}$. The resultant prior distribution is Gaussian with mean $\mu_{c^{(i)}}$ and covariance $\rho^{-1}I$. Furthermore, the encoding distribution $p_\theta(\mathbf{z}^{(i)}|\mathbf{x}^{(i)})$ is assumed to be a Gaussian with a fixed covariance $\lambda^{-1}I$. Hence, the negative log-likelihood of the observed and the latent variables conditioned on $c^{(i)}$ and $\{\mu_k\}$ is given by

$$\frac{\lambda}{2}||\phi(\mathbf{x}^{(i)}) - \mathbf{z}^{(i)}||^2 + \log \sum_{j=1}^{n} \exp\left(-\frac{\lambda}{2}||\phi(\mathbf{x}^{(j)}) - \mathbf{z}^{(i)}||^2\right) + \frac{\rho}{2}||\mathbf{z}^{(i)} - \mu_{c^{(i)}}||^2 \tag{5.8}$$

One needs to optimize the above equation with respect to the latent features $\mathbf{z}^{(i)}$ and the parameters of the network ϕ. Unfortunately, this involves an optimization within an optimization problem and hence, it is quite expensive. Hence, as discussed in Section 4.3.2, we equate $\mathbf{z}^{(i)}$ to $\phi(\mathbf{x}^{(i)})$ to get the following simplified objective.

$$\log \sum_{j=1}^{n} \exp\left(-\frac{\lambda}{2}||\phi(\mathbf{x}^{(j)}) - \phi(\mathbf{x}^{(i)})||^2\right) + \frac{\rho}{2}||\phi(\mathbf{x}^{(i)}) - \mu_{c^{(i)}}||^2 . \tag{5.9}$$

The above equation is optimized with respect to the parameters of the neural network ϕ. After the network has been trained for one epoch, we equate the latent features $\mathbf{z}^{(i)}$ to $\phi(\mathbf{x}^{(i)})$ and perform Gaussian mixture modeling on these latent features. The learnt cluster assignment variables and the parameters of the clusters are then used for training the network as discussed above. This process of clustering and training the network alternatively is repeated for a fixed number of iterations T. The steps involved in learning latent features for Gaussian mixture modeling are summarized below:

1. Forward propagate the observed features $\{\mathbf{x}^{(i)}\}$ through the network to get the latent features $\{\mathbf{z}^{(i)}\}$.

2. Repeat for $t = 1, \ldots, T$

   a) Run Algorithm 1 on the latent features to obtain the cluster assignment variables $\{c^{(i)}\}$ for all the latent features, and the parameters of the clusters $\{\mu_k\}$.

   b) Train the neural network ϕ by minimizing the objective below for the entire dataset.

$$\log \sum_{j=1}^{n} \exp\left(-\frac{\lambda}{2}||\phi(\mathbf{x}^{(j)}) - \phi(\mathbf{x}^{(i)})||^2\right) + \frac{\rho}{2}||\phi(\mathbf{x}^{(i)}) - \mu_{c^{(i)}}||^2 .$$

c) Equate the latent features to the output of the network, that is, $\mathbf{z}^{(i)} = \phi(\mathbf{x}^{(i)})$, $1 \leq i \leq n$.

The corresponding model is referred to as GMM-DisCoder.

5.2.3 Optimization details

In general, it is too expensive to run Gibbs inference in every epoch of training. The Markov chains may require a lot of iterations to mix thoroughly. Hence, instead of randomly sampling the clustering assignment variables, and the parameters of each cluster from their respective posterior distributions given $\{\mathbf{z}^{(i)}\}$, we equate the variables to the mode of the respective posterior distributions. Hence, the step 1(c) of Algorithm 4 is replaced by $\mu_k = \bar{\mu}_k$. Moreover, instead of sampling the cluster assignment variables, we perform a hard assignment similar to k-means. That is, we replace the step 2(a) of Algorithm 4 by

$$c^{(i)} = \underset{k}{\operatorname{argmax}}\, \pi_k \mathcal{N}(\mathbf{z}^{(i)} | \mu_k, \rho^{-1} I) \,.$$

Finally, rather than initializing the cluster assignment variables $\{c^{(i)}\}$ randomly in every inference step, we reuse the cluster assignment variables obtained in the previous inference step. These three approximations speed up the inference step drastically. For fixed latent features, we run the approximated inference step for 5 iterations over the entire dataset. Subsequently, the network is trained for 1 epoch to obtain the latent features. These two steps are executed alternatively for a fixed number of iterations.

5.3 Topic modeling

Most of the data that we encounter in daily life has some underlying structure. For instance, the words are organized into sentences, which are organized into documents. Images and documents are often organized into sub-folders and folders. Mixture models completely ignore the presence of these observed structures among the points in a dataset. However, these structures can provide implicit supervisory information, that may prove to be useful for learning.

Topic models form the prime example of models that exploit the observed structure present in data for secondary tasks. For instance, they exploit the representation of documents as bag of words to identify the topics associated with each document (Landauer and Dumais, 1997, Hofmann, 1999, Blei et al., 2003). Similarly, they exploit the representation of images as bag of visual words to identify the themes associated with each image (Fei-Fei and Perona, 2005). They can further be used for exploiting the organization of images in albums for identifying the themes associated with a specific album. To achieve this, one needs to learn a representation

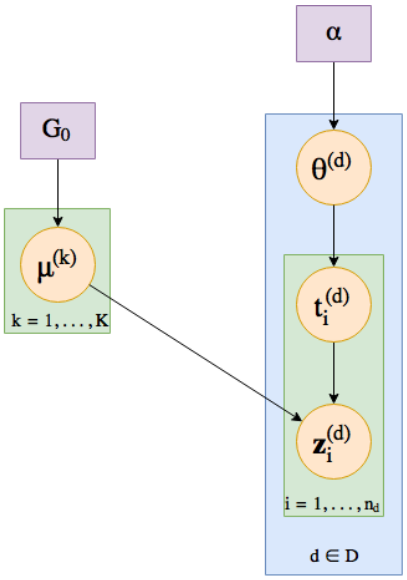


Figure 5.2: A graphical representation of the Gaussian topic model. Square nodes indicate deterministic variables, whereas circular nodes indicate random variables. Circles in a rectangle indicate that the variable must be sampled multiple times.

for each image and document, and subsequently apply topic modeling on these representations. In this section, we briefly review Gaussian topic models (Das et al., 2015) before using them as prior over the representations.

5.3.1 Gaussian Topic Models

Consider the problem of sampling the words in a document d, which forms a part of a corpus D. The document has n_d words. Furthermore, the entire corpus is assumed to contain K latent topics, where K is a hyperparameter. The generative process of a Gaussian topic model with Dirichlet prior (Das et al., 2015) is as follows:

1. For k=1 to K
   (a) Sample the parameters of the k^{th} topic $\mu_k \sim G_0$.
2. For each document d in corpus D
   (a) Sample a distribution over the topics for the document $\theta^{(d)} \sim Dir(\alpha)$.
   (b) For each word index i in the document d:
      (i) Sample a topic for the word index $t_i^{(d)} \sim Cat(\theta^{(d)})$.
      (ii) Sample a word from the topic distribution $\mathbf{z}_i^{(d)} \sim \mathcal{N}(\mu_{t_i^{(d)}}, \rho^{-1}I)$.

Here, we choose G_0 to be a normal distribution with 0 mean and fixed covariance matrix $\rho_0^{-1}I$. As before, $Cat(.)$ refers to the categorical distribution, whereas $Dir(.)$ refers to the Dirichlet distribution. The hyperparameter α governs the sparsity of topics in a given document. The joint distribution of the mixing proportions $\{\theta^{(d)}\}$, the topic assignment variables $\{t_i^{(d)}\}$, the words $\{\mathbf{z}_i^{(d)}\}$ and the parameters of the topics $\{\mu_k\}$ is given as follows:

$$p(\{\theta^{(d)}\},\{t_i^{(d)}\},\{\mathbf{z}_i^{(d)}\},\{\mu_k\}) = \left(\prod_{d\in D} p(\{\theta^{(d)}\})\prod_{i=1}^{n_d} p(\{t_i^{(d)}\}|\{\theta^{(d)}\})p(\{\mathbf{z}_i^{(d)}\}|\{t_i^{(d)}\},\{\mu_k\})\right) \times \left(\prod_{k=1}^{K} p(\{\mu_k\})\right).$$

A graphical representation of the distribution is given in Figure 5.2.

Given the inputs $\{\mathbf{z}_i^{(d)}\}$, the aim is to infer the topic assignment variables $t_i^{(d)}$ for each word index i in document d, the parameters associated with each topic and the topic distribution $\theta^{(d)}$ for each document.

Inferring the topic assignment variables $t_i^{(d)}$: Given the topic distribution for document d, and the parameters associated with each topic, the topic assignment variables are independent, and can be obtained as shown below:

$$\begin{aligned} p(t_i^{(d)}=k|\theta^{(d)},\mathbf{z}_i^{(d)},\mu_k) &\propto p(t_i^{(d)}=k|\theta^{(d)})p(\mathbf{z}_i^{(d)}|t_i^{(d)}=k,\mu_k) \\ &= \theta_k^{(d)}\mathcal{N}(\mu_k,\rho^{-1}I)\,. \end{aligned}$$

Inferring the mixing proportions for a document $\theta^{(d)}$: Given the topics assigned to all the words in a document, the mixing proportions for document d can be sampled as shown below:

$$\begin{aligned} p(\theta^{(d)}|\{t_i^{(d)}\}) &\propto p(\theta^{(d)})\prod_{i=1}^{n_d} p(t_i^{(d)}|\theta^{(d)}) \\ &= Dir(\theta^{(d)}|\alpha)\prod_{i=1}^{n_d} Cat(t_i^{(d)}|\theta^{(d)})\,. \end{aligned}$$

Since the Dirichlet distribution is the conjugate prior of categorical distribution, the posterior distribution of $\theta^{(d)}$ is also Dirichlet with parameters $(\alpha+n_{d1},\ldots,n_{dK})$, where n_{dk} is the words in document d, that have been assigned to topic k.

Inferring the parameters of the topics μ_k: Given the topic assignment variables, the parameters of a topic depends only on the words that have been assigned to this topic. Hence,

we have

$$p(\mu_k|\{\mathbf{z}_i^{(d)}\},\{t_i^{(d)}\}) \propto p(\mu_k) \prod_{d\in D} \prod_{i:t_i^{(d)}=k} p(\mathbf{z}_i^{(d)}|t_i^{(d)}=k,\mu_k)$$
$$= \mathcal{N}(\mu_k|0,\rho_0^{-1}I) \prod_{d\in D} \prod_{i:t_i^{(d)}=k} \mathcal{N}(\mathbf{z}_i^{(d)}|\mu_k,\rho^{-1}I)\,.$$

As in the case of mixture models, the posterior distribution of μ_k is also a Gaussian distribution with mean m and covariance matrix S given as follows:

$$S = (n_k\rho + \rho_0)I\,,$$
$$m = \frac{n_k\rho}{n_k\rho+\rho_0}\left(\frac{\sum_{d\in D}\sum_{t_i^{(d)}=k}\mathbf{z}_i^{(d)}}{n_k}\right)\,,$$

where n_k is the number of words assigned to the k^{th} topic over all the documents.

Algorithm 5 lists the steps involved in inference in Gaussian topic models.

5.3.2 Discriminative encoding with Gaussian topic models

Let us consider the problem of clustering the faces in the Facebook[1] albums of all the Facebook users. As has been discussed earlier, the problem of clustering real-world images without any supervisory information, is quite challenging. However, the albums provide an implicit supervisory signal: A single person is likely to interact with a few individuals only, and hence, most of the images in an album can be grouped into a few individuals.

In this problem, the album corresponds to a document, and the latent representation of images in the album of an individual are the words. The aim is to learn the representation of images that allows one to infer the topic. Hence, we use Gaussian topic model as a prior over the real-valued latent features of a DisCoder. The corresponding model is referred to as GTM-DisCoder. Given the latent representation of images in all albums, it is straightforward to run Algorithm 5 to obtain the topic associated with each image. Here, we describe the mechanism for training the DisCoder to obtain the latent features.

Let $\mathbf{x}_i^{(d)}, i = 1,\ldots,n_d$ be the images in the d^{th} album, and $\mathbf{z}_i^{(d)}$ be the corresponding latent representations. Let $t_i^{(d)}$ be the topic assigned to this representation in the Gibbs inference step. Using the same arguments as in Section 5.2.2, the DisCoder objective for real-valued

[1] www.facebook.com

Algorithm 5: Gibbs inference in Gaussian topic models.

