

An Investigation Into the Effectiveness of Latent Variable Models for Domain Adaptation

by

Xuanrui Zeng

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2025

© Xuanrui Zeng 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The proliferation of machine learning with neural networks (NNs) has revolutionized fields such as computer vision and natural language processing. However, their successes often overshadow two important weaknesses of neural networks: (i) their reliance on large amounts of training data and (ii) the assumption of independent and identically distributed (i.i.d.) data. Because of these weaknesses, the vast majority of NNs today are application-specific machineries tuned to one task and one data domain.

This thesis investigates the effectiveness of a latent variable model for unsupervised domain adaptation, aiming to bridge the gap between two different data distributions while leveraging only labeled data samples from one, and unlabeled data samples from the other. A novel generative modeling framework is proposed to address this problem, incorporating recent advances in probabilistic modeling and variational inference techniques from the neural network literature.

Empirical results of the proposed approach seem promising, and indicates adequate transfer of the labeling knowledge of the model across disparate data domains without requiring manual re-labeling or domain-specific adjustments. Moreover, the proposed approach has also shown potentials in solving the related domain translation problem. Despite these fortunes, the existing approach has shown limitation in solving more complex scenarios of unsupervised domain adaptation, specifically those involving more vibrant differences between domains.

Acknowledgements

I'm grateful to my mentor and supervisor, Professor Christopher Nielsen, who have provided me with continuous support and guidance throughout this academic journey and whose rigorous thinking I continue to admire; my colleague Alex Lovi for sharing his expertise on various topics related to my research and have helped me many times on my learning.

I also owe my gratitude to Professor Roydon Fraser, Jaden Yoo, and members from the UWAFIT student design team in Year 2023, who have supported my initiatives to take on independent research.

Lastly, I thank my family and friends for their continuous support. They have always shed a light upon me regardless of where I am and what I do.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Motivation	1
1.2 Related Works	4
1.3 Contributions and Organizations	7
2 Background	8
2.1 Definitions and Notations	8
2.2 Supervised Learning	15
2.3 Domain Shift	20
3 Problem Definition	24
3.1 Unsupervised Domain Adaptation	24
3.2 Alternate Problem	26

4	Approach	31
4.1	Motivation	31
4.2	Method	34
4.3	Implementation	39
5	Empirical Results	46
5.1	Setup of Experiment	46
5.2	Results	50
5.3	Implementation Details	54
6	Conclusion and Future Work	58
	References	60
	APPENDICES	67
A	Derivation of Variational Lower-Bounds	68
B	List of Tuning Parameters	70

List of Figures

1.1	Motivating camera example for UDA	2
2.1	An illustration of domain shift	21
2.2	Training in the presence of domain shift	23
3.1	Toy example comparing discriminative and generative modeling for the purpose of UDA	30
4.1	Illustration of a domain invariant representation	32
4.2	Illustration of a domain specific representation	35
4.3	Probabilistic graphical model of our framework	38
4.4	Overall architecture of our proposed framework	45
5.1	Demonstration of digit classification in our experiments	47
5.2	Illustration of the datasets used in our experiments	48
5.3	Results of our UDA approach on domain translation	53

List of Tables

1.1	Overview of UDA approaches in the literature	4
5.1	Results of our UDA approach	51
5.2	Construction of the neural networks f_{ϕ}^h and f_{ϕ}^z in our framework.	55
5.3	Construction of the neural networks g_{θ} and f_{ϕ}^y in our framework.	56
B.1	Ranges of tuning parameters used in our experiment for our approach.	70

Chapter 1

Introduction

In recent years, the number of successful applications of deep learning has increased rapidly. Computationally difficult problems in the areas of image recognition, image segmentation, natural language processing, and speech-to-text, have been solved to varying degree by artificial neural networks (NNs) with promising results. If the task or data domain changes, the NN will require re-training on the new task with the newly collected data samples. This is a significant limitation, as it is often the case that training data is scarce, and the cost of collecting and labeling data is high. In this thesis, we focus on the case of a single classification task with two data domains, for which there is some amount of labeled data collected for one domain, but only unlabeled data collected for the other. We use this scenario to investigate the following question: *How can we obtain a performant model on an unlabeled data domain, using labeled data from a different but related domain?*

1.1 Motivation

To motivate the above problem, consider the example of developing an object detection model for an autonomous vehicle. To train such a model, images of the street scenes are collected, after which labeling is done by some human labor to draw bounding boxes on the collected images, which together generates a labeled dataset to train the model. Clearly, the most expensive stage of this process is the manual labeling of the collected street scenes, since the collection of the images alone can be achieved autonomously. Suppose we now attempt to deploy this model on another vehicle with a different camera, where the intrinsic camera parameters differ from the previous one, causing differences in the perspective, lighting, and coloring in the images collected between the old and the new

camera, e.g. see Figure 1.1. In this case, most NN models will suffer a performance degradation when deployed to accept input images from this new camera, and it can be shown empirically that the degradation worsens with more differences in the imaging [1]. A naive solution to this problem would be to collect additional images on the new camera and then redo the labeling, but this will be extremely costly, mostly because of the manual re-labeling on the new images, and will certainly not scale to mass production where many different vehicle specifications exist. It is thus highly desirable to instead have methods that can adapt the model to the new images using the labeled old dataset, and an unlabeled new dataset consisting of images from the new camera, so that the manual re-labeling can be avoided.



(a) Image from the old camera, where the resolution is lower, the image is noisier, and the field-of-view is narrower. The image also lacks color vibrancy due to low dynamic range.

(b) Image from the new camera, with an increase in resolution, image quality and sharpness, wider field-of-view, and more vibrant colors.

Figure 1.1: Motivating camera example for UDA.

The aforementioned problem is called the *unsupervised domain adaptation* (UDA) problem, where we refer to the different distributions of the input data as *domains*, and the difference in the distributions of the input data as *domain shifts*. The domain for which there exists plenty of labeled data is named the *source domain*, whereas the domain that lacks labeled data but in which we hope to deploy the model, is named the *target domain*. In general, UDA is concerned with finding ways to achieve good performance of the model on the target domain without assumptions on the specific type of data involved, i.e., we wish to avoid using application-specific expert knowledge to aid the model in transitioning from one domain to the other. In the context of our vehicle example, this means finding a good model for the new camera without using our knowledge of the difference between the intrinsic parameters of the cameras.

1.1.1 Example Applications

Solutions to the UDA problem can be used for a wide range of applications, with the most relevant ones being medical imaging, computer vision, robotics, and sentiment analysis.

Medical Imaging Different medical equipments exhibit different specifications and device settings that can impact imaging results. UDA can be applied to overcome the performance gap between models trained on one source of data and deployed to use another source without redoing the labeling, nor analysis of the different characteristics of the equipment to seek ways that normalizes the data across the different equipments [2, 3].

Computer Vision A big portion of modern applications of NNs lies in computer vision, where numerous open-source datasets exist for a variety of applications. However, all datasets contain some biases that result from the data collection process due to different sensors being used and different demographic of where the collection is taking place. UDA can be used to adapt a model trained on open-source datasets to some private datasets, voiding the need of labeling on the private datasets [4, 5].

Robotics In robotics, it is usually much more cost effective to collect or synthesize training data from a simulation environment instead of the real world. In many cases, data collected this way can also be automatically labeled, which significantly reduces development costs. However, these data is limited by the fidelity of the simulation environment, and a model trained upon them under-performs when deployed in the real world due to domain shift. Research of how to bridge this domain gap is termed *Sim2Real*, and UDA can be applied in Sim2Real settings where some unlabeled data from the real world are available [6, 7].

Sentiment Analysis Sentiment analysis is the task of predicting the sentiment of natural language and is crucial in building recommender systems. Many datasets for sentiment analysis are collected inexpensively from online rating websites, where labels can be easily extracted from the user ratings. However, the demographic and language habits of human differs significantly outside the context of rating websites, and this makes model trained on the former datasets worse at correctly predicting the sentiment of languages outside of these websites. UDA can be used to bridge this gap by using some unlabeled sentences collected off the internet [8, 9].

1.2 Related Works

Over the past decade, many different approaches to tackling the UDA problem have been proposed in the literature. In this section, we provide an overview of them as well as their categorization. Overall, most UDA approaches to-date can be categorized into one of the following categories: re-weighting, domain alignment, domain translation, and representation learning, with some hybrid approaches utilizing ideas from more than one category. Table 1.1 shows an overview of the related works.

Category	References
Re-weighting	[10, 11, 12, 13]
Domain Alignment	[14, 15, 16, 17, 18, 6]
Domain Translation	[19, 20, 7, 21, 22]
Representation Learning	[23, 24, 25]

Table 1.1: Overview of UDA approaches in the literature. The cited works here are representative UDA approaches within each category, and the list is non-exhaustive.

Re-weighting Re-weighting based approaches solve the UDA problem by re-weighting the per-sample loss such that the expected loss being estimated are adjusted to be with respect to the target domain. The observation that underlies re-weighting approaches is that the source expected loss can be adjusted into the target expected loss by scaling the loss on each data sample by the likelihood ratio of the data sample, where the likelihood ratio can be estimated solely from unlabeled source and target samples.

Different re-weighting approaches mostly differ in their ways to estimate the unknown likelihood ratio. The authors of [10, 11] propose to perform density estimation of the unlabeled source and target samples separately, after which the likelihood ratio can be computed from the ratio of the densities. Meanwhile the authors of [12, 13, 10] propose ways to directly estimate the likelihood ratio without resorting to separate density estimation on source and target samples.

Re-weighting based approaches have demonstrated successes in cases where the dimension of the input data is low (less than 20). In cases of high dimension, the distribution of the input data often shares little support over the input space across the domains, resulting in small likelihood ratio for most source samples. To work with high-dimensional data, dimensionality reduction techniques such as PCA [26] are required as a pre-processing step, but this leads to information loss for label predictions when the label predictive function

is non-linear. Because of this, re-weighting based approaches have become less relevant in today’s supervised learning with NNs, where the input data is typically of high dimension.

Domain Alignment Domain alignment based approaches to the UDA problems have become the mainstream approaches over the last decade in NNs. Domain alignment approaches seek to find a map from the input space to some representation space such that the distributions of the representation are similar across domains. The question of what constitutes of a good representation for label predictions has been formally analyzed in [27, 28], where it is proven that the target risk is upper-bounded by a combination of the source risk, a uniquely defined distance between the distributions of the source and target representation, and the optimal combined risk on both domains. Different domain alignment approaches are mostly different realizations of this theory. Since the optimal combined risk is not tractable in practice without access to target labels, most work in this category assumes the optimal combined risk to be low and proceeds with ways to simultaneously minimize the source expected loss and the distributional difference of the source and target representation.

The typical setup for domain alignment is to divide an NN into two components, a *feature encoder* which is a function from the input space to a representation space, and a *label predictor* which is another function from the representation space to the label space. The objective for training the NN usually consists of minimization of the source expected loss along with a regularization term that attempts to minimize difference in representation distributions across the domains. Different domain alignment approaches differ mostly in their implementation of this regularization term. In [14, 15], the regularization is formed by penalizing differences in batch statistics on the representation output by the encoder. Instead of relying on matching statistics, [16, 17] proposes to minimize the *maximum mean discrepancy* (MMD) [29] between the source and target distribution, which enables matching of higher-order statistics between the distributions.

Following the invention of generative adversarial networks (GANs) [30], the use of adversarial training for the purpose of UDA has demonstrated successes. In such cases, an additional NN called the *domain discriminator* that attempts to classify the domain of the representation is introduced. And the domain discriminator is optimized to achieve good classification of the domain on the representation generated from the encoder, while the encoder is encouraged to confuse the discriminator into predicting the wrong domain. In this line of work, [31, 18] has first introduced adversarial training for solving UDA problems; [14] proposes a three-stage training procedure that separates training on source samples and domain adaptation into different stages; [6] proposes a rather different adversarial

training setup that uses two separate label predictor NNs in conjunction to detect target representations that differs from the source, and trains the encoder adversarially to avoid such representations.