Input: Input $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}$, hyperparameters ρ and ρ_0

Output: The parameters of the topics $\{\mu_k\}$, the assignment variables $\{t_i^{(d)}\}$ and the mixing proportions of a document $\theta^{(d)}$

Initialization: Randomly assign the words to topics to get an initial topic assignment.

1 Repeat the following steps until the Markov chain has mixed thoroughly.

1. Repeat for $k = 1, \ldots, K$.

   (a) Compute the number of points assigned to each topic, and the mean of each topic as follows:

$$n_k = \sum_{d \in D} \sum_{t_i^{(d)} = k} 1 \,,$$
$$\bar{\mathbf{z}}_k = \frac{\sum_{d \in D} \sum_{t_i^{(d)} = k} \mathbf{z}_i^{(d)}}{n_k} \,.$$

   (b) Compute the following quantity:

$$\bar{\mu}_k = \frac{n_k \rho}{\rho + n_k} \bar{\mathbf{z}} \,,$$
$$S_k = (n_k \rho + \rho_0) I \,.$$

   (c) Sample $\mu_k \sim \mathcal{N}(\bar{\mu}_k, S_k)$.

2. Repeat for each document in the corpus $d \in D$

   (a) Sample $\theta^{(d)} \sim Dir(\alpha + n_1, \ldots, \alpha + n_K)$

   (b) Repeat for $i = 1, \ldots, n_d$

      (i) Sample $t_i^{(d)}$ from the following distribution

$$p(t_i^{(d)} = k | \mathbf{z}^{(i)}, \{\mu_k\}) \propto \theta_k^{(d)} \mathcal{N}(\mathbf{z}_i^{(d)} | \mu_k, \rho^{-1} I) \,. \tag{5.10}$$

latent features can be written as shown below:

$$\log \sum_{d' \in D} \sum_{j=1}^{n} \exp\left(-\frac{\lambda}{2}||\phi(\mathbf{x}_j^{(d')}) - \phi(\mathbf{x}_i^{(d)})||^2\right) + \frac{\rho}{2}||\phi(\mathbf{x}_i^{(d)}) - \mu_{t_i^{(d)}}||^2 \,. \tag{5.11}$$

We assign $\mathbf{z}_i^{(d)} = \phi(\mathbf{x}_i^{(d)})$ at the end of this step, and repeat Algorithm 5 on the latent representations $\{\mathbf{z}_i^{(d)}\}$ obtained in this step.

5.3.3 Optimization details

In practice, it is too expensive to perform Gibbs inference after each step of training. Hence, as in the case of GMM-DisCoder, we equate the topic assignment variables to the mode of their respective distributions. That is, we replace the step 2(b)(i) of Algorithm 5 by a hard assignment as shown below:

$$t_i^{(d)} = \underset{k=1,\ldots,K}{\operatorname{argmax}}\, \theta_k^{(d)} \mathcal{N}(\mathbf{z}_i^{(d)} | \mu_k, \rho^{-1} I) \,.$$

Furthermore, instead of sampling the mean of each topic, we equate it to the mean of the posterior distribution, that is, $\mu_k = \bar{\mu}_k$. We also tried to change the step 2(a) of Algorithm 5 by a hard assignment, by equating $\theta^{(d)}$ to the mean of the posterior distribution. However, this resulted in unusually large topics containing almost all the images. Hence, we do not change this step. We perform 5 steps of inference after every epoch of training the network. Furthermore, the topic assignment variables are reused from the previous inference step.

5.4 Experiments

5.4.1 MNIST

We train the GMM-DisCoder for clustering the MNIST dataset. Since the representations at the beginning of training are completely random, we initialize the number of clusters to be 400. After every epoch, we decrease the number of clusters by 1. The training is stopped after the desired number of clusters are achieved.

The training network consists of 2 convolutional layers with weight normalization and filter size 5×5. Each convolutional layer consists of 50 filters. The first convolutional layer is followed by 2×2 non-overlapping max pooling layer. Each convolutional layer is also followed by ReLU non-linearity. Finally, the output of the last convolutional layer (after ReLU non-linearity) is fed to a fully connected layer of output size 200. We use the Adam optimizer with a learning rate of .0001 and $\beta_1 = .5$.

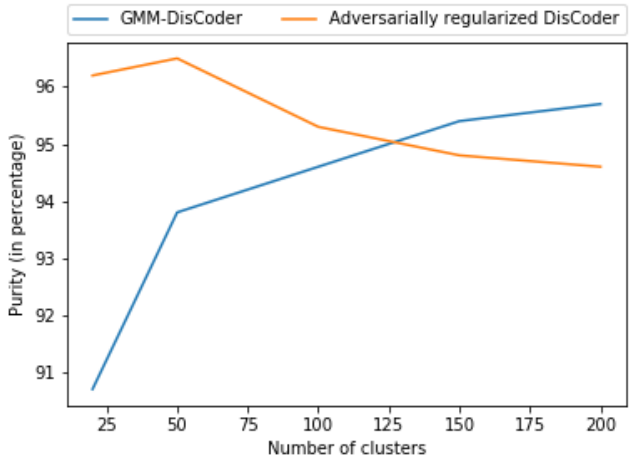


Figure 5.3: Clustering performance comparison between an adversarially regularized DisCoder and a GMM-DisCoder for varying number of clusters. The performance of a GMM-DisCoder decreases drastically as the number of clusters decrease.

We compare the proposed model with the adversarially regularized DisCoder with categorical latent features of the previous chapter for varying number of clusters. The evaluation metric used for comparison is the purity of the clusters. The results are given in Figure 5.3. As can be observed from the figure, DisCoder performs well when the number of clusters are high. However, the performance starts decreasing rapidly as the number of clusters reach 20. This is expected, since it is easier for the model to identify small clusters in the data, but much more difficult to identify global clusters. Samples from the 20 clusters identified by the model are shown in Figure 5.4

Note that unlike adversarially regularized DisCoder, a GMM-DisCoder does not require a generator for training. The model combines the advantages of hierarchical Bayesian modeling with deep neural networks. However, the performance of the model is worse than the performance of an adversarially regularized DisCoder with categorical latent features. This is expected, since a categorical DisCoder discovers embeddings that can be linearly separated into clusters with high confidence. In contrast, a GMM-DisCoder discovers embeddings that follow a Gaussian mixture prior. The discriminant function between two Gaussian distributions is a linear hyperplane, and hence, the separation boundaries between the clusters learnt by a GMM-DisCoder, form a minuscule subset of the boundaries learnt by a categorical DisCoder.

5.4.2 CUB-200-2011 dataset

Finally, we evaluate the GMM-DisCoder for clustering on the CUB-200-2011 dataset (Wah et al., 2011). The dataset consists of 200 classes of birds with approximately 50 images per class.

Figure 5.4: Samples from the clusters discovered by GMM-DisCoder. For each cluster, we have listed the points that have the highest probability of belonging to the cluster.

We use the first 100 classes as labeled data and the remaining classes are used as unlabeled data for training. For labeled data, we forward propagate the data through the network to compute the latent features. The mean is computed for each labeled class based on the true class labels. The network is then trained to predict the mean computed in the previous step as given in (5.9).

For unlabeled data, we forward propagate the data through the network to obtain the latent features. Approximate Gibbs inference is executed on the latent features as suggested in Section 5.2.3 to obtain the means of the clusters. We initialize the number of clusters to be 500 and decrease the number of clusters by 1 after every epoch. Given the means of the clusters, we train the network to minimize the objective in (5.9).

As suggested by Oh Song et al. (2016), we use the GoogleNet architecture pretrained on

ImageNet for training. We learn 128 dimensional embeddings for the data. For evaluation of clustering quality, we use normalized mutual information (NMI)[1]. The NMI between a set of clusters $\Omega = \{\omega_1, \ldots, \omega_K\}$ and a set of true classes $\mathbb{C} = \{c_1, \ldots, c_L\}$ is given by

$$\text{NMI}(\Omega, \mathbb{C}) = \frac{I(\Omega, \mathbb{C})}{[H(\Omega) + H(\mathbb{C})]/2} .$$

where I and H denote mutual information and entropy respectively, and are given as follows:

$$\begin{aligned} I(\Omega, \mathbb{C}) &= \sum_{k=1}^{K} \sum_{l=1}^{L} p(\omega_k \cap c_l) \log \frac{p(\omega_k \cap c_l)}{p(\omega_k) p(c_l)} , \\ H(\Omega) &= -\sum_{k=1}^{K} p(\omega_k) \log p(\omega_k) , \\ H(\mathbb{C}) &= -\sum_{l=1}^{L} p(c_l) \log p(c_l) . \end{aligned}$$

A maximum likelihood estimate of the above probabilities is given by

$$\begin{aligned} P(\omega_k \cap c_l) &= \frac{|\omega_k \cap c_l|}{n} , \\ P(\omega_k) &= \frac{|\omega_k|}{n} , \\ P(c_l) &= \frac{|c_l|}{n} . \end{aligned}$$

Here, $|A|$ indicates the cardinality of set A. Moreover, n is the total number of points in the dataset. We compare the proposed model with other models used for deep metric learning, such as contrastive embeddings (Hadsell et al., 2006), triplet embeddings (Schroff et al., 2015) and lifted structure embeddings (Oh Song et al., 2016). The results are given in Table 5.1. As can be observed, the proposed model outperforms the existing models for the task of clustering by a significant margin. One of the primary reason for the success of the proposed model is its ability in utilizing unlabeled classes for training the network as well. In contrast, the other models used in comparison, rely only on the labeled data.

[1] A short article about measures for evaluating clustering is given in https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html

Table 5.1: NMI metric evaluated on the unlabeled classes for CUBS dataset

Model	NMI(%)
Contrastive embeddings	46.8
Triplet embeddings	49.8
Lifted Structure embeddings	54.4
GMM-Discoder	**58.1±1.1**

5.4.3 20 newsgroup

In order to verify that the GTM-DisCoder indeed performs topic modeling, we employ it to learn topics on the 20-newsgroup dataset. In order to ensure reproducibility, we use a pre-processed version of the dataset[1]. The pre-processed dataset consists of 11,293 documents for training and 7,528 documents for testing, with a vocabulary size of 8,165 stemmed words. The data is almost evenly divided among the 20 classes. We combine the training and test sets for topic modeling. Documents with less than 2 words are discarded.

The encoding network for this task is a dictionary, that takes the word index in the vocabulary as input $\mathbf{x}_i^{(d)}$, and gives the corresponding embedding of word as output $\phi(\mathbf{x}_i^{(d)})$. We initialize the embeddings randomly, and train them using the objective in (5.11). A Gaussian topic model with 20 topics and $\alpha = 1$ is used as prior on the embeddings.

Note that the proposed model forces the embeddings of words that belongs to the same topic, to lie close to each other. Hence, in order to visualize the topics, we find the top 10 word embeddings that lie closest to the mean μ_k of the topics. The corresponding words for 10 selected topics are given in Table 5.2. Many of the topics learnt by the model correspond to a single class of the dataset. Some of the topics (for instance, topic 2) span multiple classes, indicating that the two classes have similar content.

We also evaluated the topic distribution of each document $\theta^{(d)}$ for clustering the documents $d \in D$. In particular, we assign each document to the topic k, for which $\theta_k^{(d)}$ is highest. We compare the proposed model against several standard algorithms for clustering and topic-modelling. These include K-means, Normalized cuts (Shi and Malik, 2000), probabilistic Latent Semantic indexing (pLSI) (Hofmann, 1999) and Latent Dirichlet Allocation (LDA), (Blei et al., 2003). For LDA, pLSI and the proposed model GTM-DisCoder, the topics correspond to clusters and a document is assigned to the cluster/topic with the highest posterior probability for the given document. This approach has been shown to be more effective than clustering the topic representations of documents (Lu et al., 2011).

The clustering results for 20 newsgroup are given in Table 5.3. As can be seen from the

[1]The pre-processed dataset is available at https://sites.google.com/site/renatocorrea02/textcategorizationdatasets

Table 5.2: Top 10 stemmed words from 10 selected topics learnt on the 20 newsgroup dataset by GTM-DisCoder. Note that most of the topics are coherent, that is, the words corresponds to a specific class of 20 newsgroup. Some of the topics span multiple classes of the dataset.

Topic 1	atheist bibl christian theism religion god belief church soul sin
Topic 2	driver card window mac softwar disk appl memori drive system
Topic 3	line polygon imag color code softwar graphic version format ftp
Topic 4	health diseas effect pain drug medic research doctor school food
Topic 5	game player team plai streak hockei win record playoff goal
Topic 6	israel moral kill arab death jew muslim histori citi talk
Topic 7	bui sell sale price offer ship cost interest compani book
Topic 8	car race engin bike road ride bmw mile truck diesel
Topic 9	right state govern tax bill liber administr legal presid polit
Topic 10	gun defenc kill legal shoot public firearm crimin arm shot

Table 5.3: Clustering results on text datasets

Method	Purity
K-means	34.3%
Normalized Cut	23.1%
pLSI	**57.7**%
LDA	54.7%
GTM-DisCoder	56.2%

results, there is a distinct improvement in performance as one moves from standard clustering algorithms to topic-modelling algorithms such as pLSI, LDA and GTM-DisCoder. Moreover, the performance obtained using GTM-DisCoder is at par with the performance using LDA and pLSI.

5.4.4 CIFAR-10 dataset

The topic modeling task in the previous section can be addressed using LDA or pLSI alone, since the vocabulary size is finite. In contrast, the number of possible sentences or images is unbounded. Hence, to test the proposed GTM-DisCoder, we create an artificial dataset by grouping the classes of CIFAR-10. Note that CIFAR-10 has only 10 classes, and hence, we fix the number of topics in CIFAR-10 to be 10. We set the value of α between 0 and 1. A document consists of 500 images (the latent representation of images correspond to the words in the document) and is created as follows:

1. Sample $\theta^{(d)} \sim Dir(\alpha)$.

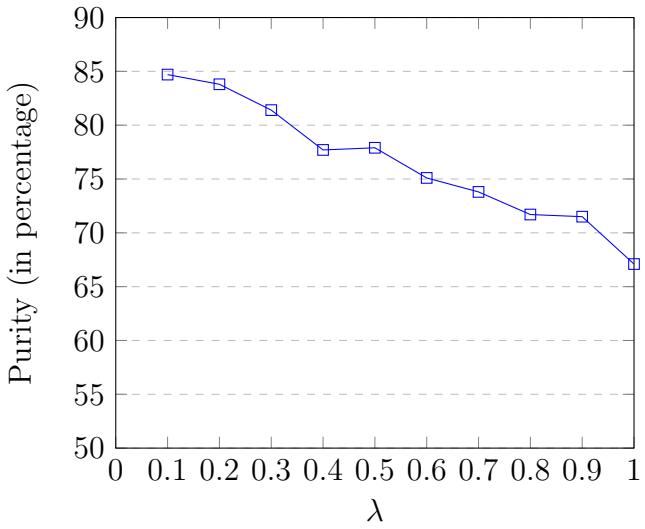


Figure 5.5: Purity of the topics on CIFAR-10 dataset for varying values of α

2. For $i = 1, \ldots, 500$

   (a) Sample $t_i^{(d)} \sim Cat(\theta^{(d)})$.

   (b) Add an image with label $t_i^{(d)}$ to the document.

We create 1000 documents in this manner. Since CIFAR-10 dataset consists of $50,000$ images, every image is present in approximately 10 documents. During training, we initialize the number of topics to be 500. After every iteration, we decrease the number of topics by 1. The α used for creating the dataset is also used during training. The training is stopped after 10 topics have been achieved. We use the same network architecture that we used for CIFAR-10 in the previous chapter, with the exception that the last linear layer generates a 500 dimensional output with no softmax.

We plot the purity of the topics for varying values of α in Figure 5.5. As can be observed, when α is small, the purity of the topics is quite high. This is expected since for small values of α, each document contains images from a single class only. However, as α increases, the number of unique classes present in a document increase, and correspondingly the purity decreases. However, it is worthy to note that even at $\alpha = 1$, the purity of the topics is close to 67%. In contrast, a completely unsupervised DisCoder achieves a purity of 31% on this dataset. Hence, the GTM-DisCoder is able to detect the CIFAR-10 classes from the above dataset.

5.5 Discussion

In this chapter, we explored the coupling of hierarchical Bayesian priors with discriminative encoders for unsupervised learning tasks. We focused on DisCoders with Gaussian mixture and Gaussian topic priors on the latent features. The Gaussian mixture prior encouraged similar objects to lie closer to each other in the latent space. Similarly, the Gaussian topic prior encourages objects that belong to the same 'topic' to lie close to each other in the latent space. Meanwhile, it also utilizes the structure of 'documents' to enforce that the objects in a given document belong to only a few topics. Here, the word 'document' is used to refer to any observed collection of objects.

In this chapter, we assumed the covariance matrix of the topics or clusters to be a scaled identity matrix. It is possible to define a prior on the covariance matrix of clusters or topics using the conjugate inverse Wishart prior. However, we expect the encoding network to be powerful enough to learn embeddings that result in spherical Gaussian clusters or topics.

The proposed work represents the first step in an intermarriage between deep learning and hierarchical Bayesian modeling for unsupervised learning. In particular, topic models utilize the structure of documents as bag of objects, to learn meaningful embeddings for the objects. In addition to the 'bag of objects' structure, there are other several commonly observed structures present in the data. For instance, the sequential structure of sentences in a document can be used to learn embeddings for sentences, by employing an autoregressive prior over the sentence embeddings. Similarly, to exploit the spatial structure of images, one can define a spatial autoregressive prior over the patches of the image. Currently, the successes and failures of discriminative encoders with more interesting priors are being explored by us.

Chapter 6

Learning to Segment from Weakly-Labeled Images

6.1 Introduction

semantic segmentation of images is the problem of assigning the pixels of an image to a selected set of predefined labels, based on the semantic structure that the pixel belongs to. Most successful models for semantic segmentation of images employ some variation of CNN for computing the probability distribution over the classes for each pixel. During inference, these distributions are fed as unary potentials to a fully connected CRF with Gaussian edge potentials, and a joint labeling for the pixels of the image is inferred from the CRF. The work by (Krähenbühl and Koltun, 2011), allows for efficient inference in such models.

Successful semantic segmentation of images requires access to a large number of images that have been densely labelled. However, it has been observed that dense labeling of images is an expensive and time-consuming operation (Papandreou et al., 2015, Pinheiro and Collobert, 2015, Pathak et al., 2014). The number of densely labeled images is a minuscule percentage of the total set of available images. Hence, the models that rely solely on densely labeled images, are limited in their scope. These models are referred to as fully-supervised models in the sequel.

The limitations of fully supervised models has necessitated the development of models that can incorporate weakly labeled images for training. These include models that utilize bounding box prior (Lempitsky et al., 2009, Dai et al., 2015, Papandreou et al., 2015, Xu et al., 2015), few points per class (Bearman et al., 2016) and image-level labels (Vezhnevets et al., 2011, Papandreou et al., 2015, Pinheiro and Collobert, 2015). Of particular interest are models that rely on image-level labels only, since the web provides an almost unlimited source of weakly annotated images.

Unfortunately, the decoupled CNN-CRF combination (or CNN alone) fares poorly, when only image-level labels are available (Pinheiro and Collobert, 2015, Pathak et al., 2015). To alleviate this hurdle, several researchers have resorted to the use of localization cues, such as saliency and attentions maps (Wei et al., 2016, Kolesnikov and Lampert, 2016) or objectness priors (Pinheiro and Collobert, 2015, Wei et al., 2016), thereby improving performance to an extent. Improvements in CNN architecture for segmentation (Chen et al., 2016, Yu and Koltun, 2015), has further aided in improved performance.

A model for semantic segmentation that can utilize weakly-labelled images is certainly of tremendous practical value. The desired model must be scalable enough to deal with the much larger set of weakly-labelled images. Furthermore, the desired model should be interpretable enough, so that one can modify it to incorporate higher levels of supervision. We present such a model in this chapter.

The main contribution of this chapter is a model that incorporates CRF in CNN that can be trained almost as fast as CNN alone. Previous attempts to incorporate CRF in CNN training (Zheng et al., 2015, Schwing and Urtasun, 2015) relied on a recursive expansion of the mean-field distribution of a CRF. Training the expanded mean-field distribution is several times more expensive than training the underlying CNN, since multiple forward and backward propagations need to be done through the iterating network (which is often an LSTM) for a single update. Hence, the authors in (Zheng et al., 2015) are forced to use a batch size of 1, and an extremely small learning rate (10^{-13} vs 10^{-3}). In contrast, we force the output of the CNN to satisfy the constraints imposed by a CRF. We show that this objective can be achieved by forcing the output distribution at each pixel to be close to the output distribution of its neighbors. Consequently, the time required for training the CNN-CRF combination is almost the same as the time required for training the CNN alone.

6.2 Preliminaries and background

A brief introduction to conditional random fields for semantic segmentation of images is provided in this section. The probability distributions are indicated by lower case letters p and q. Subscripts in the distribution indicate the location, whereas superscripts indicate the name of the distribution. The labels in the segmentation mask form a grid, which is denoted by $\mathcal{G}$.

A conditional random field is an example of undirected graphical model that models the conditional distribution of output given the input, when the output is structured in the form of factors. For the problem of semantic segmentation of images, the image is the input, whereas the pixel-level labels form the output. A conditional random field is completely characterized by its potential functions.

Traditionally, conditional random fields employ two forms of potential functions: unary potentials and binary potentials. A unary potential ϕ_i is a function of the conditioning variable $\mathbf{x}$ and a single output variable z_i. A binary potential ϕ_{ij} is a function of the conditioning variable $\mathbf{x}$ and two output variables z_i, z_j.

Unary potentials: The unary potentials encode the suitability of assigning a specific label at a specific location. For instance, the potential $\phi_i(z_i, \mathbf{x})$ encodes the suitability of assigning the label z_i to location i, based on the input image $\mathbf{x}$. A popular approach is to learn a local classifier for each location in the image, using the features extracted locally from the image. The unary potential for a specific label at a specific location is then equated to the negative log-probability of observing the label at that location, that is, $\phi_i(z_i, \mathbf{x}) = -\log p^{\mathrm{u}}(z_i|\mathbf{x})$. Here, p^{u} is the conditional distribution of the labels given the input learnt by the local classifier. These local classifiers have largely been replaced by convolutional neural networks that consider local as well as global information (Zheng et al., 2015, Chen et al., 2016).

Binary potentials: The binary potential $\phi_{ij}(z_i, z_j, \mathbf{x})$ measures the compatibility of two labels z_i and z_j for locations i and j. For instance, if i and j are neighboring locations with similar color, the labels at the two locations is expected to be the same. To ensure the tractability of inference, it is common to specify the binary potentials for neighboring locations only, that is $\phi_{ij}(z_i, z_j, \mathbf{x}) = 0$ if $j \notin \mathcal{N}(i)$. This reduces the number of factors present in the model from $\mathcal{O}(m^2)$ to $\mathcal{O}(m)$, where m is the number of pixels.

The most commonly used binary potential is the contrast sensitive Potts model (Shotton et al., 2006, Kohli et al., 2009, Boykov and Jolly, 2001), which captures the difference in color among neighboring locations, that is

$$\phi_{ij}(z_i, z_j, \mathbf{x}) = \alpha \exp\left(-\beta||x_i - x_j||^2\right) \mathbb{1}(z_i \neq z_j)\,.$$

When the labels are correlated, it is reasonable to replace the indicator function by a learnable parameter (Krähenbühl and Koltun, 2011).

$$\phi_{ij}(z_i, z_j, \mathbf{x}) = \exp\left(-\beta||x_i - x_j||^2\right) \mu(z_i, z_j)\,.$$

Here, μ is a matrix of size $K \times K$, where K is the number of labels.

CRFs that rely on neighborhood-based binary potentials only, favor smooth object boundaries. Hence, the resultant segmentation masks are often misaligned with the actual boundaries of the object. Using permutohedral lattices (Adams et al., 2010), it is possible to extend the binary potentials from neighboring locations to all pairs of locations in an image while allowing

tractable inference (Krähenbühl and Koltun, 2011). However, this is only true for potentials that can be written in a specific form. In particular, let the binary potential $\phi_{ij}(z_i, z_j, \mathbf{x})$ have the form as shown below:

$$\phi_{ij}(z_i, z_j, \mathbf{x}) = \mu(z_i, z_j) \sum_{r=1}^{R} \alpha_r k_r(f_i(\mathbf{x}), f_j(\mathbf{x})) \,, \tag{6.1}$$

for some choice of functions f_i and f_j. If $k_r(f_i(\mathbf{x}), f_j(\mathbf{x}))$ has the form of a Gaussian kernel, that is

$$k_r(f_i(\mathbf{x}), f_j(\mathbf{x})) = \exp\left(-(f_i(\mathbf{x}) - f_j(\mathbf{x})^{\mathrm{T}} \Lambda_r (f_i(\mathbf{x}) - f_j(\mathbf{x}))\right) \,,$$

for some choice of Λ_r, then it is possible to use permutohedral lattice to allow binary potentials for all pairs of locations, while allowing tractable inference. Such a CRF is referred to as a fully connected Gaussian CRF, and is popularly used for semantic segmentation (Krähenbühl and Koltun, 2011, Chen et al., 2016, Zheng et al., 2015). A common choice is to equate $f_i(\mathbf{x}) = [x_i, i]$, that is, the color of the i^{th} pixel and its location.