Domain Translation Domain translation methods address the UDA problem by transforming input data from the source domain to resemble data from the target domain, or vice versa, thereby reducing the domain gap at the input level. The use of domain translation to solve UDA problems has gained popularity following the work of [19] for unpaired image-to-image translation, where one NN is trained to map source samples into target samples and another NN trained vice versa. A novel cycle consistency loss is introduced to regularize the two NNs to be inverse of each other. With this, the authors have shown successes in achieving domain translation without access to paired input data across the domains, allowing it to be used for UDA. Since then, [20, 7] have applied this idea onto UDA with promising results in the computer vision literature. The authors of [21] improve upon this idea by making a shared latent space assumption between the two NNs along with probabilistic modeling of the latent space using VAE [32]. In [22], further improvements are made through disentangling the latent space into a domain-invariant and a domain-specific portion. Empirical results from these works have shown strong performance for computer vision related tasks.

Representation Learning Compared to the other categories of solutions to the UDA problem, representation learning (RL) based UDA approaches are far less uniform and vary significantly across the literature. One commonality of RL approaches is that they typically involve the reconstruction of the source and target input data, similar to that of an *autoencoder*, which are reconstruction-based NNs that have been empirically shown to extract useful representations of the input data in some low dimensional space that capture essential features of the input data [33, 34, 32].

In this line of work, [23] proposes a scheme where the representation output from a feature encoder network is trained to simultaneously achieve low source error by a downstream label predictor network and low reconstruction error by a downstream decoder. The authors of [24] follow a similar scheme but the representation is further divided into domain-shared and domain-specific components. In [25], a novel GAN-based architecture for UDA is proposed in which two generator-discriminator pairs are used to generate in-domain samples for each domain, with shared layers across the two generators and discriminators to facilitate learning of a joint representation across the domains.

1.3 Contributions and Organizations

To the best of our knowledge, the isolated use of generative modeling to solve the UDA problem has not been widely explored in the literature. Although most works in representation learning, including [23, 24], can be seen as frameworks that contain generative components, they utilize the generative components solely for the purpose of learning a good representation of the input that can be fed downstream into a label predictor network, instead of as models for the input data as done in generative modeling. Our contributions are to fill in this unexplored gap between UDA and generative modeling.

Specifically, in this work we investigate the potential use of generative modeling to solve UDA and propose a novel UDA framework based on probabilistic modeling and variational inference following the works [24, 35, 36]. Our contributions are summarized as follows:

1. Motivations towards a generative modeling view of UDA.
2. A novel framework for solving the UDA problem by generative modeling.
3. Empirical results demonstrating success and failure cases of our approach and their analysis.
4. Adoption of the proposed framework to solve the closely-related domain translation problem.

This thesis is organized as follows: Chapter 2 contains an overview of the definition of supervised learning, followed by a discussion on the problem of domain shift; we formally define the UDA problem in Chapter 3 followed by our alternative UDA problem formulation; Chapter 4 covers the details of our approach to solve the UDA problem, along with its implementation; lastly, Chapter 5 presents the experimental results of our approach compared to other mainstream UDA approaches, together with our discussion on its success and failure cases.

Chapter 2

Background

We review the notion of supervised learning called *empirical risk minimization* (ERM) in this chapter along with the problem of domain shift. The ERM paradigm underlies most of machine learning today. We provide a brief definition of ERM in this chapter, examples of ERM in practice, and some of its statistical guarantees. Later, we introduce the *domain shift* problem that arises in applying ERM in an out-of-distribution fashion, which is a more generalized problem than that of the UDA problem, but nevertheless provides insights into why ERM fails in such cases. The purpose of this chapter is to provide the foundation leading to the UDA problem introduced in Chapter 3.

2.1 Definitions and Notations

We dedicate this section to the definitions and notations used throughout this thesis.

2.1.1 Probability Theory

We assume a common probability space $(\Omega, \mathcal{A}, \mathbb{P})$ for all random variables being discussed in this section. Here Ω is a set called the sample space, $\mathcal{A} \subseteq 2^\Omega$ is a σ -algebra, and $\mathbb{P} : \mathcal{A} \rightarrow [0, 1]$ is a probability measure.

Random Variable

Let $X : \Omega \rightarrow \mathbb{R}^n$ be a random variable. Given a set $A \in \mathcal{B}(\mathbb{R}^n)$, where $\mathcal{B}(\mathbb{R}^n)$ denotes the Borel sigma algebra on \mathbb{R}^n , we will usually write

$$\mathbb{P}(X \in A) := \mathbb{P}(\{\omega : X(\omega) \in A\}).$$

Given a random variable X , its **cumulative distribution function (cdf)**, $F_X : \mathbb{R}^n \rightarrow [0, 1]$, is defined as

$$F_X(x) := \mathbb{P}(X \leq x),$$

where the inequality $X(\omega) \leq x$ is component-wise.

If X is a continuous random variable¹ and F_X is absolutely continuous in x , then X has a well-defined **probability density function (pdf)**, $f_X : \mathbb{R}^n \rightarrow \mathbb{R}_{\geq 0}$, defined by the property

$$F_X(x) = \int_{u \leq x} f_X(u) du,$$

where $\int_{u \leq x} f_X(u) du$ is a shorthand for the multidimensional integral

$$\int_{u_1 \leq x_1} \dots \int_{u_n \leq x_n} f_X(u_1, \dots, u_n) du_n \dots du_1.$$

If X is a discrete random variable², then X has a well-defined **probability mass function (pmf)**, $f_X : \mathbb{R}^n \rightarrow [0, 1]$, defined by the property

$$F_X(x) = \sum_{u \leq x} f_X(u),$$

where $\sum_{u \leq x} f_X(u)$ is a shorthand for the sum

$$\sum_{u_1 \leq x_1} \dots \sum_{u_n \leq x_n} f_X(u_1, \dots, u_n).$$

We refer to f_X as the **probability function** of X to cover for both cases. In this thesis all continuous random variables are assumed to be absolutely continuous and therefore have well-defined pdfs.

¹Its codomain is an uncountable set.

²Its codomain is a countable set.

In addition, we will often discuss different probability functions on the same random variable $X : \Omega \rightarrow \mathbb{R}^n$, e.g. f_X and g_X . In such cases, f_X and g_X each induces a different probability measure on the measurable space $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$, and we say X follows f_X to refer to the probability measure induced by f_X , similarly for g_X .

Joint Distribution Functions

Consider the random variable X given as before, and a new random variable $Y : \Omega \rightarrow \mathbb{R}^k$, then (X, Y) is another random variable $(X, Y) : \Omega \rightarrow \mathbb{R}^n \times \mathbb{R}^k$. The **joint cdf** of (X, Y) , $F_{X,Y} : \mathbb{R}^n \times \mathbb{R}^k \rightarrow [0, 1]$, is defined as

$$F_{X,Y}(x, y) := \mathbb{P}(X \leq x, Y \leq y).$$

If Y is a continuous random variable and $F_{X,Y}$ is absolutely continuous in x and y , then (X, Y) has a well-defined **joint pdf** $f_{X,Y} : \mathbb{R}^n \times \mathbb{R}^k \rightarrow \mathbb{R}_{\geq 0}$, defined by the property

$$F_{X,Y}(x, y) = \int_{u \leq x} \int_{v \leq y} f_{X,Y}(u, v) \, dv du,$$

Similarly, if Y is a discrete random variable, then (X, Y) has a well-defined joint pdf defined by the property

$$F_{X,Y}(x, y) = \int_{u \leq x} \sum_{v \leq y} f_{X,Y}(u, v) \, du,$$

The term joint pdf is used in the context of this thesis to refer to the probability function associated with the concatenation of different random variables.

Marginalization

The individual probability functions f_X and f_Y can always be obtained from $f_{X,Y}$ through marginalization:

$$f_X(x) = \int_{\mathbb{R}^k} f_{X,Y}(x, y) \, dy, \quad f_Y(y) = \int_{\mathbb{R}^n} f_{X,Y}(x, y) \, dx,$$

where the integration is replaced with summation if the random variable being marginalized is discrete.

Conditional Probability

For two random variables $X : \Omega \rightarrow \mathbb{R}^n$, $Y : \Omega \rightarrow \mathbb{R}^k$ and two events $A \in \mathcal{B}(\mathbb{R}^n)$, $B \in \mathcal{B}(\mathbb{R}^k)$, the **conditional probability** of $Y \in B$ given $X \in A$ is defined as

$$\mathbb{P}(Y \in B | X \in A) := \frac{\mathbb{P}(Y \in B \cap X \in A)}{\mathbb{P}(X \in A)}.$$

The **conditional probability function** of Y given X , $f_{Y|X} : \mathbb{R}^k \rightarrow \mathbb{R}_{\geq 0}$, is defined as

$$f_{Y|X}(y|x) := \frac{f_{X,Y}(x,y)}{f_X(x)}. \quad (2.1)$$

The conditional probability function $f_{Y|X}$ relates to the conditional probability of Y given X through the relationship

$$\mathbb{P}(Y \in B | X = x) = \int_{y \in B} f_{Y|X}(y|x) dy.$$

As usual, the integral above is replaced with sum if Y is discrete.

Bayes Theorem

Given marginal probability functions $f_X : \mathbb{R}^n \rightarrow \mathbb{R}_{\geq 0}$ and $f_Y : \mathbb{R}^k \rightarrow \mathbb{R}_{\geq 0}$, the conditional probability function $f_{Y|X} : \mathbb{R}^n \times \mathbb{R}^k \rightarrow \mathbb{R}_{\geq 0}$, Bayes Theorem can be used to obtain the conditional probability function $f_{X|Y} : \mathbb{R}^n \times \mathbb{R}^k \rightarrow \mathbb{R}_{\geq 0}$, which is the property that

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)f_X(x)}{f_Y(y)}.$$

Factorization

Any joint pdf $f_{X,Y} : \mathbb{R}^n \times \mathbb{R}^k \rightarrow \mathbb{R}_{\geq 0}$ over the random variables X and Y can be factorized into conditional and marginal probability functions via (2.1).

$$f_{X,Y}(x,y) = f_{X|Y}(x|y)f_Y(y),$$

or alternatively

$$f_{X,Y}(x,y) = f_{Y|X}(y|x)f_X(x).$$

More generally, given a set of random variables $\{X_1, \dots, X_N\}$, their joint pdf can be factorized into the form

$$f_{X_1, \dots, X_N}(x_1, \dots, x_N) = f_{X_1}(x_1) \prod_{i=2}^N f_{X_i|X_{i-1}, \dots, X_1}(x_i|x_{i-1}, \dots, x_1).$$

A Comment on Notation

To reduce notational clutter from the various probability functions that arise in this thesis, we will use the symbol p to denote different probability functions, where the arguments of p are used to indicate which random variables and probability functions are involved. Below are some examples:

1. $p(x) := f_X(x)$, $p(y) := f_Y(y)$
2. $p(y|x) := f_{Y|X}(y|x)$, $p(x|y) := f_{X|Y}(x|y)$, and
3. $p(x, y) := f_{X,Y}(x, y)$.

At times we also use the notation $p(x, y)$ to refer to the probability function $f_{X,Y}$ itself instead of its value at (x, y) .