A CRF with unary and binary potentials only, is referred to as a *pairwise CRF*. The joint distribution of a pairwise CRF is given below:

$$p(\mathbf{z}|\mathbf{x}) = \frac{1}{\mathcal{Z}} \exp\left(-\sum_{i \in \mathcal{G}} \phi_i(z_i, \mathbf{x}) - \sum_{i \in \mathcal{G}} \sum_{j \in \mathcal{N}(i)} \phi_{ij}(z_i, z_j, \mathbf{x})\right) \,, \tag{6.2}$$

where $\mathcal{N}(i)$ indicates the neighbors of the i^{th} node.

6.3 Proposed Model

6.3.1 From image-level labels to pixel-level labels

Most fully supervised approaches for semantic segmentation learn a mapping from the pixels $\mathbf{x}$ to the pixel-level labels $\mathbf{z}$ using deep convolutional networks. Unfortunately, this approach cannot be employed when pixel-level labels are not available. Hence, we use heuristics to obtain fake pixel-level labels using the image-level information, and the output of the CNN.

In particular, let $p(\mathbf{z}|\mathbf{x})$ be the conditional distribution over the pixel-level labels given the image. We assume that the distribution factorizes completely, that is, $p(\mathbf{z}|\mathbf{x}) = \prod_{j \in \mathcal{G}} p_j(z_j|\mathbf{x})$, where p_j is the distribution at location j, and z_j is the corresponding label. Furthermore, we

assume that the distribution is parametrized by a CNN f, that is,

$$p_j(\ell|\mathbf{x}) = \frac{\exp(f_{j\ell}(\mathbf{x}))}{\sum_{\ell'=0}^{L}\exp(f_{j\ell'}(\mathbf{x})}\,.$$

Given the image $\mathbf{x}$ and the image-level labels $\mathbf{y}$, our aim is to infer the fake pixel-level labels $\mathbf{z}^{\text{fake}}$. One can then maximize the log-probability of the inferred labels given the image, that is, $\log p(\mathbf{z}^{\text{fake}}|\mathbf{x})$.

Alternatively, one can infer a distribution over the fake labels $q^{\text{fake}}(\mathbf{z}|\mathbf{x},\mathbf{y})$. Given the fake distribution, the classification objective can be rewritten as

$$\mathcal{L}_{\text{class}} = \mathbb{E}_{q^{\text{fake}}(\mathbf{z}|\mathbf{x},\mathbf{y})}\log p(\mathbf{z}|\mathbf{x})\,.$$

The advantage of using a distribution over the labels rather than the fake-labels themselves, is to allow for the incorporation of confidence in the fake labels. For instance, if we are not sure whether a pixel at location j belongs to the background class or an object class (say 'person'), the fake distribution at that location $q_j^{\text{fake}}(z_j|\mathbf{x},\mathbf{y})$ can assign high probability to the background and the 'person' class, while assigning 0 probability to all the other classes. The same effect can not be achieved using fake labels only. Hence, we use distribution over the fake labels rather than fake labels in our model.

6.3.2 Inferring the fake labels

In absence of any restriction, the model can choose to assign all the pixels to a single class, for instance, the background class. In order to prevent this from happening, one needs to ensure that for each class present in the image, at least a certain percentage of pixels are allotted to that class. Furthermore, one also needs to ensure that no pixels are allotted to classes absent from the image. Hence, we couple the distribution with a prior to obtain the distribution over the fake labels.

$$q_j^{\text{fake}}(\ell|\mathbf{x},\mathbf{y}) = \frac{p_j(\ell|\mathbf{x})p^{\text{prior}}(\ell|\mathbf{y})}{\sum_{\ell'}p_j(\ell'|\mathbf{x})p^{\text{prior}}(\ell'|\mathbf{y})}\,,$$

for $\ell = 0,\ldots,L$, and $j \in \mathcal{G}$. In order to complete the description, we define the prior distribution $p^{\text{prior}}(\ell|\mathbf{y})$ as below:

$$p^{\text{prior}}(\ell|\mathbf{y}) = \begin{cases}\beta_\ell, & \text{if } \ell \in \mathbf{y}\\ 0, & \text{otherwise}\end{cases}$$

Images in the ImageNet dataset are assumed to contain only one foreground object, and hence $\#\mathbf{y} = 2$. We learn the constants $\beta_\ell, \ell \in \mathbf{y}$ independently for each image. In particular,

for each image, we learn the most *non-informative* prior that can guarantee the assignment of a certain percentage of pixels to each class present in the image, while assigning no pixels to classes absent from the image. In order to quantify *information*, we maximize the *entropy* of the prior distribution, while simultaneously forcing it to satisfy a set of constraints. That is,

$$\begin{aligned} \underset{\beta_\ell, \ell \in \mathbf{y}}{\text{minimize}} \quad & \sum_{\ell \in \mathbf{y}} \beta_\ell \log \beta_\ell \\ \text{subject to} \quad & \frac{1}{m} \sum_{j \in \mathcal{G}} q_j^{\text{fake}}(\ell | \mathbf{x}, \mathbf{y}) \geq c_\ell, \ \ell \in \mathbf{y} \\ \text{and} \quad & \sum_{\ell \in \mathbf{y}} \beta_\ell = 1, \text{ and } \beta_\ell \geq 0, \ \ell \in \mathbf{y}. \end{aligned}$$

The constants c_ℓ dictates the percentage of pixels that are guaranteed to belong to class ℓ, if label ℓ is present in the image. We choose $c_\ell = .4$ for the background class, and $c_\ell = .2$ for all the other object classes present in the image.

For images in the ImageNet dataset, $\#\mathbf{y} = 2$, and hence the above optimization problem contains only two variables, β_0 and β_ℓ, where ℓ is the label of the foreground object is present in the image. Furthermore, by equating β_0 to $1 - \beta_\ell$, we further reduce the number of variables from 2 to 1. The resultant optimization problem can be written as follows:

$$\begin{aligned} \underset{0<\beta_\ell<1}{\text{minimize}} \quad & \beta_\ell \log \beta_\ell + (1 - \beta_\ell) \log(1 - \beta_\ell) \\ \text{subject to} \quad & \frac{1}{m} \sum_{j \in \mathcal{G}} q_j^{\text{fake}}(\ell | \mathbf{x}) \geq c_\ell \,, \\ \text{and} \quad & \frac{1}{m} \sum_{j \in \mathcal{G}} q_j^{\text{fake}}(0 | \mathbf{x}) \geq c_0 \,, \end{aligned}$$

where

$$q_j^{\text{fake}}(\ell | \mathbf{x}) = \frac{p(\ell | \mathbf{x}) \beta_\ell}{p(\ell | \mathbf{x}) \beta_\ell + p(0 | \mathbf{x})(1 - \beta_\ell)}$$

and $q_j^{\text{fake}}(0 | \mathbf{x}) = 1 - q_j^{\text{fake}}(\ell | \mathbf{x})$.

The above problem is an optimization problem in single variable. We can solve it approximately by evaluating the constraints for several values of $\beta_\ell \in [0, 1]$. This can be achieved very efficiently on a GPU, since it involves element-wise operations on matrices of probability distributions. Let S be the set of all the selected values of β_ℓ which satisfy the constraints. Among these values, we return the value of β_ℓ which minimizes the objective.

If none of the selected values of β_ℓ satisfy the constraints, we choose the value of β_ℓ which

minimizes the following:

$$\left(c_\ell - \frac{1}{m}\sum_{j\in\mathcal{G}} q_j^{\text{fake}}(\ell|\mathbf{x})\right)_+ + \left(c_0 - \frac{1}{m}\sum_{j\in\mathcal{G}} q_j^{\text{fake}}(0|\mathbf{x})\right)_+ .$$

6.3.3 Incorporating neighborhood information

To ensure the correct alignment between the predicted boundaries and the actual boundaries, we utilize the following information: Pixels that lie close together and have similar color, also have the same label. Hence, we force the distribution at location i to be close to the distribution of its neighbors. Towards that end, we compute a neighborhood distribution for each location p^{neighb}, and minimize the KL-divergence between the output distribution at that location and the corresponding neighborhood distribution. The corresponding objective is given by

$$\mathcal{L}_{\text{neighb}} = -\text{KL}(p(\mathbf{z}|\mathbf{x})||p^{\text{neighb}}(\mathbf{z}|\mathbf{x}))\,.$$

The combined objective is given by

$$\mathcal{L} = \mathcal{L}_{\text{class}} + \lambda\mathcal{L}_{\text{neighb}}\,.$$

We propose two approaches for computing the neighborhood distribution.

Weighted mean: In this approach, the neighborhood distribution is computed as follows:

$$p_i^{\text{neighb}}(\ell|\mathbf{x}) = \frac{\sum_{j\neq i} k(f_i(\mathbf{x}), f_j(\mathbf{x}))p_j(\ell|\mathbf{x})}{\sum_{j\neq i} k(f_i(\mathbf{x}), f_j(\mathbf{x}))}\,,$$

for $\ell \in \{0,\ldots,L\}$. Here, $k(f_i(\mathbf{x}), f_j(\mathbf{x}))$ is a measure of similarity between the locations i and j. For our purpose, we define the neighbors as all the locations that lie close to the current location, and the corresponding pixels have similar color. Hence, we use the contrast sensitive two-kernel potential (Krähenbühl and Koltun, 2011) defined in terms of pixel locations i and pixel brightness x_i as follows:

$$\begin{aligned} k(f_i(\mathbf{x}), f_j(\mathbf{x})) =& k_1(f_i(\mathbf{x}), f_j(\mathbf{x})) + k_2(f_i(\mathbf{x}), f_j(\mathbf{x})) \\ =& w_1 \exp\left(-\frac{|x_i - x_j|^2}{2\theta_\alpha^2} - \frac{|i-j|^2}{2\theta_\beta^2}\right) + w_2 \exp\left(-\frac{|i-j|^2}{2\theta_\gamma^2}\right)\,, \end{aligned}$$

where, $f_i(\mathbf{x}) = [x_i, i]$. Here, $w_1, w_2, \theta_\alpha, \theta_\beta$ and θ_γ are hyperparameters that are fixed during

training. As discussed in (Krähenbühl and Koltun, 2011), the second term prevents the formation of small isolated regions as segments.

Exponentiated weighted mean: As the name suggests, the neighborhood distribution in this approach is obtained by exponentiating the weighted mean. The exponentiation causes the neighborhood distribution to be sharper, resulting in high confidence predictions.

$$p_i^{\text{neighb}}(\ell|\mathbf{x}) = \frac{1}{\mathcal{Z}_i} \exp\left(\sum_{j \neq i} \hat{k}_{\mathbf{x}}(i,j) p_j(\ell|\mathbf{x})\right), \tag{6.3}$$

where

$$\hat{k}(f_i(\mathbf{x}), f_j(\mathbf{x})) = \frac{k(f_i(\mathbf{x}), f_j(\mathbf{x}))}{\sum_{j' \neq i} k(f_i(\mathbf{x}), f_{j'}(\mathbf{x}))}.$$

Next, we show the connection between the exponentiated weighted mean and CRF.

6.3.4 Connections with CRF

We derived the model in the previous section by relying on intuition alone. In particular, it made sense to force the prediction at any output neuron to be close to the predictive distribution of its neighbors. In this section, we provide a formal justification for the choice of the neighborhood-based objective function. In particular, we show that the objective emerges naturally, when a CRF is used as a prior while computing the conditional log-likelihood.

Given an image $\mathbf{x}$, let the CRF prior over the segmentation masks be defined as below:

$$p^{\text{CRF}}(\mathbf{z}|\mathbf{x}) = \frac{1}{\mathcal{Z}} \exp\left(-\sum_{i<j} \phi_{ij}(z_i, z_j, \mathbf{x})\right),$$

where $\mathcal{Z}$ is the normalization constant. Note that the prior distribution has no unary potentials. We further assume that binary potentials ϕ_{ij} have no trainable parameters.

The prior provides a distribution over all possible segmentation masks for a given image. Furthermore, let $\psi_{ij}(z_i, z_j, \mathbf{x})$ has the form

$$\psi_{ij}(z_i, z_j, \mathbf{x}) = -k(f_i(\mathbf{x}), f_j(\mathbf{x})) \mathbb{1}(z_i = z_j),$$

for some choice of kernel k. The corresponding CRF prior gives low probability to masks $\mathbf{z}$ that assign different labels to pixels i and j with high similarity (that is, high $k(f_i(\mathbf{x}), f_j(\mathbf{x}))$). This is a reasonable prior assumption about the segmentation mask of an image. Note that the prior does not penalize masks that assign the same label to pixels i and j with low similarity. This allows the inclusion of object classes with multicolored instances. For instance, the dress

a person wears will often be colored differently from his skin color.

If the CRF prior is approximated by a fully factorized distribution, the resultant distribution will have to satisfy the constraints entailed in Lemma 4.

Proposition 4 *Let q^{MF} be the mean field approximation to the CRF prior p^{CRF}. That is, among all distributions of the form $q(\mathbf{z}) = \prod_{i \in \mathcal{G}} q_i(z_i)$, let q^{MF} be the one that minimizes $KL(q||p^{CRF})$. Then the distribution q^{MF} satisfies the following constraints:*

$$q_i^{MF}(\ell) = \frac{1}{\mathcal{Z}_i} \exp\left(\sum_{j \neq i} k(f_i(\mathbf{x}), f_j(\mathbf{x})) q_j^{MF}(\ell)\right) ,$$

for $\ell = 0, \ldots, L$ and $i \in \mathcal{G}$. These constraints are referred to as mean field constraints. $\mathcal{Z}_i$ is the normalization constant that ensures that the distribution sums up to 1.

Proof: *The proof of this result can be obtained by differentiating the KL-divergence divergence with-respect-to the components of q and equating it to 0. Towards that end, we expand the KL-divergence term as given below:*

$$\begin{aligned} KL(q||p^{CRF}) &= \sum_{\mathbf{z}} q(\mathbf{z}) \log q(\mathbf{z}) - \sum_{\mathbf{z}} q(\mathbf{z}) \log p^{CRF}(\mathbf{z}|\mathbf{x}) + C \\ &= \sum_{i \in \mathcal{G}} \sum_{z_i=0}^{L} q_i(z_i) \log q_i(z_i) - \sum_{i<j} \sum_{z_i=z_j=0}^{L} q_i(z_i) q_j(z_j) k(f_i(\mathbf{x}), f_j(\mathbf{x})) + C \,. \end{aligned} \tag{6.4}$$

Here, C is a constant that does not depend on q. Differentiating the above expression with respect to $q_i(z_i)$ and equating it to 0, we get

$$q_i(z_i) = \exp\left(1 + \sum_{j \neq i} q_j(z_j) k(f_i(\mathbf{x}), f_j(\mathbf{x}))\right) .$$

Finally, since q_i is a probability distribution that sums up to 1, we normalize q_i to get the desired distribution q^{MF}. □

Coming back to the predictive distribution for segmentation masks in our model p, if one wishes to impose a CRF prior on p, one must force it to satisfy the mean-field constraints, that is,

$$p_i(\ell) = \frac{1}{\mathcal{Z}_i} \exp\left(\sum_{j \neq i} k(f_i(\mathbf{x}), f_j(\mathbf{x})) p_j(\ell)\right) ,$$

for $\ell \in \{0 \ldots, L\}$ and $i \in \mathcal{G}$. The distribution on the RHS of the above equation is exactly the neighborhood distribution p^{neighb} of (6.3), with the exception that the kernel k is normalized

in (6.3). The distribution p is defined in terms of the output of a neural network. Hence, instead of the equality constraints imposed by the mean-field, we add the term $\mathrm{KL}(p||p^{\text{neighb}})$ to the objective. The KL-divergence term forces the output of the network to satisfy the mean field constraints imposed by the CRF prior.

Note: The binary potential used in the CRF prior in this section is given by $\phi_{ij}(z_i, z_j, \mathbf{x}) = -k(f_i(\mathbf{x}), f_j(\mathbf{x}))\mathbb{1}(z_i = z_j)$. In contrast, the binary potential commonly used for semantic segmentation has the form $\phi(z_i, z_j, \mathbf{x}) = k(f_i(\mathbf{x}), f_j(\mathbf{x}))\mathbb{1}(z_i \neq z_j)$. However, when the kernel k is normalized at the pixel-level (as has been suggested in (Krähenbühl and Koltun, 2011)), the resultant distributions are exactly the same.

6.4 Relation with similar works

Recent works on semantic segmentation using deep architectures have focused on pairwise CRFs with unary and binary potentials only. The unary potentials are specified by the output of a CNN while the binary potentials have no learnable parameters (Chen et al., 2016, Zheng et al., 2015, Schwing and Urtasun, 2015). The work by Lin et al. (2016) allows the binary potentials to be learnable as well.

These works differ from the proposed model in the learning algorithm used for training the parameters of the CRF. Most of the works that combine CNNs with CRFs use piecewise learning (Sutton and McCallum, 2005, Lin et al., 2016, Chen et al., 2016), that is, the energy function is decomposed into its potentials, and each potential is normalized individually. For instance, if $\phi_1, \ldots, \phi_K$ are the potentials that form the energy function, the piecewise approximation to the objective is given by

$$\log \prod_{k=1}^{K} \frac{\exp \phi_k(\mathbf{x})}{\mathcal{Z}_k},$$

where $\mathcal{Z}_k$ is the normalization constant for the k^{th} potential.

Hence, in a pairwise CRF, with unary potentials given by the output of a CNN, each output location of the CNN is trained independently. Furthermore, the contribution of the pairwise potentials is not incorporated during training of the parameters of the CNN. Hence, the training is equivalent to the training of several independent classifiers, one for each location in the segmentation mask.

While piecewise training is extremely efficient, the lower bound that it optimizes is a very weak lower bound of true log-likelihood. By training the local classifier without incorporating the binary potentials, we ignore the dependence among labels of nearby pixels with similar color. More importantly, it is completely unsuitable when the pixel-level labels are absent,

which is the main concern of this thesis.

More recently, several authors have considered training the mean field approximation $q(\mathbf{z})$ rather than the actual CRF distribution $p(\mathbf{z}|\mathbf{x})$ (Kraehenbuehl and Koltun, 2013, Zheng et al., 2015, Schwing and Urtasun, 2015) for semantic segmentation. The mean-field approximation $q(\mathbf{z})$ for a distribution $p(\mathbf{z})$ is the distribution that minimizes the KL-divergence $\mathrm{KL}(q(\mathbf{z})||p(\mathbf{z}))$. By computing the gradient of the KL-divergence with respect to q, and equating it to 0, one can obtain an iterative algorithm for finding the minima. For a pairwise CRF, the mean-field update equations are given by

$$q_i^{(t+1)}(z_i) = \frac{1}{\mathcal{Z}_i} \exp\left(-\phi_i(z_i, \mathbf{x}) - \sum_{j \in \mathcal{N}(i)} \mathbb{E}_{z_j \sim q_j^{(t)}} \phi_{ij}(z_i, z_j, \mathbf{x})\right), \tag{6.5}$$

where $q_i^{(t)}$ is the distribution of the i^{th} location of the mean field approximation at the t^{th} iteration.

Hence, the mean field approximation at the $(t+1)^{st}$ iteration can be defined recursively, as a function of the mean-field approximation at the t^{th} iteration and the potential functions. Consequently, the gradient of the mean-field distribution for $(t+1)^{st}$ iteration can be written as a function of the gradient of the approximation at the t^{th} iteration and the gradient of the potential functions. This approach for training CRFs has been used in (Kraehenbuehl and Koltun, 2013, Zheng et al., 2015, Schwing and Urtasun, 2015).

While the above approach is clearly superior to piecewise learning, it is also extremely expensive to implement and train. Several updates need to be performed during each iteration of training to compute the mean field distribution. Similarly, the gradient of the loss computed from $q^{(T)}$ (after T time steps) needs to be backpropagated through several iterations. The resultant model is quite slow to train and requires access to a large amount of memory during training. Furthermore, the model is prone to vanishing gradient problems. The corresponding CNN needs to be initialized by training the CNN end-to-end using piecewise learning. Unfortunately, when pixel-level labels are absent, the resultant initialization may be quite bad. In contrast, the proposed model requires a single step per iteration that computes the neighborhood distribution. The neighborhood distribution is then forced to be closed to the original distribution. Hence, we are able to satisfy the constraints of a CRF without resorting to a recursive expansion of the mean-field distribution, by adding a KL-divergence loss to the output of a CNN. We demonstrate the performance improvement achieved by the proposed method as compared to a weakly supervised CRF-RNN (Zheng et al., 2015) in the experiments section.

6.5 Experimental setup

In this section, we describe the network architecture, the hyperparameters and the datasets used for the experiments.

6.5.1 Network architecture

We download the pretrained VGG16 network (trained on ImageNet for classification) from torchvision[1], and modify it for the task of semantic segmentation. The VGG16 network consists of 13 convolutional layers and 3 fully connected layers. Firstly, we remove the fully connected layers and the last pooling layer. The resultant network receives images of size 224×224, and generates 512 feature maps of size 14×14. The *receptive field* of a neuron in the last convolutional layer is 196×196, and hence, it nearly encompasses the entire input image. This implies that every neuron in the last layer has access to almost the entire image. This network serves as an encoder in our model.

To learn fine-grained contours of objects, we add skip connections from the layers of the encoder to the layers of the decoder. The receptive-field size of the neurons in the encoder is much smaller than their counterpart in the decoder. Hence, they have access to more fine-grained information. We perform 1×1 convolution to the output of the layers of the encoder before concatenating them with the layers of the decoder. The concatenated output is fed through multiple layers of convolution and non-linearities, to allow the final prediction layer to learn non-linear mapping of the local and global information. The final architecture is shown in Figure 6.1.

6.5.2 Dataset

The primary purpose of weakly supervised semantic segmentation is to utilize the vast quantity of images labelled at the image-level. Hence, models that use weak-supervision for training often employ a much larger dataset than fully-supervised models. For instance, the model in (Wei et al., 2016) mines the pages of *Flickr*[2], to generate the training data. Similarly, the model in (Saleh et al., 2016) uses a subset of Flickr dataset (Huiskes et al., 2010) for training. A clean subset of the ImageNet dataset is used in (Hou et al., 2016), while a much larger subset of the same dataset is used in (Pinheiro and Collobert, 2015). More importantly, almost all the approaches for semantic segmentation utilize the ImageNet dataset for pretraining the network.

We follow the experimental setup of (Pinheiro and Collobert, 2015). In particular, we downloaded images of objects belonging to the 20 object classes in the VOC 2012 dataset (Ev-

[1]https://github.com/pytorch/vision/tree/master/torchvision
[2]https://www.flickr.com/

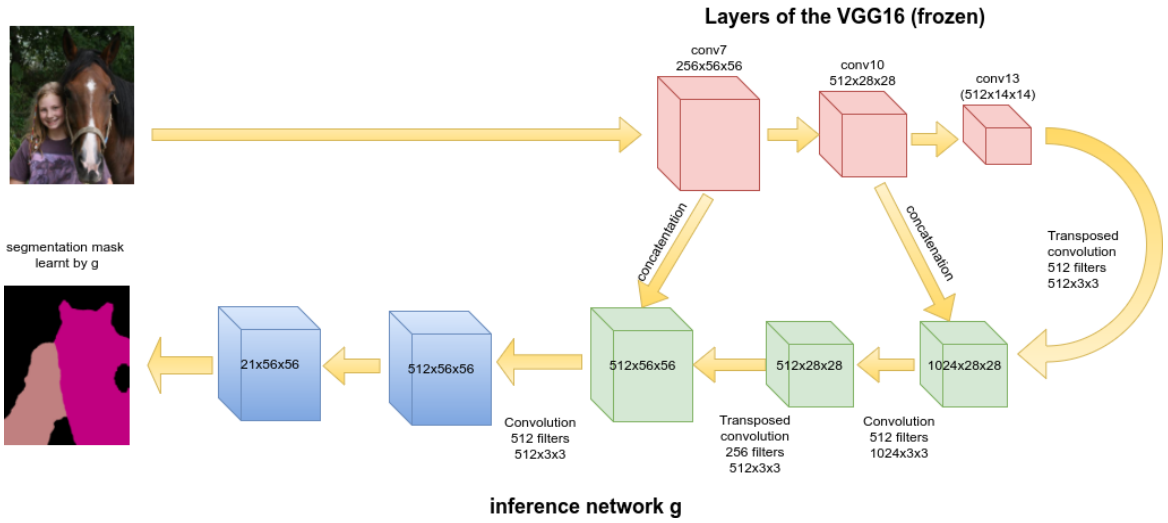


Figure 6.1: The inference network of the proposed model. Except for the last layer, each convolutional layer in the inference network is followed by a ReLU layer and a batch normalization layer. The last layer is followed by exponentiation and normalization to get ϕ.

eringham et al., 2012) from the Imagenet database (Russakovsky et al., 2015). Several authors mine the dataset further to obtain a set of simple images which contain the object against a plain background. However, we choose to use the entire dataset to minimize manual dependence.