Expectation

Let $X : \Omega \rightarrow \mathbb{R}^n$ be a random variable and $p(x)$ a probability function for X , the **expectation** of X with respect to $p(x)$ is defined as

$$\mathbb{E}_{X \sim p(x)}[X] := \int_{\mathbb{R}^n} xp(x) dx.$$

Let $Y : \Omega \rightarrow \mathbb{R}^k$ be another random variable, and $p(y|x)$ a conditional probability function of Y given X , the **conditional expectation** of Y given $X(\omega) = x$, with respect to $p(y|x)$, is defined as

$$\mathbb{E}_{Y \sim p(y|x)}[Y] := \int_{\mathbb{R}^k} yp(y|x) dy.$$

Given X and $p(x)$, a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^o$, the expectation of $f(X)$ with respect to $p(x)$ can be obtained through the property

$$\mathbb{E}_{X \sim p(x)}[f(X)] = \int_{\mathbb{R}^n} f(x)p(x) dx.$$

Sampling

Let $X : \Omega \rightarrow \mathbb{R}^k$ be a random variable and $p(x)$ be a probability function of X . A **sample** from $p(x)$, denoted $x_i \sim p(x)$ with i being a numerical label, is defined as

$$x_i := X(\omega_i), \text{ such that } \omega_i \in \Omega \text{ and } \mathbb{P}(X \leq u) = \int_{-\infty}^u p(x) dx.$$

2.1.2 Information Theory

Entropy

Given a random variable X distributed according to $p(x)$, its **entropy** is defined as

$$\mathcal{H}[X] = \mathbb{E}_{X \sim p(x)}[-\log p(X)].$$

Given two random variables X and Y with joint probability function $p(x, y)$, the **conditional entropy** of Y given X is defined as

$$\mathcal{H}[Y|X] = \mathbb{E}_{X, Y \sim p(x, y)}[-\log p(Y|X)].$$

KL-Divergence

Given a random variable X , two probability functions $p(x)$ and $q(x)$ for X , the **Kullback-Leibler (KL) divergence** between $p(x)$ and $q(x)$ is defined as

$$D_{\text{KL}}[p(x)||q(x)] := \mathbb{E}_{X \sim p(x)} \left[\log \frac{p(X)}{q(X)} \right].$$

The KL-divergence characterizes the difference between one probability function $q(x)$ from a reference probability function $p(x)$, and is zero if and only if $p(x) = q(x)$ for all $x \in \mathbb{R}^n$.

Let X and Y be random variables with joint pdf $p(x, y) = p(y|x)p(x)$. Let $q(y|x)$ be a conditional probability function of Y given X , the **conditional KL divergence** is defined as

$$D_{\text{KL}}[p(y|x)||q(y|x)] := \mathbb{E}_{X, Y \sim p(x, y)} \left[\log \frac{p(Y|X)}{q(Y|X)} \right].$$

Mutual Information

The **mutual information (MI)** between two random variables X and Y with joint pdf $p(x, y)$ is defined as

$$\mathcal{I}(X; Y) := D_{\text{KL}}[p(x, y) \| p(x)p(y)].$$

The mutual information can be rewritten in terms of entropy as

$$\mathcal{I}(X; Y) = \mathcal{H}[X] - \mathcal{H}[X|Y].$$

In addition, the mutual information enjoys symmetry, i.e.

$$\mathcal{I}(X; Y) = \mathcal{I}(Y; X).$$

The **conditional MI** of two random variables X and Y given another random variable Z , where $p(x, y, z)$ is the joint pdf of the three, is defined as

$$\mathcal{I}(X; Y|Z) := \mathbb{E}_{Z \sim p(z)} [D_{\text{KL}}[p(x, y|Z) \| p(x|Z)p(y|Z)]]$$

The conditional MI can be expressed in terms of conditional entropy:

$$\mathcal{I}(X; Y|Z) = \mathcal{H}[X|Z] - \mathcal{H}[X|Y, Z],$$

and it enjoys symmetry.

Jensen's Inequality

For a random variable $X : \Omega \rightarrow \mathbb{R}^n$ and a convex function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the **Jensen's inequality** states that

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)].$$

For example, the Jensen's inequality can be applied to lower-bound the KL-divergence:

$$\begin{aligned} D_{\text{KL}}[p(x) \| q(x)] &= \mathbb{E}_{X \sim p(x)} \left[\log \frac{p(X)}{q(X)} \right] = \mathbb{E}_{X \sim p(x)} \left[-\log \frac{q(X)}{p(X)} \right] \\ &\geq -\log \mathbb{E}_{X \sim p(x)} \left[\frac{q(X)}{p(X)} \right] \quad \triangleright \text{since } -\log \text{ is convex.} \\ &= 0. \end{aligned}$$

2.2 Supervised Learning

In supervised learning, a dataset comprising of the inputs and their corresponding labels is given, and the goal is to obtain a model of the underlying relationship between them such that we can use the model to make predictions on future unseen (unlabeled) data. We use classification as the prime example of a supervised learning task throughout this thesis, in which case the set of labels is finite. Formally, let $(\Omega, \mathcal{A}, \mathbb{P})$ be the underlying probability space governing the experiment of data collection and data labeling.³ We model the input as being elements of a set $\mathbb{X} \subseteq \mathbb{R}^n$ and refer to \mathbb{X} as the input space; and the label as being elements of a set $\mathbb{Y} = \{1, \dots, K\}$, where K is the number of labels. To reason about the uncertainty in the data, we introduce the random variables $X : \Omega \rightarrow \mathbb{X}$ for the input and $Y : \Omega \rightarrow \mathbb{Y}$ for the label.

Given these definitions, let $p(x, y)$ be the joint pdf of (X, Y) induced from \mathbb{P} , i.e. it satisfies the relationship

$$\mathbb{P}(X \leq u, Y \leq v) = F_{X,Y}(u, v) = \sum_{y \leq v} \int_{x \leq u} p(x, y) dx.$$

We refer to the pdf induced from \mathbb{P} over some random variables the *true underlying pdf* of the random variables. In this case $p(x, y)$ is the true underlying pdf of (X, Y) .

Next, we review the notion of a *dataset*.

Definition 1 (Dataset). A **dataset** \mathcal{D} is a finite collection of (input, label) pairs from the experiment,

$$\mathcal{D} = \{(x_i, y_i) \in \mathbb{X} \times \mathbb{Y} : (x_i, y_i) \sim p(x, y)\}_{i=1}^N.$$

In practice, generating a dataset consists of collecting input data from a population, followed by manual labeling of the data by human experts. This process that generates our dataset is assumed to be characterized by some unknown underlying probability distribution $p(x, y)$.

Risk Minimization

Most of supervised learning reduces to the problem of risk minimization. In short, it is concerned with finding the best *hypothesis* within a *hypothesis space* that minimizes a *loss function* over the underlying probability space.

³We will use this same probability space throughout this thesis.

Definition 2 (Hypothesis & Hypothesis Space). A **hypothesis space** \mathcal{F} is a collection of functions $f : \mathbb{X} \rightarrow \mathbb{Y}$. Each element $f \in \mathcal{F}$ is called a **hypothesis**.

A **loss function** is a function $L : \mathbb{Y} \times \mathbb{Y} \rightarrow \mathbb{R}_{\geq 0}$ that measures the discrepancy between the label predicted by a hypothesis and the groundtruth label on a given input instance.

Definition 3 (Risk). Given a hypothesis $f \in \mathcal{F}$ and loss function L , the **risk** of f with respect to a probability function $p(x, y)$ is

$$\epsilon(f; p) = \mathbb{E}_{(X, Y) \sim p(x, y)} [L(f(X), Y)].$$

Risk minimization refers to minimizing the risk over the hypothesis space

$$\min \{ \epsilon(f; p) : f \in \mathcal{F} \}. \quad (2.2)$$

Once solved, $f^* = \arg \min_{f \in \mathcal{F}} \epsilon(f; p)$ is referred to as the *optimal hypothesis*.

Application of Risk Minimization

To provide more insights for risk minimization, we present some common examples of its application in classification tasks.

Example 2.2.1. Minimize Misclassification

Choose $\mathcal{F} = \{f : \mathbb{X} \rightarrow \{1, \dots, K\}\}$, the set of all functions mapping from the input space to the label space. A natural loss function for classification in this case is the *0-1 loss* defined as

$$L(\hat{y}, y) = \begin{cases} 1, & \text{if } \hat{y} \neq y, \\ 0, & \text{otherwise.} \end{cases}$$

In this case the risk of $f \in \mathcal{F}$ equals

$$\begin{aligned} \epsilon(f; p) &= \mathbb{E}_{(X, Y) \sim p(x, y)} [L(f(X), Y)] \\ &= \int_{\mathbb{X}} \sum_{y \in \mathbb{Y}} L(f(x), y) p(x, y) dx \\ &= \int_{\mathbb{X}} p(x) \sum_{y \in \mathbb{Y}} L(f(x), y) p(y|x) dx \quad \triangleright \text{factorization of } p(x, y). \end{aligned}$$

In the last equality, the inner sum represents $\mathbb{P}(Y \neq f(X)|X = x)$. The optimal f is the one which minimizes this probability across all $x \in \mathbb{X}$. Thus f returns the label with the highest probability given x , i.e.

$$f^*(x) = \arg \max_{y \in \mathbb{Y}} p(y|x). \quad (2.3)$$

As a result, if $p(y|x)$ is known, then the optimal hypothesis under this setting is trivial to obtain. This observation motivates another paradigm of risk minimization that focuses on direct estimation of $p(y|x)$, which we discuss next. ▲

Example 2.2.2. Maximum Likelihood Estimation

In maximum likelihood estimation, instead of a hypothesis space of functions from \mathbb{X} to \mathbb{Y} , our hypothesis space is taken as

$$\mathcal{F} = \{f : \mathbb{X} \rightarrow \Delta^K\},$$

where $\Delta^K = \{u \in \mathbb{R}^K : \sum_{i=1}^K u_i = 1, u_i \geq 0\}$ and u_i is the i th component of u . Each element $u \in \Delta^K$ can be viewed as a vector of label probabilities, where the i th component is the probability of selecting label i given x . So \mathcal{F} is now the set of all possible conditional probability mass functions mapping an input x to probabilities over the labels. Since now f is a distribution, a natural loss function is the negative-log-likelihood from the statistical literature, defined by

$$L(\hat{y}, y) = -\log(\hat{y}_y),$$

where

$$\hat{y}_y := y\text{-th element of } \hat{y}.$$

The risk in this case is given by

$$\begin{aligned} \epsilon(f; p) &= \mathbb{E}_{(X,Y) \sim p(x,y)} [-\log f(X)_Y] \\ &= \mathbb{E}_{(X,Y) \sim p(x,y)} \left[\frac{\log p(Y|X)}{\log f(X)_Y} \right] - \mathbb{E}_{(X,Y) \sim p(x,y)} [\log p(Y|X)]. \end{aligned} \quad (2.4)$$

In the last expression, the first term is the KL-divergence between $p(y|x)$ and $f(x)$, where we emphasize that $f(x)$ is a conditional probability distribution of y given x . The second

term is the conditional entropy of Y given X , denoted by $\mathcal{H}[Y|X]$. Hence (2.4) can be written

$$\epsilon(f; p) = D_{\text{KL}}[p(y|x)||f(x)] + \mathcal{H}[Y|X].$$

Since $\mathcal{H}[Y|X]$ is independent of f , the risk is minimized when $D_{\text{KL}}[p(y|x)||f(x)_y] = 0$, which happens if and only if $f(x)_y = p(y|x)$. Therefore the solution to this risk minimization problem is

$$f^*(x) = \begin{bmatrix} p(y = 1|x) \\ p(y = 2|x) \\ \vdots \\ p(y = K|x) \end{bmatrix}.$$

With f^* in hand, the classification rule that minimizes misclassification is trivially given by

$$\arg \max_{y \in \mathbb{Y}} f^*(x)_y, \tag{2.5}$$

similar to that given in (2.3).

▲

Empirical Risk Minimization

In general, (2.2) is not directly solvable due to two reasons:

1. The joint pdf $p(x, y)$ of X and Y is unknown. Instead, we only have samples in the form of a dataset \mathcal{D} .
2. The hypothesis space $\mathcal{F} = \{f : \mathbb{X} \rightarrow \mathbb{Y}\}$ is infinite dimensional.

To address issue 1, the method of empirical risk minimization (ERM) is used instead. In ERM, the expectation in Definition 3 is approximated by the empirical average over \mathcal{D} . This yields the *empirical risk* of a hypothesis f , defined as follows.

Definition 4 (Empirical Risk). *Given a dataset \mathcal{D} , a loss function L , and a hypothesis $f \in \mathcal{F}$, the **empirical risk** of f is*

$$\hat{\epsilon}(f; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} L(f(x_i), y_i).$$

The ERM problem is then the following optimization problem

$$\min \{\hat{\epsilon}(f; \mathcal{D}) : f \in \mathcal{F}\}.$$

To address issue 2, the hypothesis space is restricted to a set of parameterized functions $f_\theta : \mathbb{X} \rightarrow \mathbb{Y}$ with parameter vector $\theta \in \Theta$ and parameter space $\Theta \subseteq \mathbb{R}^p$. In this case, the hypothesis space becomes $\mathcal{F}_\Theta = \{f_\theta : \theta \in \Theta\}$. Some examples are

1. Space of affine functions:

$$\mathcal{F}_\Theta = \{f_\theta : \mathbb{X} \rightarrow \mathbb{Y}, f_\theta(x) = Wx + b\}, \quad W \in \mathbb{R}^{K \times n}, b \in \mathbb{R}^K,$$

in which $\theta = (W, b)$ and $\Theta = \mathbb{R}^{K \times n} \times \mathbb{R}^K$.

2. Space of a two-layer neural network:

$$\begin{aligned} \mathcal{F}_\Theta = \{f_\theta : \mathbb{X} \rightarrow \mathbb{Y}, f_\theta(x) = W_2\sigma(W_1x + b_1) + b_2\}, \\ W_1 \in \mathbb{R}^{h \times n}, W_2 \in \mathbb{R}^{K \times h}, b_1 \in \mathbb{R}^h, b_2 \in \mathbb{R}^K, \end{aligned}$$

in which $\theta = (W_1, b_1, W_2, b_2)$ and $\Theta = \mathbb{R}^{h \times n} \times \mathbb{R}^{K \times h} \times \mathbb{R}^h \times \mathbb{R}^K$. Here $\sigma : \mathbb{R}^h \rightarrow \mathbb{R}^h$ is an *activation function*, a nonlinear function applied in a component-wise manner, i.e. $\sigma(z)_i = \sigma(z_i)$.

With this restriction in place, ERM turns into a tractable optimization over a finite-dimensional parameter space.

$$\min \{\hat{\epsilon}(f_\theta; \mathcal{D}) : \theta \in \Theta\}. \tag{2.6}$$

And this is the optimization problem that underlies the vast majority of modern applications of supervised learning involving neural networks.

Under the additional assumption that the loss function L is bounded from above, the optimality gap between solutions to ERM in (2.6) and solutions to risk minimization in (2.2) can be probabilistically bounded [37]. To state the probabilistic bound, first note that the dataset \mathcal{D} in Definition 1 can be seen as an i.i.d random dataset with the following definition.

Definition 5 (Random Dataset). *A **random dataset** \mathcal{D}_{RV} is a finite collection of pairs of random variables*

$$\mathcal{D}_{RV} = \{(X_i, Y_i) : X_i : \Omega \rightarrow \mathbb{X}, Y_i : \Omega \rightarrow \mathbb{Y}\}_{i=1}^N,$$

in which each pair (X_i, Y_i) is an input and a label random variable.

A random dataset \mathcal{D}_{RV} is independent and identically distributed (**i.i.d.**) if, for all i , (X_i, Y_i) are independent and identically distributed. Under the assumption of a bounded loss function, we can take, without further loss of generality, $L : \mathbb{Y} \times \mathbb{Y} \rightarrow [0, 1]$. Let $L_i := L(X_i, Y_i)$ be the random variable representing the loss of the i -th item in the random dataset. Then $\text{Var}[L_i]$ is finite and the same for all i . By the weak law of large numbers it follows that

$$\begin{aligned} & \mathbb{P}(|\hat{\epsilon}(f_\theta; \mathcal{D}_{\text{RV}}) - \epsilon(f_\theta; p)| \geq \delta) \\ &= \mathbb{P}\left(\left|\frac{1}{N} \sum_{i=1}^N L(f_\theta(X_i), Y_i) - \mathbb{E}[L(f_\theta(X_i), Y_i)]\right| \geq \delta\right) \\ &\leq \frac{\text{Var}[L_i]}{N\delta^2} \quad \triangleright \text{Chebyshev's inequality.} \end{aligned}$$

This result implies that the empirical risk converges in probability to the true risk as the size of our dataset grows. As a result, for sufficiently large dataset, we can expect the hypothesis that achieves a low empirical risk to achieve low true risk with high probability.

2.3 Domain Shift

From our previous discussion on supervised learning, the empirical risk provides a good estimate of the true risk given a dataset of independent and identical distributed (i.i.d.) samples from the underlying distribution. Once a hypothesis that achieves low empirical risk is identified, we can expect it to perform well on all data samples $(x_i, y_i) \sim p(x, y)$, including those not seen during training.⁴

Much of the successes of supervised learning today relies on the existence of such a dataset that reflects the distribution of the data encountered by the model⁵ in deployment. However, in practice, it is often the case that the available dataset at hand is limited and unrepresentative of the distribution of data that will be encountered during deployment. This is known as the problem of *domain shift*.

For an example, consider the task of recognizing handwritten digits in images, where we have available a previously collected public dataset containing images of digits along with their labels. The dataset is relatively old, and the images are of low resolution due to the limitations of cameras at that time. In addition, people before used a different writing

⁴“Training” refers to carrying out the optimization in (2.6).

⁵Synonym for hypothesis.

style for the digits, so the digits today appear somewhat differently. A model obtained through supervised learning on the old dataset when applied to recognize modern images of digits is then said to experience domain shift, as the distribution of the data the model was trained on differs from the distribution it is applied to. This example is depicted in Figure 2.1, where for demonstration, we use the USPS dataset [38] as the training dataset and the MNIST dataset [39] as the deployment dataset.

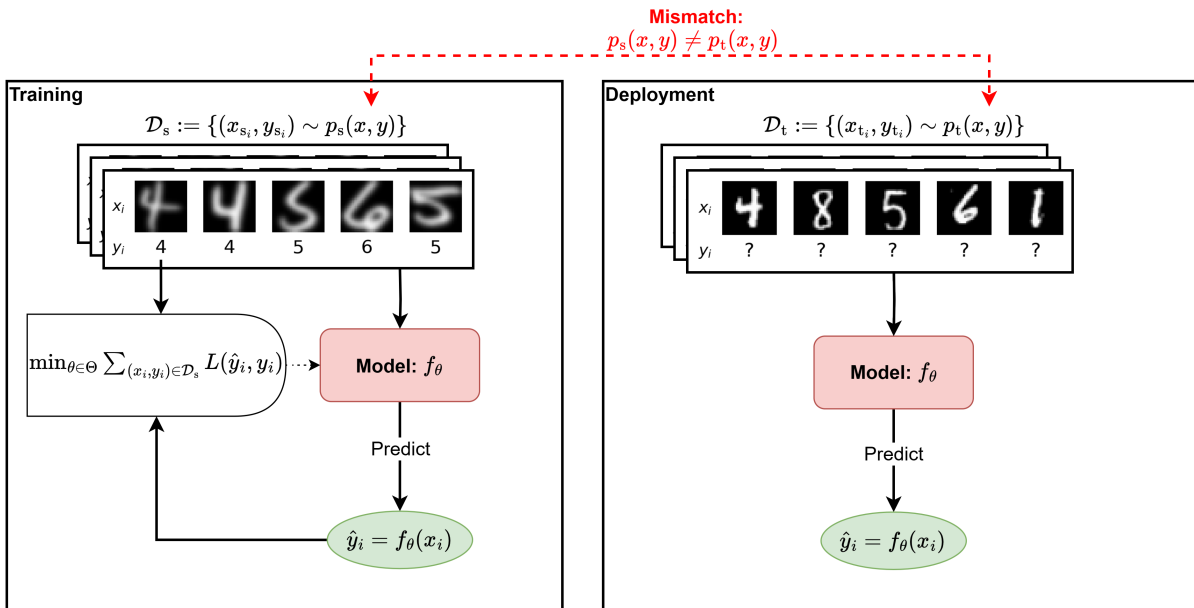


Figure 2.1: An illustration of domain shift in the task of recognizing handwritten digits. The model is trained on a dataset of low-resolution images of digit with a specific writing style (USPS dataset [38]), and is later applied to recognize images of digits with an obvious difference in visual appearance (MNIST dataset [39]).

Definition 6 (Domain). Let $\mathcal{P} = \{p : \mathbb{X} \times \mathbb{Y} \rightarrow [0, \infty], \int_{\mathbb{X}} \sum_y p(x, y) dx = 1\}$ be the set of all joint pdfs over the input-label space $\mathbb{X} \times \mathbb{Y}$. A **domain** is an element of \mathcal{P} .

We use $p_d(x, y)$ to refer a domain, where d is a label that identifies the domain.

Define the *source domain* $p_s(x, y)$ to be the distribution that generated the training dataset, and the *target domain* $p_t(x, y)$ to be the distribution of the data we want to deploy to. Domain shift refers to the fact that

$$p_s(x, y) \neq p_t(x, y).$$

In this thesis, we model the domain itself as a random variable and define $D : \Omega \rightarrow \{0, 1\}$ to be a random variable associated to the domain label, where $D(\omega) = 0$ corresponds to the source domain and $D(\omega) = 1$ the target domain. Similar to previous definitions of joint pdfs, we denote the joint pdf over (X, Y, D) as $p(x, y, d)$, with $p(x, y, d)$ satisfying the following relationship with respect to \mathbb{P}

$$\mathbb{P}(X \leq u, Y \leq v, D \leq w) = \sum_{d \leq w} \sum_{y \leq v} \int_{x \leq u} p(x, y, d) dx.$$

It is easy to see that direct application of the paradigm of supervised learning from Section 2.2 under domain shift fails to provide guarantees on the risk of the model in the target domain.

Domain shifts commonly occur in applications of supervised learning. They arise as a result of different data collection processes, biases in data collection, or changes in the data distribution over time. In general, it is not realistic to perform extensive data collection and labeling that takes into account all possible variations in the data. As a result, the dataset available for training usually has limited expressiveness in representing the data distribution at deployment.

Example 2.3.1. Domain Shift in Practice

To see the effect of domain shift in practice, we perform an experiment for the aforementioned digits example shown in Figure 2.1, where we train a convolutional neural network with a similar architecture to [40] using the maximum-likelihood paradigm introduced in Example 2.2.2 on the source domain alone. Figure 2.2 shows the empirical risk and classification accuracy of the model on unseen source and target samples as training progresses.



From the results of this toy experiment, we see that the empirical risk of the model on unseen source samples continues to decrease as we optimize the model parameters through maximum-likelihood with respect to the source training samples. However, since the optimization is carried out on the source domain without regards to the distribution of the target domain, the model ultimately overfits to the source domain with a high empirical risk on the unseen target samples. As a result, the model in the end achieves excellent accuracy on the source domain, but the accuracy on the target domain remains poor.

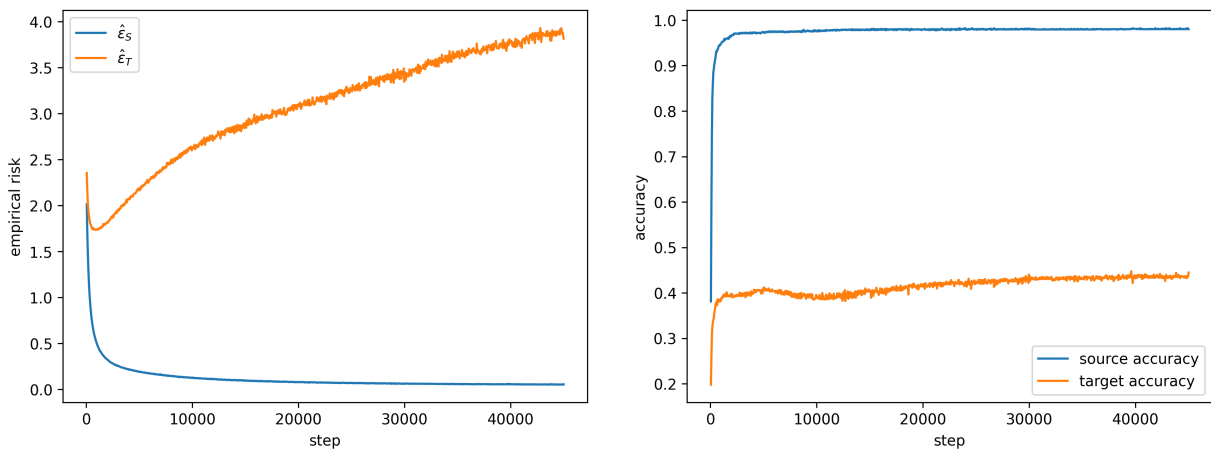


Figure 2.2: The training curves on the domain shift example of recognizing handwritten digits. Left: Empirical risk of the model on the source and target domains as training progresses. The empirical risk on the source domain monotonically decreases, while on the target domain it increases after a certain step. Right: Classification accuracy of the model on the source and target domains as training progresses. There is a persistent gap between the source and target accuracy as training progresses.