For testing, we use the validation and test set of VOC 2012 dataset. For most of our experiments, we use the validation set only, since the ground truth is publicly available. The test set is used only for comparing the final model against the state-of-the-art.

Almost all approaches for weakly supervised semantic segmentation report the results on VOC 2012 dataset. This is primarily because the classes in VOC 2012 are discrete object categories such as sheep, person, dog, etc. Most images in VOC 2012 contain very few of these object classes, and hence, it is easy to utilize weak labels for training. Two classes that almost always occur together, will be impossible to discern using weak labels only, for instance, road and sky.

6.5.3 Training protocol

Stochastic gradient descent (SGD) with a minibatch of 20 images is used for training. The initial learning rate for the pretrained layers is set at 0.001, while the initial learning rate for newly added layers is set at .01. The momentum and the weight decay for gradient descent is set at 0.9 and .00005 respectively. We halve the learning rate after every 4000 iterations. All the networks are trained for 40000 iterations.

6.6 Experiments

We use a weighted combination of the objective based on fake labels and the neighborhood based objective in our experiments. For our comparisons, we have limited ourselves to models that do not employ pixel level information. Hence, we have excluded the models that employ agnostic segmentation or saliency masks that have been manually labelled (Hou et al., 2016, Wei et al., 2017) with the exception of STC (Wei et al., 2016).

$$\mathcal{L} = \mathbb{E}_{q^{\text{fake}}(\mathbf{z}|\mathbf{x},\mathbf{y})} \log p(\mathbf{z}|\mathbf{x}) - \lambda \text{KL}(p(\mathbf{z}|\mathbf{x})||p^{\text{neighb}}(\mathbf{z}|\mathbf{x}))\,.$$

The model is trained on the ImageNet subset and evaluated on the VOC 2012 validation set.
Hyperparameters: We use the same hyperparameter settings for the kernel as used in the publicly available code associated with the thesis (Krähenbühl and Koltun, 2011). In particular, we modify the code in (Krähenbühl and Koltun, 2011) to allow us to compute the weighted and exponentiated weighted mean. To account for the fact that our network reduces the segmentation mask to one-fourth of the input image, we divide θ_β and θ_γ by 4. during training.

We also evaluate the effect of the hyperparameter λ on the performance of the model. Note that the KL-divergence term will be minimized when all the pixels are assigned the same label. Hence, when higher weight is given to the KL-divergence term, all the pixels get assigned to the background class (since it is the most prominent class). In contrast if λ is close to 0, the boundaries of segmentation masks learnt by the model do not coincide with the boundaries of the objects. We vary the value of λ from 0 to 1 to study the effects of λ on the performance of the model. The results are shown in Figures 6.2. As can be observed, when weighted mean is used as the neighborhood distribution, the model remains quite stable to the choice of λ. However, the overall performance achieved is worse as compared to the model that relies on exponentiated weighted mean (51.6% vs 50.7%). The segmentation masks generated by using exponentiated weighted mean at $\lambda = .3$ are shown in Figure 6.2.

We also compare the proposed model against other models for weakly supervised segmentation on the VOC 2012 val set. The models used for comparison are (i) MIL+ILP (Pinheiro and Collobert, 2015). (ii) EM-Adapt (Papandreou et al., 2015), (iii) CCNN (Pathak et al., 2015), (iv) SEC (Kolesnikov and Lampert, 2016) and (v) STC (Wei et al., 2016). Among these models, SEC employs separate convolution networks for computing the saliency maps for foreground and background. On the other hand, the saliency network in STC is trained using dense (pixel-level) supervision. Hence, STC is not a weakly-supervised segmentation model in the true sense of the word. The results are given in Table 6.1. Note that the proposed model is similar in spirit to CCNN (Pathak et al., 2015) and EM-Adapt (Papandreou et al., 2015), with

class	MIL+ILP	EM-Adapt	CCNN	SEC	STC	Ours
background	77.2	67.2	68.5	82.4	84.5	**85.4**
aeroplane	37.3	29.2	25.5	62.9	68.0	**68.4**
bike	18.4	17.6	17.0	26.4	19.5	**28.3**
bird	25.4	28.6	25.4	61.6	60.5	**63.6**
boat	28.2	22.2	20.2	27.6	42.5	**42.9**
bottle	31.9	29.6	26.3	38.1	44.8	**54.4**
bus	41.6	47.0	46.8	66.6	**68.4**	62.7
car	48.1	44.0	47.1	62.7	**64.0**	62.9
cat	50.7	44.2	48.0	**75.2**	64.8	67.5
chair	12.7	14.6	15.8	**22.1**	14.5	10.6
cow	53.5	45.7	35.1	**53.5**	52.0	46.3
diningtable	14.6	24.9	21.0	28.3	22.8	**37.2**
dog	50.9	41.0	44.5	**65.8**	58.0	48.7
horse	44.1	34.8	34.5	**57.8**	55.3	53.8
motorbike	39.2	41.6	46.2	**62.3**	57.8	57.3
person	37.9	32.1	40.7	62.3	60.5	**64.7**
plant	28.3	30.4	24.8	32.5	40.6	**44.3**
sheep	44.0	36.3	37.4	**62.6**	56.7	58.2
sofa	19.6	24.0	22.2	32.1	23.0	**35.8**
train	37.6	38.1	38.8	45.4	**57.1**	43.6
tvmonitor	35.0	31.6	36.9	45.3	31.2	**47.4**
mean	36.6	33.8	35.3	50.7	49.8	**51.6**

Table 6.1: Results on PASCAL VOC 2012 (mIoU in %) *val* set for weakly supervised segmentation.

the exception of the incorporation of a neighborhood based objective. One can observe that the incorporation of neighborhood information results in drastically improved performance.

Finally, we evaluate the performance of our model on the VOC 2012 test set by submitting our results to the evaluation server of VOC 2012. An anonymous view of the results is available at *http://host.robots.ox.ac.uk:8080/anonymous/BEC3EB.html*. We achieve an accuracy of 52.01% which is the state-of-the-art for weakly supervised semantic segmentation of images on this dataset.

We also evaluate the speedup obtained by using the proposed model as compared to CRF-RNN (Zheng et al., 2015). Note that CRF-RNN was originally proposed for fully supervised semantic segmentation of images, and uses a recursive expansion of the mean-field distribution

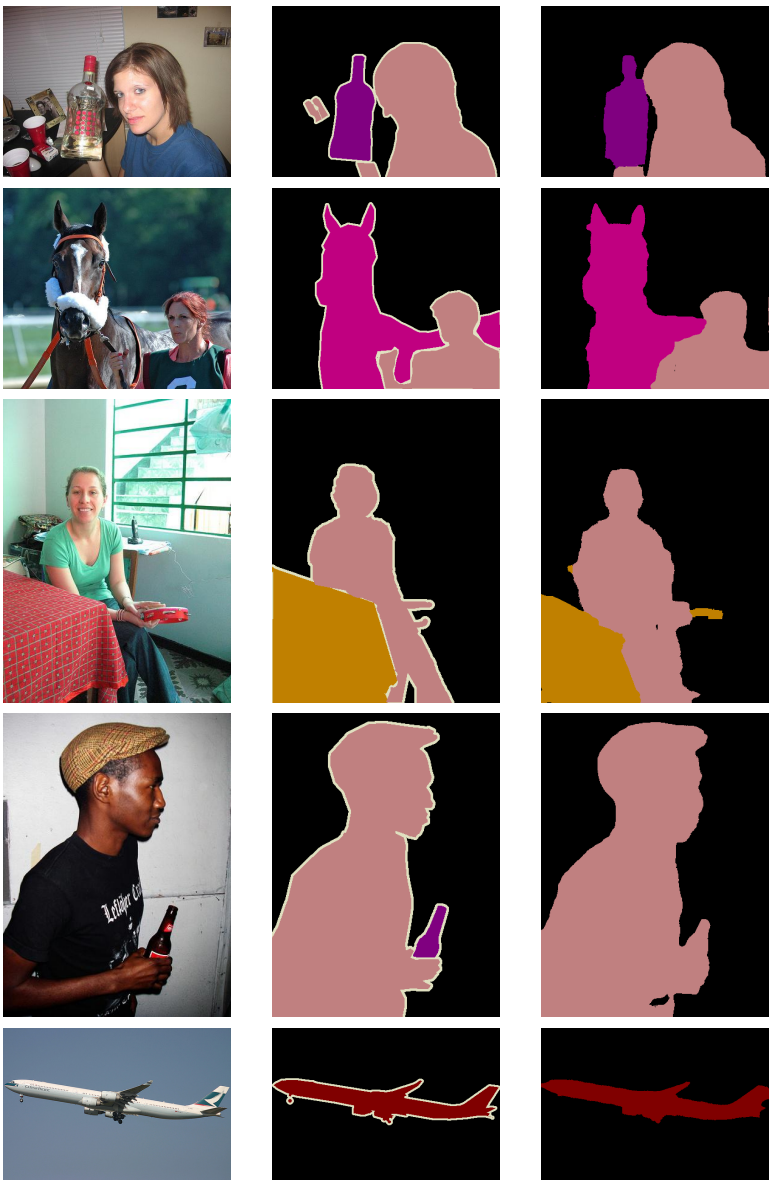


Table 6.2: Examples of predicted segmentation masks. The middle row is the ground truth. Note that the model has learnt to align the predicted boundaries with the true boundaries.

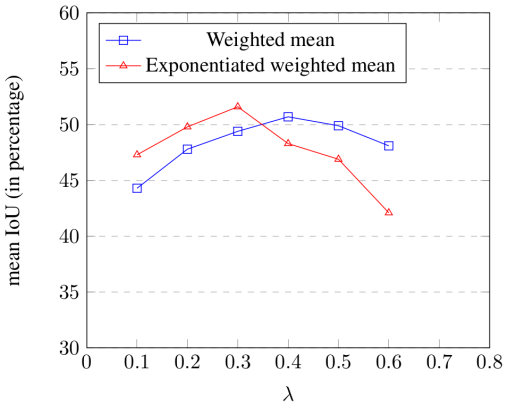


Figure 6.2: When weighted mean is used as neighborhood distribution

implemented using a recurrent neural network. For the sake of comparison, we replace the CNN architecture used in CRF-RNN with the CNN architecture used in our model. Due to memory constraints, we are forced to use a batch size of 1 image for CRF-RNN, while we use a batch size of 20 images for our model.

In order to modify the CRF-RNN for handling weakly annotated images, we generate fake labels for each image. The process for generating fake labels is exactly the same as used in our model. The learning rate and momentum used for training is exactly the same as suggested in (Zheng et al., 2015). The mean IoU on validation set as a function of the time required for training the model is plotted in Figure 6.3. As can be observed from the Figure, the proposed model achieves its asymptotic IoU within a few hours. In contrast, the mean IoU of CRF-RNN keeps improving, and even after 24 hours, the mean IoU of CRF-RNN was quite low as compared to our model. We aborted the training of the models after 24 hours.

Till now, we have used contrast sensitive Potts potential defined in terms of pixel-locations and brightness for modeling the neighborhood constraints. That is, we model the following criteria: If two pixels have similar brightness and lie close to each other, they are likely to have the same label. This function, which is a pairwise potential function, is used in most of the works that employ CRFs for semantic segmentation and has been shown to work quite well. In recent years, models that efficiently learn the pairwise potentials have been proposed for fully supervised semantic segmentation tasks.