Chapter 3

Problem Definition

In this chapter, we give the standard definition of the UDA problem. In general, UDA is concerned with addressing domain shifts. We start with a brief mention of the assumption of no concept shift which underlies most UDA methods, followed by the formal objective of UDA in a typical setting. We then motivate and provide an alternative objective to UDA from a generative modeling perspective.

3.1 Unsupervised Domain Adaptation

Unsupervised domain adaptation (UDA) aims to overcome the performance gap of the model between the source and target domain in the presence of domain shift, provided that some unlabeled samples from the target domain are available. However, because the target samples¹ are unlabeled, we cannot in general expect to find a good model in the target domain if the labeling distribution $p(y|x, d)$ differs between the domains. This leads to an assumption termed *no concept shift* on the allowable types of domain shift which underlies most UDA methods. Recall, as in Chapter 2, the joint pdf of (X, Y, D) is $p(x, y, d)$ and $d = 0$ corresponds to the source domain while $d = 1$ corresponds to the target domain.

Assumption 1 (No Concept Shift). *The joint pdf of (X, Y, D) has **no concept shift***

$$p(y|x, d = 1) = p(y|x, d = 0).$$

¹Samples from the target domain.

In other words, UDA assumes that the domain shift is limited to the difference in the marginal distribution of the input between domains,

$$p(x|d = 1) \neq p(x|d = 0),$$

where the difference between the marginal distributions is termed the *covariate shift*.

In practice, given two datasets corresponding to two domains, no concept shift can be assumed if the examples from one dataset when transferred into the other domain share their original labels. Our digits example shown in Figure 2.1 satisfies this property: If the images of “9” from the source dataset is treated as a sample in the target domain, its true label remains to be that of “9”, and vice-versa for “9”-images in the target dataset. For an example in which no concept shift fails, consider the same digits example, but now the target dataset consists of inverted images (such that “6” appears as “9” and vice-versa). Clearly the previous observation fails: an image of “9” has label “9” in the source domain but label “6” in the target domain.

With the motivations and definitions in place, we give a formal definition of the problem of UDA in this section, tailored specifically to the case of neural networks and classification tasks.

Given two domains, where one is the source domain $p_s(x, y)$ and the other the target domain $p_t(x, y)$, an i.i.d. labeled dataset from the source domain and an i.i.d. unlabeled dataset from the target domain,

$$\mathcal{D}_s = \{(x_{s_i}, y_{s_i}) \in \mathbb{X} \times \mathbb{Y} \sim p_s(x, y)\}_{i=1}^{N_s}; \quad \mathcal{D}_t = \{(x_{t_i}, y_{t_i}) \in \mathbb{X} \times \mathbb{Y} \sim p_t(x, y)\}_{i=1}^{N_t},$$

where y_{t_i} is not accessible since \mathcal{D}_T is unlabeled. Let \mathcal{F} denote a hypothesis space. The objective of UDA is to find the risk minimization hypothesis with respect to the target domain, namely to minimize

$$\epsilon(f; p_t) = \mathbb{E}_{(X, Y) \sim p(x, y|d=1)} [L(f(X), Y)].$$

We primarily focus on UDA applicable to neural networks in this work, and use the task of classification as the running example. For this, we follow the maximum-likelihood paradigm described in Example 2.2.2. We take the hypothesis space $\mathcal{F}_\Theta = \{f_\theta : \mathbb{X} \rightarrow \Delta^K; \theta \in \Theta\}$ to be the set of conditional distribution of Y given X parameterized by a neural network, where $\Theta \subseteq \mathbb{R}^p$ is the parameter space. In addition, the loss function is chosen to be the usual negative-log-likelihood $L(f_\theta(x), y) = -\log(f_\theta(x)_y)$. In this case, the objective becomes the minimization of

$$\epsilon(f_\theta; p_t) = \mathbb{E}_{(X, Y) \sim p(x, y|d=1)} [-\log f_\theta(x)_y], \tag{3.1}$$

We refer to f_θ as the *classifier*, and denote $f_\theta(x)_y$ by $p_\theta(y|x)$ when a probabilistic interpretation of f_θ is sensible, to signify the fact that f_θ is an approximation of $p(y|x)$.

3.2 Alternate Problem

Though the common objective of UDA is (3.1), this optimization problem is intractable without access to samples of (input, label) pairs on the target domain, which is precisely the situation with UDA. Although re-weighting approaches provide a principled way to estimate the objective function of interest under the assumption of no concept shift, they are commonly incompatible with high-dimensional data that are common in modern applications, and dimensionality reduction techniques are required as a pre-processing step. On the other hand, domain alignment approaches have mostly relied on the theoretical bounds derived in [27, 28], but the bound contains an intractable optimal combined risk term that cannot be estimated without labeled data on the target domain, and it is instead assumed to be low in practice, which may not be realistic.

Due to the lack of a uniform solution to the UDA problem, we make an attempt to instead perform generative modeling of the underlying joint distribution $p(x, y, d)$ through a latent variable model $p_\theta(x, y, d)$ such that the joint relationship between X, Y is captured through

$$p_\theta(x, y) = p_\theta(x, y, d = 0) + p_\theta(x, y, d = 1)$$

We motivated this by the following observations.

1. Existence of Domain-Invariant and Domain-Specific Factors Many UDA problems exhibit the trait that the label-discriminative features in the input data are shared across the domains, whereas the other features that are not essential for label predictions vary between the domains. We refer to the former set of features the *domain-invariant factors* and the latter the *domain-specific factors*, similar to that in [24]. This prior knowledge can be directly embedded into the factorization of a latent variable model in the forms of independence assumptions.

2. Direct Regularization from Unlabeled Data A generative model naturally benefits from an abundance of unlabeled data [41]. As opposed to the discriminative modeling used in (3.1), which solely exploits labeled data from the source domain, generative model can be optimized with both labeled and unlabeled data through maximum-likelihood on both the joint density $p_\theta(x, y)$ and the marginal density $p_\theta(x)$. Through sharing components in the factorizations of $p_\theta(x, y)$ across the domains, maximizing the likelihood of $p_\theta(x)$ effectively acts as a regularization on the shared components, encouraging them to closely model the distribution of the data in the target domain.

3. Connection to Representation Learning Many successful approaches based on representation learning performs reconstruction to extract a useful representation of the input data for label prediction across domains [24, 23, 21]. This can be seen as simultaneously learning a generative model of the input data $p_\theta(x)$ that approximates the input distribution $p(x)$ along with a discriminative model $p_\theta(y|x)$ that approximates the conditional distribution $p(y|x)$, where the weights of the two models are shared to various degree.

Our proposal is similar in terms of joint modeling of the input and label distributions. But instead of combining them through weight-sharing at the level of the architecture of the neural networks involved, they are embedded into $p_\theta(x, y)$ as factors, where sharing of components happens at the factorization level. This offers better interpretation in terms of the statistical relationship among the different representations and more granular control in placing prior beliefs on the structure of the representations.

3.2.1 Our Revised Objective

Motivated by the above considerations, we define our model to be a parameterized joint pdf $p_\theta : \mathbb{X} \times \mathbb{Y} \times \{0, 1\} \rightarrow \mathbb{R}_{\geq 0}$, denoted by $p_\theta(x, y, d)$, that satisfies the normalization constraints:

$$\int_{\mathbb{X}} \sum_y \sum_{d \in \{0,1\}} p_\theta(x, y, d) dx = 1.$$

And we propose a revised objective for UDA given by

$$\min_{\theta \in \Theta} -\mathbb{E}_{X, Y \sim p(x, y|d=0)} [\log p_\theta(X, Y|d=0)] - \mathbb{E}_{X \sim p(x|d=1)} [\log p_\theta(X|d=1)], \quad (3.2)$$

where $p_\theta(x, y|d=0)$ and $p_\theta(x|d=1)$ is the corresponding conditional probability function derived from $p_\theta(x, y, d)$:

$$p_\theta(x, y|d=0) = \frac{p_\theta(x, y, d=0)}{\int_{\mathbb{X}} \sum_y p_\theta(x, y, d=0) dx}, \quad p_\theta(x|d=1) = \frac{\sum_y p_\theta(x, y, d=1)}{\int_{\mathbb{X}} \sum_y p_\theta(x, y, d=1) dx}.$$

We refer to (3.2) as our revised objective for UDA.

The first term in (3.2) can be seen as minimizing the KL-divergence between $p(x, y|d=$

0) and $p_\theta(x, y|d = 0)$:

$$\begin{aligned}
& - \mathbb{E}_{X, Y \sim p(x, y|d=0)} [\log p_\theta(X, Y|d = 0)] \\
& = \mathbb{E}_{X, Y \sim p(x, y|d=0)} \left[\log \frac{p(X, Y|d = 0)}{p_\theta(X, Y|d = 0)} \right] - \mathbb{E}_{X, Y \sim p(x, y|d=0)} [\log p(X, Y|d = 0)] \\
& = D_{\text{KL}}[p(x, y|d = 0) \| p_\theta(x, y|d = 0)] + \mathcal{H}[X, Y|D = 0], \tag{3.3}
\end{aligned}$$

and similarly the second term is minimizing the KL-divergence between $p(x|d = 1)$ with $p_\theta(x|d = 1) = \sum_y p_\theta(x, y|d = 1)$:

$$- \mathbb{E}_{X, Y \sim p(x|d=1)} [\log p_\theta(X|d = 1)] = D_{\text{KL}}[p(x|d = 1) \| p_\theta(x|d = 1)] + \mathcal{H}[X|D = 1]. \tag{3.4}$$

Once (3.2) is solved, the desired classifier $f : \mathbb{X} \rightarrow \mathbb{Y}$ can be obtained in the usual way as in Example 2.2.1, i.e.

$$\begin{aligned}
\hat{y} & = \arg \max_{y \in \mathbb{Y}} p_\theta(y|x) \\
& = \arg \max_{y \in \mathbb{Y}} \frac{p_\theta(x, y)}{p_\theta(x)} \\
& = \arg \max_{y \in \mathbb{Y}} \frac{\sum_{d \in \{0,1\}} p_\theta(x, y, d)}{\sum_{y'} \sum_{d \in \{0,1\}} p_\theta(x, y', d)}.
\end{aligned}$$

Using (3.3) and (3.4), along with the fact that the entropy term in both is constant, we can express the objective in (3.2) as

$$\begin{aligned}
& D_{\text{KL}}[p(x, y|d = 0) \| p_\theta(x, y|d = 0)] + D_{\text{KL}}[p(x|d = 1) \| p_\theta(x|d = 1)] \\
& = D_{\text{KL}}[p(y|x, d = 0) \| p_\theta(y|x, d = 0)] \\
& \quad + D_{\text{KL}}[p(x|d = 0) \| p_\theta(x|d = 0)] + D_{\text{KL}}[p(x|d = 1) \| p_\theta(x|d = 1)].
\end{aligned}$$

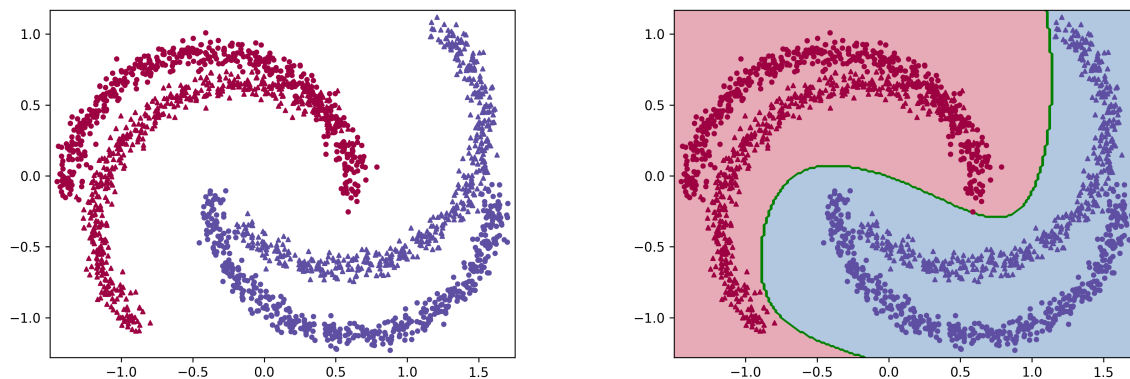
The first term corresponds to the typical ML objective for training $p_\theta(y|x)$ on the source domain as introduced in Example 2.2.2, except now $p_\theta(y|x)$ is part of $p_\theta(x, y, d)$. The second and third terms encourage $p_\theta(x)$ to resemble samples from both the source and target domain. We hypothesize that the latter two terms form a strong regularizer that prevents overfitting the source domain from optimizing solely the first term, given that $p_\theta(x, y, d)$ has some shared factors in both domains. We discuss our specific choice on the shared factors of $p_\theta(x, y, d)$ in Chapter 4.

3.2.2 Generative vs. Discriminative

We end this chapter with a toy example on how generative modeling can help solve the UDA problem. Consider the toy binary-classification example depicted in Figure 3.1a (generated from the Moon dataset in scikit-learn [42]), where the domain shift is a rotation to the two-dimensional feature space.

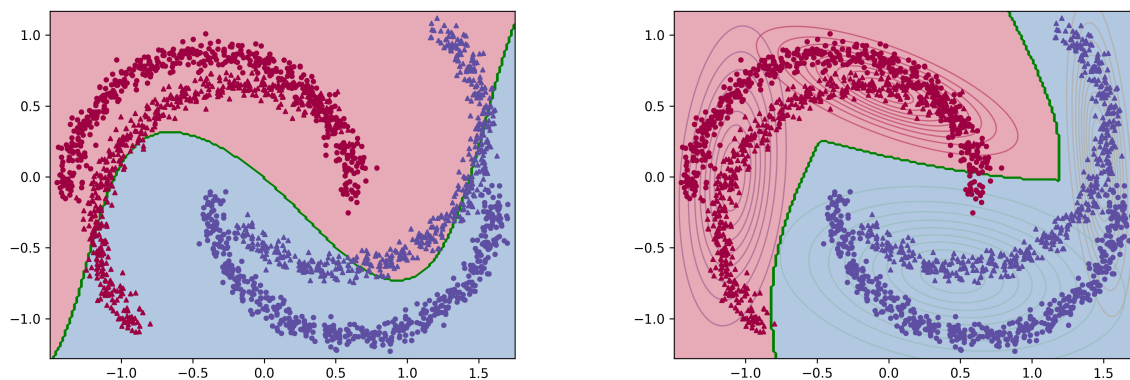
For illustration, a logistic regression model is fitted to only the labeled source dataset, which resembles a discriminative approach to this task; to contrast, a Gaussian mixture model (GMM) is also fitted, which resembles a generative approach. The GMM is first optimized to fit the unlabeled source and target datasets, after which each component in the GMM is assigned a label based on majority voting using the labeled source datapoints.

Figure 3.1c and Figure 3.1d shows the resulting decision boundary of the two approaches respectively. Comparing the two, we see that the decision boundary derived from the generative approach naturally accounts for the input distributions in both domains, and benefits from the abundance of unlabeled target datapoints. Lastly, Figure 3.1b is the decision boundary of a logistic regression model fitted to both labeled source and target datasets, and it is comparable to that resulting from the generative approach.



(a) The generated Moon dataset from scikit-learn. Different colors indicate different labels. Source datapoints are denoted by ● and target datapoints ▲.

(b) Discriminative fit: The decision boundary of logistic regression fitted with both the source and target datapoints.



(c) Discriminative approach: The decision boundary of logistic regression with only the labeled source datapoints.

(d) Generative approach: The decision boundary of a classifier based on Gaussian mixture models fitted with labeled source and unlabeled target datapoints.

Figure 3.1: Toy example comparing discriminative and generative modeling for the purpose of UDA.

Chapter 4

Approach

In this chapter, we begin with a discussion of the specific construct of our model $p_\theta(x, y, d)$ that approximates the joint pdf $p(x, y, d)$ of (X, Y, D) . We start with the motivation towards defining a new random variable that represents domain-invariant features in the input and infer the necessary independence relationships between it and (X, Y, D) , leading to a latent variable model.¹ We then elaborate on our approach to realize the minimization of our revised objective in (3.2) with respect to our choice of model.

4.1 Motivation

Many UDA problems share the trait that there exists some domain-invariant representation of the input that is consistently useful for classification without regards to the domains. For tabular input data, this could be the set of features that are directly associated with deciding the label; for our digits example in Figure 2.1 this could be the stroke of the digit as illustrated in Figure 4.1.

Formally, we model this as a map from the input space $\mathbb{X} \subseteq \mathbb{R}^n$ to some representation space $\mathbb{H} \subseteq \mathbb{R}^{n_h}$, $M_h : \mathbb{X} \rightarrow \mathbb{H}$, where $n_h \ll n$. Let $H = M_h \circ X$ be the random variable associated with the representation, and define the joint pdf $p(x, h, y, d)$ similar to $p(x, y, d)$,

$$\mathbb{P}(X \leq u, H \leq v, Y \leq w, D \leq a) = \int_{x \leq u} \int_{h \leq v} \sum_{y \leq w} \sum_{d \leq a} p(x, h, y, d) dh dx, \quad (4.1)$$

¹A factorization of $p_\theta(x, y, d)$ involving additional random variables.



Figure 4.1: Illustration of a domain invariant representation: Through a transformation that extracts the stroke of the digits, much of the difference in visual appearance across domains is eliminated.

We say that H is invariant across domains if

$$p(h|d = 0) = p(h|d = 1), \quad (4.2)$$

which also implies independence between H and D . If H is invariant across domains, then it can be seen as the encoding of some domain-invariant factors in \mathbb{H} . On the other hand, “usefulness for classification” can be characterized by the mutual information between H and Y , given by

$$\mathcal{I}(H; Y) := D_{\text{KL}}[p(h, y) \| p(h)p(y)] = \mathcal{H}[H] - \mathcal{H}[H|Y]. \quad (4.3)$$

From (4.3), it is evident that H becomes most useful when $\mathcal{H}[H|Y] \ll \mathcal{H}[H]$, which happens when for each $y \in \mathbb{Y}$, $p(h|y)$ has its mass concentrated over a much smaller subset of \mathbb{H} than that of $p(h) = \sum_y p(h|y)p(y)$. This informs the existence of clustering among the data in \mathbb{H} per the label y . This in combination to (4.2) implies that H corresponds to a data representation eliciting an invariant predictor (across domains) as defined in [43].

In addition, H should extract all of the label-related information from X , meaning that

$$\mathcal{I}(X; Y|H) = \mathcal{H}[Y|H] - \mathcal{H}[Y|X, H] = 0,$$

which implies conditional independence of X and Y given H , since the mutual information between random variables is zero if and only if they are independent [44]. This implies the equality

$$p(x|h, y, d) = p(x|h, d).$$

A desirable mapping M_h which generates the random variable H should have an associated induced pdf $p(x, h, y, d)$ with the aforementioned properties, namely:

1. H is independent of D .

2. $p(h) = \sum_y p(h|y)p(y)$ exhibits multi-modalness, where each mode corresponds to a cluster of the representations in \mathbb{H} for a given label y .
3. X and Y are independent given H .

Combining these properties, and further assuming that Y is independent of D , implies the existence of a factorization for $p(x, h, y, d)$ of the form:

$$p(x, h, y, d) = p(x|h, d)p(h|y)p(y)p(d).$$

If $p(h)$ associated with such M_h is known, then the revised objective (3.2) can be solved directly without regards to M_h by choosing our model of $p_\theta(x, y, d)$ to be the generative model $p_\theta(x, h, y, d)$ given by

$$p_\theta(x, h, y, d) = p_\theta(x|h, d)p(h|y)p(y)p(d), \quad p_\theta(x, y, d) = \int_{\mathbb{H}} p_\theta(x, y, h, d) dh.$$

With this particular choice of $p_\theta(x, y, d)$, the first term in (3.2) resolves to

$$\begin{aligned} & - \mathbb{E}_{X, Y \sim p(x, y|d=0)} [\log p_\theta(X, Y|d=0)] \\ & = - \mathbb{E}_{X, Y \sim p(x, y|d=0)} [\log \mathbb{E}_{H \sim p(h|Y)} [p_\theta(X|H, d=0)p(Y)]] , \end{aligned}$$

which fits $p_\theta(x|h, d)$ to reflect the per-label input distribution from the source: the input distribution given the cluster corresponding to y in the representation space \mathbb{H} . The second term in (3.2) results in

$$\begin{aligned} & - \mathbb{E}_{X \sim p(x|d=1)} [\log p_\theta(X|d=1)] \\ & = - \mathbb{E}_{X \sim p(x|d=1)} [\log \mathbb{E}_{H \sim p(h)} [p_\theta(X|H, d=1)]] , \end{aligned}$$

where

$$p(h) = \sum_y p(h|y)p(y).$$

This fits $p_\theta(x|h, d)$ to reflect the overall input distribution from the target when considering all clusters in \mathbb{H} among all labels.

Objective (3.2) hence becomes

$$\begin{aligned} & - \mathbb{E}_{X, Y \sim p(x, y|d=0)} [\log \mathbb{E}_{H \sim p(h|Y)} [p_\theta(X|H, d=0)p(Y)]] \\ & \quad - \mathbb{E}_{X \sim p(x|d=1)} [\log \mathbb{E}_{H \sim p(h)} [p_\theta(X|H, d=1)]] . \end{aligned} \tag{4.4}$$

In (4.4), $p(h|y)$ is shared among the two terms corresponding to the two domains, and $p_\theta(x|h, d)$ can be seen as means of modeling the distribution over the inverse image of M_h for each h , namely the set $M_h^{-1}(\{h\}) = \{x \in \mathbb{X} : M_h(x) = h\}$. The final classifier can be formed by Bayes' theorem via

$$p_\theta(y|x) = \frac{1}{K} \int_{\mathbb{H}} \sum_{d \in \{0,1\}} p_\theta(x|h, d) p(h|y) p(y) p(d) dh, \quad (4.5)$$

where $K = \int_{\mathbb{H}} \sum_y \sum_{d \in \{0,1\}} p_\theta(x|h, d) p(h|y) p(y) p(d) dh$ is a normalization constant, and $p(h|y) p(y)$ effectively acts as a prior for classification shared among the domains. In general, we can expect a good classifier for both domains if $p_\theta(x|h, d)$ closely approximates $p(x|h, d)$, such that $p_\theta(x, h, y, d) \approx p(x, h, y, d)$, and hence indirectly tackle the UDA problem.