Since the pairwise potentials use only pixel brightness and location for modeling, it is reasonable to wonder if the performance will improve if brightness is replaced by texture. Hence, we replace the features in Section 6.3.3 by the output of the first 2 convolutional layers of VGG-16. In particular, we concatenate the feature maps obtained after the first and second

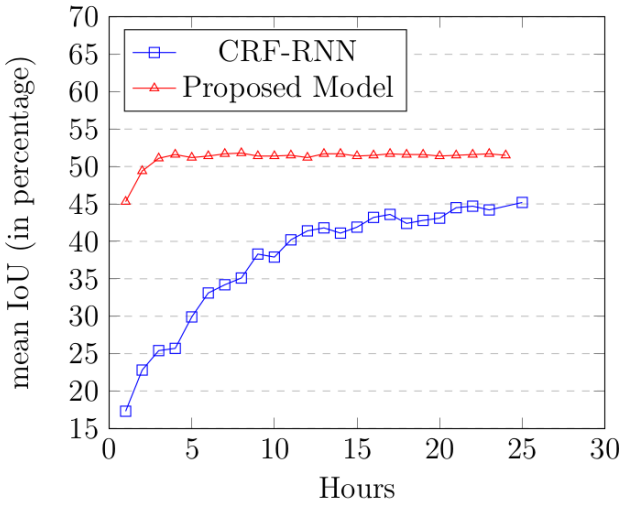


Figure 6.3: Improvement in performance with time. The proposed model for incorporating CRF reaches its asymptotic error in a few hours. In contrast, the error in CRF-RNN keeps decreasing even after 24 hours.

convolutional layers in VGG-16 to obtain a 128 dimensional vector for each pixel location. For each location i, this vector $h(x_i)$ captures the discriminative features in a local neighborhood surrounding the pixel x_i. The corresponding potential function is given by:

$$\begin{aligned} k(f_i(\mathbf{x}), f_j(\mathbf{x})) =& k_1(f_i(\mathbf{x}), f_j(\mathbf{x})) + k_2(f_i(\mathbf{x}), f_j(\mathbf{x})) \\ =& w_1 \exp\left(-\frac{|h(x_i) - h(x_j)|^2}{2\theta_\alpha^2} - \frac{|i-j|^2}{2\theta_\beta^2}\right) + w_2 \exp\left(-\frac{|i-j|^2}{2\theta_\gamma^2}\right), \end{aligned}$$

where, $f_i(\mathbf{x}) = [h(x_i), i]$. Here, $w_1, w_2, \theta_\alpha, \theta_\beta$ and θ_γ are hyperparameters that are fixed during training. Except for θ_α, all the other hyperparameters are assigned the same value as in the thesis. The mean IoU on Pascal VOC-2012 validation set for various values of θ_α is plotted in Figure 6.4.

As can be observed, the accuracy does not improve with the use of richer features for neighborhood potentials. This is expected since these features have been trained for classification and hence, give more weight to discriminative features such as edges. We hypothesize that if we intend to replace pixel-based pairwise potentials, we need to learn these potentials directly from the data as in (Lin et al., 2016).

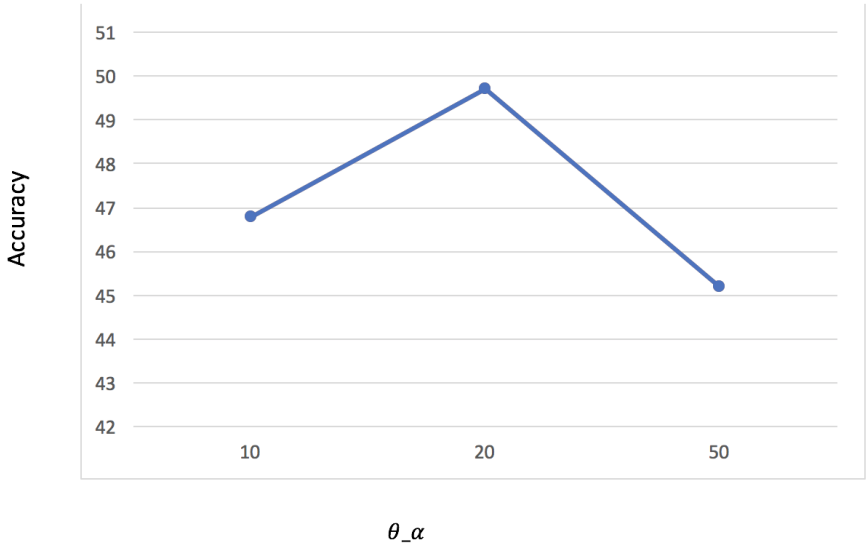


Figure 6.4: Mean IoU on Pascal VOC-2012 validation set for various values of θ_α

6.7 Discussion

In this thesis, we proposed a model for weakly supervised semantic segmentation of images that utilizes fake labels and conditional random fields. We showed that the output of the CNN can be forced to satisfy the constraints of a conditional random field, without explicitly evaluating the mean-field distribution at every step. We achieve this forcing the output distribution at every pixel to be close to its neighbors. As a consequence, we achieve significant performance improvement over traditional CNNs with negligible increase in training time.

We focused on weakly annotated images in this thesis, since CRFs can achieve drastic performance improvements for this task. When pixel-level information is available, forcing the pixel-level labels to be close to its neighbors will serve as an unnecessary and often over-smooth regularizer, unless only rough object boundaries are only available. A model capable of handling rough object boundaries, can drastically reduce the time required for manually generating segmentation masks. We intend to explore the utility of the model for handling rough boundaries in the future.

Subsequent to the submission of this thesis there have been several more works on weakly supervised semantic segmentation, that have pushed the performance of weakly supervised models to be very close to the fully supervised models (Wang et al., 2018, Tang et al., 2018, Ahn and Kwak, 2018, Huang et al., 2018). Most of these methods are combinations of multiple heuristics that are known to work well individually. We list some of the heuristics below:

1. Initialization with class action maps - This is a heuristic used in almost all the weakly supervised works to obtain a coarse segmentation network.
2. Segmentation into superpixels.
3. Grabcut for refinement.
4. Seed region growing.
5. Using a set of filtered web images for aiding the segmentation network.
6. Using supervised saliency masks for refining segmentation.

These are all well-known heuristics. Hence, it is no surprise that a combination of one or more of these results in improved performance. The novelty in many of the papers is determining a good technique for incorporating these heuristics. Hence, if the aim is to outperform the current state-of-the-art, a combination of these heuristics among others will be helpful.

However, we find the idea of learning affinity between pixels, proposed in (Ahn and Kwak, 2018) to be a much more appealing direction, rather than determining the correct combination of known heuristics. Affinity among pixels correlates with the binary potentials of a CRF, and is generally assumed to be fixed during training. Recently, a few models have been proposed that learn this affinity as part of CRF training (Lin et al., 2016). However, there is no known mechanism to learn these affinities in a weakly supervised domain. Hence, the work in (Ahn and Kwak, 2018) provides a good starting point in this respect.

Chapter 7

Concluding Remarks

Prior to the advent of deep learning, most models for machine learning on images, speech and text datasets incorporated a feature extraction stage, that involved the expertise of researchers in a specific domain. For most tasks, deep neural architectures have replaced the feature extraction stage, thereby diminishing the necessity of domain experts for learning useful features from raw data. However, training these architectures often requires access to a large amount of labeled data. Hence, the need for domain experts for learning useful features has largely been replaced by the need for individuals that can label a large amount of data. Depending upon the underlying problem, the individuals required may or may not be experts in a specific area. In this thesis, we have attempted to reduce the reliance on manually generated supervision for machine learning tasks, thereby taking another tiny step towards a completely automated future.

The primary focus of this thesis has been to propose models that minimize the amount of supervision required for training deep architectures. In particular, we have attempted to address the problems of learning with limited, unlabeled and weakly-labeled data by employing architectures that have been inspired from deep networks. For limited data, we proposed a model that randomly generates a large number of features from the input, and incorporates those features for the final supervised learning task via the kernel trick. The features are obtained by passing the input through neural networks with random weights for several iterations. The resultant models, referred to as stretched deep networks, perform well for object recognition tasks, when the amount of labeled data is limited, but are easily outperformed by fully supervised CNNs with increase in labeled data. Furthermore, the inability of the model to incorporate the weights from pre-trained networks, severely limits the applicability of these models.

This brings us to the second important contribution of this thesis - discriminative encoders.

Discriminative encoders are probabilistic models that specify a joint distribution over the observed input and the unobserved latent features, without modeling the actual distribution of the input. This property distinguishes discriminative encoders from other models for specifying joint distributions. Discriminative encoders are specified by the prior distribution on the latent features, and the encoding distribution of the latent features given the input. By employing neural networks for specifying the distributions, the model addresses all the concerns of stretched deep networks, while having the added advantage of incorporating unlabeled data for learning. Using categorical latent features, we achieved improved performance for unsupervised and semi-supervised classification for several datasets.

Furthermore, as is the case with other probabilistic models, one can specify several choices of prior distributions on the latent features. In this thesis, we employed hierarchical Bayesian distributions such as Gaussian mixture and Gaussian topic models as priors on the latent features of a discriminative encoder. Topic models employ the 'bag of objects' structure of data for learning the topics associated with each data point in a collection. In addition, one can learn sentence embeddings by employing autoregressive priors to capture the sequential structure of sentences. We intend to explore the successes and failures of discriminative encoders for several interesting choices of priors in the future.

Finally, we addressed the problem of semantic segmentation from weakly annotated images. In particular, we trained a CNN to predict segmentation masks, without employing any pixel-level supervision. We achieved this by generating fake pixel-level supervision from the output of the CNN and image-level labels, while simultaneously forcing the output of the CNN to satisfy the constraint imposed by a conditional random field. The CRF forces the CNN to generate segmentation masks with sharp boundaries that correlate well with the true boundaries. As a result, we were able to achieve state-of-the-art performance for semantic segmentation of images without pixel-level supervision.

An interesting use case for unsupervised learning is in robotics for building general purpose AI agents. For an AI agent to perform a single task such as Fetching a glass of water, the agent needs to perform tens of thousands of actions in a sequence (these include the motion of the hundreds of gears in such an agent). Except for extremely limited environments, generating supervision for these actions for all states of the environment is impossible. Hence, reward-based reinforcement learning is used to train such models. However, the rewards are quite sparse, and hence, provide very weak signals for training.

Imitation learning, which is a form of supervised learning, has been used with success for a few domains such as autonomous car driving. However, cars have a simple purpose, to allow people to reach from one place to another with safety and comfort. Hence, the collection of

data for such problems is quite straightforward. Even for these limited domains, millions of hours of data need to be collected over several years[1].

In contrast, a general-purpose AI agent must be able to perform several tasks for which there is very limited supervision. Hence, for such problems, weakly supervised learning and semi-supervised learning will prove to be necessary.

It is worthy to point out that except for stretched deep networks, all our proposed models are probability models parametrized by neural networks. Hence, these architectures combine the interpretability of probabilistic modelling with the representation power of deep neural networks to achieve improved performance with minimal supervision. Going forward, it appears that the key to combat the scarcity of labeled data, is to employ probability models for modeling the missing information, while simultaneously utilizing neural networks to parametrize these probability distributions.

[1] https://devblogs.nvidia.com/training-self-driving-vehicles-challenge-scale/

Bibliography

Adams, A., J. Baek, and M. A. Davis (2010). Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum 29*(2), 753–762. 94

Ahn, J. and S. Kwak (2018). Learning pixel-level semantic affinity with image-level supervision for weakly supervised semantic segmentation. 110, 111

Amodei, D., S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al. (2016). Deep speech 2: End-to-end speech recognition in English and Mandarin. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pp. 173–182. 5

Bay, H., T. Tuytelaars, and L. Van Gool (2006). SURF: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 404–417. Springer. 1

Bearman, A., O. Russakovsky, V. Ferrari, and L. Fei-Fei (2016). Whats the point: Semantic segmentation with point supervision. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 549–565. Springer. 92

Bengio, Y., P. Lamblin, D. Popovici, H. Larochelle, et al. (2007). Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems (NIPS)*. 1, 20, 29

Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research (JMLR) 3*, 993–1022. 44, 63, 79, 88

Bo, L., X. Ren, and D. Fox (2013). Multipath sparse coding using hierarchical matching pursuit. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 660–667. 1

Bowman, S. R., L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio (2016). Generating sentences from a continuous space. In *Proceedings of the SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pp. 10. 47

Boykov, Y. Y. and M.-P. Jolly (2001). Interactive graph cuts for optimal boundary & region segmentation of objects in ND images. In *Proceedings of the Eighth International Conference on Computer Vision (ICCV)*, Volume 1, pp. 105–112. IEEE Computer Society. 94

Bylinskii, Z., T. Judd, A. Borji, L. Itti, F. Durand, A. Oliva, and A. Torralba (2015). MIT Saliency Benchmark. 7

Cai, D., X. He, and J. Han (2011). Speed up kernel discriminant analysis. *The International Journal on Very Large Databases 20*(1), 21–33. 29, 37

Chapelle, O., B. Scholkopf, and A. Zien (2009). Semi-supervised learning. *IEEE Transactions on Neural Networks 20*(3), 542–542. 3

Chen, L.-C., G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille (2016). DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *arXiv preprint arXiv:1606.00915*. 93, 94, 95, 101

Cho, Y. and L. K. Saul (2009). Kernel methods for deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 342–350. 9, 25, 28, 29, 41

Coates, A., A. Ng, and H. Lee (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 215–223. 1, 31, 38

Dai, J., K. He, and J. Sun (2015). BoxSup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1635–1643. 92