4.2 Method

Despite our previous discussion, the above approach at its current state encompasses the following challenges:

1. Since H is domain-invariant, the true underlying $p(x|h)$ will contain modes that correspond to the domains for a given h . This multi-modal aspect is difficult to model using the factor $p_\theta(x|h, d)$ alone, especially when \mathbb{X} is of high dimension.
2. There is no access to M_h and hence $p(h)$. In fact, the UDA problem would be trivial to solve given access to M_h .
3. The two inner expectations over H in (4.4) are intractable to compute in general due to non-conjugacy between $p_\theta(x|h, d)$ and $p(h)$ or $p(h|y)$.
4. Similarly, the posterior $p_\theta(y|x)$ in (4.5) which is also an expectation is intractable to compute.

We hereby present the method to solve the above problems under an unified framework using variational inference applied to neural networks.

To overcome Problem 1, similar to our previous motive on H , we hypothesize the existence of a label-invariant representation defined by a map $M_z : \mathbb{X} \rightarrow \mathbb{Z}$ where $\mathbb{Z} \subseteq \mathbb{R}^{n_z}$, that is useful for classifying the domain D of the input X while being independent of

the label Y . Let $Z = M_z \circ X$ be its associated random variable and define the joint pdf $p(x, h, z, y, d)$ similar to (4.1).

Examples of such representation include the set of features that relate only to the domain of the input data in the case of tabular data; for our digits example in Figure 2.1, this could be the blurriness of the input image measured by the variance of its Laplacian, shown in Figure 4.2.

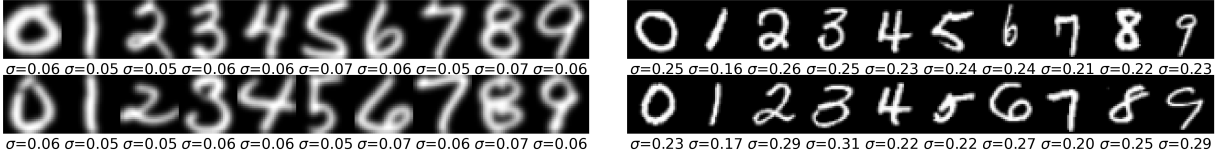


Figure 4.2: Illustration of a domain specific representation: The blurriness of the image as given by the variance of its Laplacian σ does not relate to its label, but identifies its domain. Note the different ranges of values for σ across domains, and that the value of σ relates weakly to an image’s label.

We refer to Z as encoding some domain-specific factors. Following similar reasoning on H , the independence of X and D given Z is implied. And the UDA problem can be solved by fitting a revised generative model given by

$$p_\theta(x, h, z, y, d) = p_\theta(x|h, z)p(h|y)p(z|d)p(y)p(d)$$

where

$$p_\theta(x, y, d) = \int_{\mathbb{H}} \int_{\mathbb{Z}} p_\theta(x, y, h, z, d) dz dh.$$

With this particular choice of $p_\theta(x, y, d)$, the first term of (3.2) resolves to

$$\begin{aligned} & -\mathbb{E}_{X, Y \sim p(x, y|d=0)} [\log p_\theta(X, Y|d=0)] \\ & = -\mathbb{E}_{X, Y \sim p(x, y|d=0)} [\log \mathbb{E}_{H, Z \sim p(h, z|Y, d=0)} [p_\theta(X|H, Z)]] , \text{ where } p(h, z|y, d) = p(h|y)p(z|d). \end{aligned}$$

And the second term of (3.2) resolves to

$$\begin{aligned} & -\mathbb{E}_{X \sim p(x|d=1)} [\log p_\theta(X|d=1)] \\ & = -\mathbb{E}_{X \sim p(x|d=1)} [\log \mathbb{E}_{H, Z \sim p(h, z|d=1)} [p_\theta(X|H, Z)]] , \text{ where } p(h, z|d) = \sum_y p(h|y)p(z|d)p(y). \end{aligned}$$

Objective (3.2) becomes

$$= -\mathbb{E}_{X,Y \sim p(x,y|d=0)} \left[\log \mathbb{E}_{H,Z \sim p(h,z|Y,d=0)} [p_\theta(X|H,Z)] \right] \\ - \mathbb{E}_{X \sim p(x|d=1)} \left[\log \mathbb{E}_{H,Z \sim p(h,z|d=1)} [p_\theta(X|H,Z)] \right],$$

The above combined enables the per-domain modes in $p(x|h)$ to be captured through variations introduced on Z , via

$$p_\theta(x|h) = \int_{\mathbb{Z}} \sum_d p_\theta(x|h,z)p(z|d)p(d) dz.$$

With regards to Problem 2, the unavailability of $p(h)$ and $p(z)$, we note that in general there exist many M_h and M_z that satisfy our proposed criteria. This produces a family of $p(h)$ and $p(z)$. Thus, we propose to treat H and Z as latent variables within our generative model subject to the respective independence constraints, with their distributions modeled by

$$p_\theta(h) = \sum_y p_\theta(h|y)p_\theta(y), \\ p_\theta(z) = \sum_d p_\theta(z|d)p_\theta(d).$$

This results in the generative model $p_\theta(x, h, z, y, d) = p_\theta(x|h, z)p_\theta(h|y)p_\theta(z|d)p_\theta(y)p_\theta(d)$ depicted in Figure 4.3, where we choose to model each component by

$$p_\theta(y) = \text{Cat}(y|\pi_y); \quad p_\theta(h|y) = \mathcal{N}(\mu_{h|y}[y], \Sigma_{h|y}[y]); \\ p_\theta(d) = \text{Cat}(d|\pi_d); \quad p_\theta(z|d) = \mathcal{N}(\mu_{z|d}[d], \Sigma_{z|d}[d]); \\ p_\theta(x|h, z) = \mathcal{N}(\hat{\mu}_{x|h,z}, \sigma \mathbf{I}),$$

with $\hat{\mu}_{x|h,z} \in \mathbb{X}$ being the output of a neural network:

$$g_\theta : \mathbb{H} \times \mathbb{Z} \rightarrow \mathbb{X}, \quad \hat{\mu}_{x|h,z} = g_\theta(h, z).$$

Notation-wise, $\text{Cat}(\cdot|\pi)$ denotes the categorical distribution with parameter π ; the $\mu_{h|y}[y] \in \mathbb{H}$ and $\Sigma_{h|y}[y] \in \mathbb{R}^{n_h \times n_h}$ are the mean and covariance to parameterize $p_\theta(h|y)$ where we use y as an index into the different sets of parameters; similarly for $\mu_{z|d}$ and $\Sigma_{z|d}$. The conditional distribution $p_\theta(h|y)$ is chosen to be normal to reflect the cluster of H in \mathbb{H} given Y per our discussion in Section 4.1; similarly for $p_\theta(z|d)$.

Note that compared to typical VAEs where the covariances of the latent variables are restricted to be diagonal, we do not make such restrictions in our case. This enables correlations among the factors in the latent space to be captured and a richer set of factors to be explored during the optimization of our generative model.

Incorporating the above model while treating H and Z as latent variables resolves objective (3.2) into

$$- \mathbb{E}_{X,Y \sim p(x,y|d=0)} [\log \mathbb{E}_{H,Z \sim p_\theta(h,z|Y,d=0)} [p_\theta(X|H,Z)]] - \mathbb{E}_{X \sim p(x|d=1)} [\log \mathbb{E}_{H,Z \sim p_\theta(h,z|d=1)} [p_\theta(X|H,Z)]] . \quad (4.6)$$

Similar to before, the final classifier can be obtained via

$$p_\theta(y|x) = \frac{1}{K} \int_{\mathbb{H}} \int_{\mathbb{Z}} \sum_{d \in \{0,1\}} p_\theta(x|h,z) p_\theta(h|y) p_\theta(z|y) p_\theta(y) p_\theta(d) dz dh,$$

where

$$K = \int_{\mathbb{H}} \int_{\mathbb{Z}} \sum_y \sum_{d \in \{0,1\}} p_\theta(x|h,z) p_\theta(h|y) p_\theta(z|y) p_\theta(y) p_\theta(d) dz dh$$

is the normalization constant.

Now onto Problem 3 and 4. To solve the intractability of the expectation with respect to H and Z , we employ the variational inference approach in [32, 45], and proceed to devise the inference model for the latent variables H , Z , and Y as shown in Figure 4.3:

$$\begin{aligned} q_\phi(h, z|x, y, d) &= q_\phi(h|x, y) q_\phi(z|x, d) && \text{for } y \text{ observed} \\ q_\phi(h, z, y|x, d) &= q_\phi(h|x, y) q_\phi(y|x) q_\phi(z|x, d) && \text{for } y \text{ not observed.} \end{aligned}$$

Each component above we choose to model by

$$\begin{aligned} q_\phi(h|x, y) &= \mathcal{N}(\hat{\mu}_{h|x,y}, \hat{\Sigma}_{h|x,y}); & q_\phi(y|x) &= \text{Cat}(y|\hat{\pi}_{y|x}); \\ q_\phi(z|x, d) &= \mathcal{N}(\hat{\mu}_{z|x,d}, \hat{\Sigma}_{z|x,d}), \end{aligned}$$

where each set of parameters is the output of a respective neural network:

$$\begin{aligned} f_\phi^h &: \mathbb{X} \times \mathbb{Y} \rightarrow \mathbb{H} \times \mathbb{R}^{n_h \times n_h}, & (\hat{\mu}_{h|x,y}, \hat{\Sigma}_{h|x,y}) &= f_\phi^h(x, y); \\ f_\phi^z &: \mathbb{X} \times \{0, 1\} \rightarrow \mathbb{Z} \times \mathbb{R}^{n_z \times n_z}, & (\hat{\mu}_{z|x,d}, \hat{\Sigma}_{z|x,d}) &= f_\phi^z(x, d); \\ f_\phi^y &: \mathbb{X} \rightarrow \Delta^K, & \hat{\pi}_{y|x} &= f_\phi^y(x). \end{aligned}$$

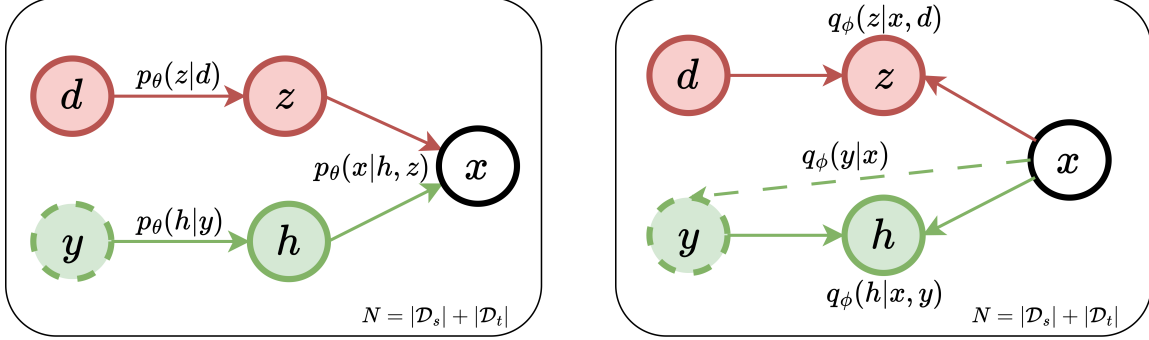


Figure 4.3: The probabilistic graphical model of our framework. Left: The probabilistic graphical model for the generative model $p_\theta(x, h, z, y, d)$. Right: Variational approximations $q_\phi(h, z|x, y, d)$ and $q_\phi(h, z, y|x, d)$ to the posteriors $p_\theta(h, z|x, y, d)$ and $p_\theta(h, z, y|x, d)$ respectively. Dashed circle around y denotes that it is partially observed.