Dalal, N. and B. Triggs (2005). Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Volume 1, pp. 886–893. IEEE. 4, 24

Das, R., M. Zaheer, and C. Dyer (2015). Gaussian lda for topic models with word embeddings. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*. 80

Decoste, D. and B. Schölkopf (2002). Training invariant support vector machines. *Machine learning 46*(1), 161–190. 29

Donahue, J., P. Krähenbühl, and T. Darrell (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*. 23

Dosovitskiy, A., J. T. Springenberg, M. Riedmiller, and T. Brox (2014). Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 766–774. 6

Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR) 12*(Jul), 2121–2159. 16

Dumoulin, V., I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville (2016). Adversarially learned inference. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 23

Everingham, M., L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html. 5, 103

Fei-Fei, L., R. Fergus, and P. Perona (2007). Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding 106*(1), 59–70. 37

Fei-Fei, L. and P. Perona (2005). A Bayesian hierarchical model for learning natural scene categories. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Volume 2, pp. 524–531. IEEE. 79

Fischer, A. and C. Igel (2012). An introduction to restricted Boltzmann machines. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, 14–36. 19

Goodfellow, I. et al. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2672–2680. 9, 22, 58, 71

Goodfellow, I., M. Mirza, A. Courville, and Y. Bengio (2013). Multi-prediction deep Boltzmann machines. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 548–556. 6

Goodfellow, I. J., J. Shlens, and C. Szegedy (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*. 56

Grandvalet, Y. and Y. Bengio (2005). Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 529–536. 3, 6

Hadsell, R., S. Chopra, and Y. LeCun (2006). Dimensionality reduction by learning an invariant mapping. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Volume 2, pp. 1735–1742. IEEE. 87

Hannun, A., C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng (2014). Deep Speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*. 5

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation 14*(8), 1771–1800. 1, 6, 18, 19, 44

Hinton, G. E., S. Osindero, and Y.-W. Teh (2006). A fast learning algorithm for deep belief nets. *Neural computation 18*(7), 1527–1554. 29

Hofmann, T. (1999). Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 289–296. Morgan Kaufmann Publishers Inc. 79, 88

Hofmann, T., B. Schölkopf, and A. J. Smola (2008). Kernel methods in machine learning. *The Annals of Statistics*, 1171–1220. 24

Hou, Q., P. K. Dokania, D. Massiceti, Y. Wei, M.-M. Cheng, and P. Torr (2016). Mining pixels: Weakly supervised semantic segmentation using image labels. *arXiv preprint arXiv:1612.02101*. 103, 105

Huang, Z., X. Wang, J. Wang, W. Liu, and J. Wang (2018). Weakly-supervised semantic segmentation network with deep seeded region growing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 110

Hubel, D. H. and T. N. Wiesel (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology 160*(1), 106–154. 3

Huiskes, M. J., B. Thomee, and M. S. Lew (2010). New trends and ideas in visual concept detection: the MIR Flickr retrieval evaluation initiative. In *Proceedings of the International Conference on Multimedia Information Retrieval*, pp. 527–536. ACM. 103

Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 448–456. 14

Itti, L., C. Koch, and E. Niebur (1998). A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 20*(11), 1254–1259. 7

Jarrett, K., K. Kavukcuoglu, Y. LeCun, et al. (2009). What is the best multi-stage architecture for object recognition? In *Proceedings of 12th International Conference on Computer Vision (ICCV)*, pp. 2146–2153. IEEE. 37, 38, 40

Kingma, D. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 17, 53

Kingma, D. P., S. Mohamed, D. J. Rezende, and M. Welling (2014). Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3581–3589. 6, 70

Kingma, D. P. and M. Welling (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*. 6, 21, 44, 70

Kohli, P., P. H. Torr, et al. (2009). Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision (IJCV) 82*(3), 302–324. 94

Kolesnikov, A. and C. H. Lampert (2016). Seed, expand and constrain: Three principles for weakly-supervised image segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 695–711. Springer. 7, 93, 105

Kraehenbuehl, P. and V. Koltun (2013). Parameter learning and convergent inference for dense random fields. In *Proceedings of The 30th International Conference on Machine Learning (ICML)*, pp. 513–521. 102

Krähenbühl, P. and V. Koltun (2011). Efficient inference in fully connected CRFs with Gaussian edge potentials. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems (NIPS)*, pp. 109–117. Curran Associates, Inc. 92, 94, 95, 98, 99, 101, 105

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105. 1, 5

Landauer, T. K. and S. T. Dumais (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review 104*(2), 211. 79

Landauer, T. K., P. W. Foltz, and D. Laham (1998). An introduction to latent semantic analysis. *Discourse processes 25*(2-3), 259–284. 63

Larochelle, H., D. Erhan, A. Courville, J. Bergstra, and Y. Bengio (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine learning (ICML)*, pp. 473–480. ACM. xii, 28, 29

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation 1*(4), 541–551. 5

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324. 30, 31, 59

LeCun, Y., F. J. Huang, and L. Bottou (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Volume 2, pp. II–104. IEEE. 40

Lee, H., A. Battle, R. Raina, and A. Y. Ng (2007). Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 801–808. 1

Lee, H., R. Grosse, R. Ranganath, and A. Y. Ng (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on machine learning (ICML)*, pp. 609–616. ACM. 19

Lempitsky, V., P. Kohli, C. Rother, and T. Sharp (2009). Image segmentation with a bounding box prior. In *Proceedings of the 12th International Conference on Computer Vision (ICCV)*, pp. 277–284. IEEE. 92

Lin, G., C. Shen, A. van den Hengel, and I. Reid (2016). Efficient piecewise training of deep structured models for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3194–3203. 101, 109, 111

Long, J., E. Shelhamer, and T. Darrell (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440. 5

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV) 60*(2), 91–110. 1, 4, 24

Lu, Y., Q. Mei, and C. Zhai (2011). Investigating task performance of probabilistic topic models: an empirical study of pLSA and LDA. *Information Retrieval 14*(2), 178–203. 88

Maaløe, L., C. K. Sønderby, S. K. Sønderby, and O. Winther (2016). Auxiliary deep generative models. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, ICML'16, pp. 1445–1454. 66, 70

Mairal, J. (2016). End-to-end kernel learning with supervised convolutional kernel networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1399–1407. 41

Mairal, J., P. Koniusz, Z. Harchaoui, and C. Schmid (2014). Convolutional kernel networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2627–2635. 41

Makhzani, A., J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey (2015). Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*. 59, 61

Nair, V. and G. E. Hinton (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on machine learning (ICML)*, pp. 807–814. 19

Neal, R. M. (2012). *Bayesian learning for neural networks*, Volume 118. Springer Science & Business Media. 41

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. 64

Ng, A. Y. and M. I. Jordan (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 841–848. 3, 6

Noroozi, M. and P. Favaro (2016). Unsupervised learning of visual representations by solving jigsaw puzzles. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 69–84. Springer. 7

Oh Song, H., Y. Xiang, S. Jegelka, and S. Savarese (2016). Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4004–4012. 86, 87

Pandey, G. and A. Dukkipati (2014a). Learning by stretching deep networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pp. 1719–1727. 25, 41

Pandey, G. and A. Dukkipati (2014b). To go deep or wide in learning? In *Proceedings of the Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 724–732. 9, 28

Pandey, G. and A. Dukkipati (2017). Unsupervised feature learning with discriminative encoder. In *Proceedings of the International Conference on Data Mining (ICDM)*. 45

Papandreou, G., L.-C. Chen, K. Murphy, and A. L. Yuille (2015). Weakly-and semi-supervised learning of a DCNN for semantic image segmentation. *arXiv preprint arXiv:1502.02734*. 92, 105

Pathak, D., R. Girshick, P. Dollár, T. Darrell, and B. Hariharan (2016). Learning features by watching objects move. *arXiv preprint arXiv:1612.06370*. 7

Pathak, D., P. Krahenbuhl, and T. Darrell (2015). Constrained convolutional neural networks for weakly supervised segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1796–1804. 93, 105

Pathak, D., E. Shelhamer, J. Long, and T. Darrell (2014). Fully convolutional multi-class multiple instance learning. *arXiv preprint arXiv:1412.7144*. 92

Pinheiro, P. O. and R. Collobert (2015). From image-level to pixel-level labeling with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1713–1721. 92, 93, 103, 105

Rahimi, A. and B. Recht (2009). Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1313–1320. 25

Rasmus, A., M. Berglund, M. Honkala, H. Valpola, and T. Raiko (2015). Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3546–3554. 65

Rezende, D. J., S. Mohamed, and D. Wierstra (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of The 31st International Conference on Machine Learning (ICML)*, pp. 1278–1286. 44

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group (Eds.), *Parallel Distributed Processing*, Chapter Learning internal representations by error propagation, pp. 318–362. MIT Press. 5

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*(3), 211–252. 5, 104

Salakhutdinov, R. and G. Hinton (2009). Deep Boltzmann machines. In *Proceedings of the Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 448–455. 1, 6

Saleh, F., M. S. A. Akbarian, M. Salzmann, L. Petersson, S. Gould, and J. M. Alvarez (2016). Built-in foreground/background prior for weakly-supervised semantic segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 413–432. Springer. 7, 103

Salimans, T., I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen (2016). Improved techniques for training GANs. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems (NIPS)*, pp. 2234–2242. Curran Associates, Inc. 14, 58, 63, 64, 65, 66, 71

Salimans, T. and D. P. Kingma (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 901–909. 14

Schroff, F., D. Kalenichenko, and J. Philbin (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823. 87

Schwing, A. G. and R. Urtasun (2015). Fully connected deep structured networks. *arXiv preprint arXiv:1503.02351*. 93, 101, 102

Shi, J. and J. Malik (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 22*(8), 888–905. 88

Shimoda, W. and K. Yanai (2016). Distinct class-specific saliency maps for weakly supervised semantic segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 218–234. Springer. 7

Shotton, J., J. Winn, C. Rother, and A. Criminisi (2006). Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 1–15. Springer. 94

Springenberg, J. T. (2015). Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 6, 55, 56, 61, 65, 71

Sutton, C. and A. McCallum (2005). Piecewise training for undirected models. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 568–575. AUAI Press. 101

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9. 1, 5

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826. 56

Tang, M., A. Djelouah, F. Perazzi, Y. Boykov, and C. Schroers (2018). Normalized cut loss for weakly-supervised cnn segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 110

Vezhnevets, A., V. Ferrari, and J. M. Buhmann (2011). Weakly supervised semantic segmentation with a multi-image model. In *Proceedings of IEEE Conference on Computer Vision (ICCV)*, pp. 643–650. IEEE. 92

Wah, C., S. Branson, P. Welinder, P. Perona, and S. Belongie (2011). The Caltech-UCSD Birds-200-2011 dataset. 85

Wang, X. and A. Gupta (2015). Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2794–2802. 7

Wang, X., S. You, X. Li, and H. Ma (2018). Weakly-supervised semantic segmentation by iteratively mining common object features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 110

Wei, Y. et al. (2016). STC: A simple to complex framework for weakly-supervised semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*. 7, 93, 103, 105

Wei, Y., J. Feng, X. Liang, M.-M. Cheng, Y. Zhao, and S. Yan (2017). Object region mining with adversarial erasing: A simple classification to semantic segmentation approach. *arXiv preprint arXiv:1703.08448*. 105

Wei, Y., X. Liang, Y. Chen, Z. Jie, Y. Xiao, Y. Zhao, and S. Yan (2016). Learning to segment with image-level annotations. *Pattern Recognition 59*, 234–244. 93

Williams, C. K. (1997). Computing with infinite networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 295–301. 41

Wilson, A. G., Z. Hu, R. Salakhutdinov, and E. P. Xing (2016). Deep kernel learning. In *Proceedings of the Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 370–378. 42

Xie, J., R. Girshick, and A. Farhadi (2016). Unsupervised deep embedding for clustering analysis. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*, pp. 478–487. 55, 61

Xu, J., A. G. Schwing, and R. Urtasun (2015, June). Learning to segment under various forms of weak supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 92

Yang, J., D. Parikh, and D. Batra (2016). Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5147–5156. 55

Yu, F. and V. Koltun (2015). Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*. 93

Zeiler, M. and R. Fergus (2013). Stochastic pooling for regularization of deep convolutional neural networks. In *Proceedings of the International Conference on Learning Representation (ICLR)*. 30

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2017). Understanding deep learning requires rethinking generalization. In *Proceedings of the International Conference on Learning Representations*. 24

Zheng, S., S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr (2015). Conditional random fields as recurrent neural networks. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 1529–1537. 5, 93, 94, 95, 101, 102, 106, 108

Zhou, B., A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2921–2929. 7