In this case, $q_\phi(y|x)$ approximates the posterior $p_\theta(y|x)$ and allows direct inference for classification. Furthermore, now objective (4.6) can be optimized through the tractable variational lower-bound on the terms:

$$\begin{aligned} \log p_\theta(x, y|d) &= \log \mathbb{E}_{H, Z \sim p_\theta(h, z|y, d)} [p_\theta(x|H, Z)p_\theta(y)], \\ \log p_\theta(x|d) &= \log \mathbb{E}_{H, Z \sim p_\theta(h, z|d)} [p_\theta(X|H, Z)]. \end{aligned}$$

The bound on $p_\theta(x, y|d)$ corresponds to the case when y is observed, and it is given by

$$\begin{aligned} \log p_\theta(x, y|d) &\geq \mathbb{E}_{H, Z \sim q_\phi(h, z|x, y, d)} [\log p(x|H, Z)] - D_{\text{KL}}[q_\phi(h|x, y) \| p_\theta(h|y)] \\ &\quad - D_{\text{KL}}[q_\phi(z|x, d) \| p_\theta(z|d)] + \log p_\theta(y). \end{aligned}$$

For notational convenience, we denote the lower-bound above by

$$\begin{aligned} \mathcal{L}_s(x, y|d) &:= \mathbb{E}_{H, Z \sim q_\phi(h, z|x, y, d)} [\log p(x|H, Z)] - D_{\text{KL}}[q_\phi(h|x, y) \| p_\theta(h|y)] \\ &\quad - D_{\text{KL}}[q_\phi(z|x, d) \| p_\theta(z|d)] + \log p_\theta(y). \end{aligned}$$

The bound on $p_\theta(x|d)$ corresponds to the case when y is not observed, and it is

$$\begin{aligned} \log p_\theta(x|d) &\geq \sum_y q_\phi(y|x) \{ \mathbb{E}_{H, Z \sim q_\phi(h, z|x, y, d)} [\log p_\theta(x|H, Z)] - D_{\text{KL}}[q_\phi(h|x, y) \| p_\theta(h|y)] \} \\ &\quad - D_{\text{KL}}[q_\phi(z|x, d) \| p_\theta(z|d)] - D_{\text{KL}}[q_\phi(y|x) \| p_\theta(y)]. \end{aligned}$$

We denote this lower-bound by

$$\begin{aligned} \mathcal{L}_u(x|d) := & \sum_y q_\phi(y|x) \left\{ \mathbb{E}_{H,Z \sim q_\phi(h,z|x,y,d)} [\log p_\theta(x|H,Z)] - D_{\text{KL}}[q_\phi(h|x,y) \| p_\theta(h|y)] \right\} \\ & - D_{\text{KL}}[q_\phi(z|x,d) \| p_\theta(z|d)] - D_{\text{KL}}[q_\phi(y|x) \| p_\theta(y)]. \end{aligned} \quad (4.7)$$

Derivations of the above lower-bounds can be found in Appendix A. Using the variational lower-bounds in terms of the inner expectations in (4.6) yields the objective below that can be optimized efficiently using stochastic gradient descent:

$$\begin{aligned} & - \mathbb{E}_{X,Y \sim p(x,y|d=0)} [\mathcal{L}_s(X,Y|d=0)] - \mathbb{E}_{X \sim p(x|d=1)} [\mathcal{L}_u(X|d=1)] \\ & \approx - \frac{1}{|\mathcal{D}_s|} \sum_{(x_i,y_i) \in \mathcal{D}_s} \mathcal{L}_s(x_i,y_i|d=0) - \frac{1}{|\mathcal{D}_t|} \sum_{x_i \in \mathcal{D}_t} \mathcal{L}_u(x_i|d=1), \end{aligned} \quad (4.8)$$

where in the last line the two outer expectations are replaced with the respective finite-sample estimates.

4.3 Implementation

Recall our objective has become the minimization of (4.8) with neural networks embedded in q_ϕ and p_θ under the latent variable model given in Figure 4.3.

Implementation-wise, we have empirically found that the KL-divergence terms involving $q_\phi(h|x,y)$ and $q_\phi(z|x,d)$ in $\mathcal{L}_s(x,y|d)$ and $\mathcal{L}_u(x|d)$ to cause gradient instability when optimized jointly with the conditional priors $p_\theta(h|y)$ and $p_\theta(z|d)$ due to oscillation in the KL-divergence. To remedy this, we alternate between updating the conditional priors and the rest of the terms through out the optimization process.

In addition, note that the classifier $q_\phi(y|x)$ only contributes to $\mathcal{L}_u(x_i|d=1)$ in (4.8) and hence only learns on the target domain, which is undesirable as labeled data are directly available from the source and provide additional training. A resolution for this is to introduce a direct classification loss on $q_\phi(y|x)$ as done in [45]. However, we have found the typical classification loss to have significantly lower variance than the rest of the components in our objective function and hence overrules the training in practice. Thus, we instead incorporate an additional unsupervised term $\mathcal{L}_u(x_i|d=0)$ into (4.8) to enable classifier learning on the source domain:

$$\mathcal{J}(\mathcal{D}_s, \mathcal{D}_t) := - \frac{\gamma_s}{2|\mathcal{D}_s|} \sum_{(x_i,y_i) \in \mathcal{D}_s} \{ \mathcal{L}_s(x_i,y_i|d=0) + \mathcal{L}_u(x_i|d=0) \} - \frac{\gamma_t}{|\mathcal{D}_t|} \sum_{x_i \in \mathcal{D}_t} \mathcal{L}_u(x_i|d=1), \quad (4.9)$$

where γ_s and γ_t are tuning parameters controlling the relative weighting of the source and target loss terms. The added term $\mathcal{L}_u(x_i|d=0)$ behaves similarly to subsampling of \mathcal{D}_s to form another unlabeled source dataset that is used in addition to train the classifier.

The overall training procedure of our framework is summarized in Algorithm 1. Once training completes, the approximate posterior $q_\phi(y|x)$ is used to perform classification on the target domain, providing the solution to our original UDA problem: *A good classifier on the target domain.*

Algorithm 1 Training Procedure for Our Approach

Require: Source and target datasets: $\mathcal{D}_s, \mathcal{D}_t$.

Tuning parameters: γ_s, γ_t .

Neural networks parameterized by ϕ, θ : $f_\phi^h, f_\phi^z, f_\phi^y, g_\theta$.

Prior parameters: $\psi = [\mu_{h|y}, \Sigma_{h|y}, \pi_y, \mu_{z|d}, \Sigma_{z|d}, \pi_d]$.

function SAMPLE()

$h_i^s \sim q_\phi(h|x_i, y); \quad z_i^s \sim q_\phi(z|x_i, d=0); \quad \forall x_i \in B_s, y \in \mathbb{Y}$

$h_i^t \sim q_\phi(h|x_i, y); \quad z_i^t \sim q_\phi(z|x_i, d=1); \quad \forall x_i \in B_t, y \in \mathbb{Y}$

end function

$\phi, \theta \leftarrow$ Neural network parameter initialization

$\mu_{h|y}[y], \Sigma_{h|y}[y] \leftarrow \mathbf{0}, \mathbf{I} \quad \forall y \in \mathbb{Y}$

$\mu_{z|d}[d], \Sigma_{z|d}[d] \leftarrow \mathbf{0}, \mathbf{I} \quad \forall d \in \{0, 1\}$

$\pi_y \leftarrow [\frac{1}{K}, \frac{1}{K}, \dots]$

repeat

while TRAINNETWORKS() **do**

$B_s, B_t \leftarrow$ Random mini-batch of size N from $\mathcal{D}_s, \mathcal{D}_t$

 SAMPLE()

$\nabla_{\theta, \phi} \leftarrow \nabla_{\theta, \phi} \mathcal{J}(B_s, B_t)$

$\theta, \phi \leftarrow$ Parameter update using gradient descent with $\nabla_{\theta, \phi}$

end while

while TRAINPRIORS() **do**

$B_s, B_t \leftarrow$ Random mini-batch of size N from $\mathcal{D}_s, \mathcal{D}_t$

 SAMPLE()

$\nabla_\psi \leftarrow \nabla_\psi \mathcal{J}(B_s, B_t)$

$\psi \leftarrow$ Parameter update using gradient descent with ∇_ψ

end while

until convergence of θ, ϕ

4.3.1 Distribution Constraints

We discuss our method to enforce constraints on the parameters of distributions used in our latent variable model in this section.

Categorical Distribution The categorical distribution with C categories, denoted by $\text{Cat}(\cdot|\pi)$, is a probability mass function parameterized by a vector of C elements $\pi = [p_1, \dots, p_C]$ satisfying

$$p_i \geq 0 \quad \forall i = 1, \dots, C \quad \text{and} \quad \sum_{i=1}^C p_i = 1, \quad (4.10)$$

with p_i being the probability of selecting the i -th category. To enforce the constraint in (4.10) for $p_\theta(y)$, we store a parameter vector $\mathbf{v}_{\pi_y} \in \mathbb{R}^K$ for $p_\theta(y)$ and apply the softmax function $\sigma : \mathbb{R}^K \rightarrow \Delta^K$ defined as

$$\begin{aligned} \sigma(\mathbf{v}_{\pi_y})_i &= \frac{e^{\mathbf{v}_{\pi_y} i}}{\sum_{j=1}^K e^{\mathbf{v}_{\pi_y} j}} \\ \sigma(\mathbf{v}_{\pi_y}) &= [\sigma(\mathbf{v}_{\pi_y})_1, \dots, \sigma(\mathbf{v}_{\pi_y})_K]^T, \end{aligned}$$

and parameterize $p_\theta(y)$ by $\pi_y = \sigma(\mathbf{v}_{\pi_y})$; and similarly for $p_\theta(d)$. In terms of $p_\theta(y|x)$, which is parameterized by $f_\phi^y(x)$, we design f_ϕ^y to output a parameter vector $\mathbf{v}_{\hat{\pi}_{y|x}} \in \mathbb{R}^K$ and use $\hat{\pi}_{y|x} = \sigma(\mathbf{v}_{\hat{\pi}_{y|x}})$ to parameterize $p_\theta(y|x)$.

Multivariate Normal Distribution The multivariate normal distribution is only well-defined given a symmetric positive-definite covariance matrix. To enforce this constraint on the covariance matrices of $p_\theta(h|y)$, $p_\theta(z|d)$, $q_\phi(h|x, y)$, and $q_\phi(z|x, d)$, we parametrize the covariance matrices through their Cholesky factors instead. More specifically, the Cholesky decomposition of a symmetric positive-definite matrix \mathbf{M} is defined as

$$\mathbf{M} = \mathbf{L}\mathbf{L}^T,$$

where \mathbf{L} is termed the *Cholesky factor* and is a lower-triangular matrix with real and positive diagonal entries. Since every symmetric positive-definite matrix has a unique Cholesky decomposition, parameterizing the covariance matrices by their Cholesky factors enforces the constraint on symmetry and positive-definiteness.

This means that in implementation, we store elements to the Cholesky factor of the covariance of $p_\theta(h|y)$ as parameters: for each $\Sigma_{h|y}[y]$, that is a parameter vector $\mathbf{v}_{\Sigma_{h|y}[y]} \in$

$\mathbb{R}^{\frac{n_h(n_h+1)}{2}}$; similarly for each $\Sigma_{z|d}[d]$ of $p_\theta(z|d)$. In terms of the covariance of $q_\phi(h|x, y)$, which is an output from the neural network f_ϕ^h , we design f_ϕ^h to output the elements to the Cholesky factor represented by an output vector $\mathbf{v}_{\hat{\Sigma}_{h|x,y}} \in \mathbb{R}^{\frac{n_h(n_h+1)}{2}}$; similarly for f_ϕ^z of $q_\phi(z|x, d)$. Lastly, to ensure that the Cholesky factors have positive diagonal entries, we exponentiate the elements corresponding to these diagonal entries in the parameter vectors.

4.3.2 Reparametrization Trick

Estimates to the expectation terms in $\mathcal{L}_s(x, y|d)$ and $\mathcal{L}_u(x|d)$ are obtained through sampling L samples of $q_\phi(h, z|x, y, d)$, $(h^{(l)}, z^{(l)}) \sim q_\phi(h, z|x, y, d)$, and forming the sample estimate

$$\mathbb{E}_{H, Z \sim q_\phi(h, z|x, y, d)} [\log p_\theta(x|H, Z)] \approx \frac{1}{L} \sum_{l=1}^L \log p_\theta(x|h^{(l)}, z^{(l)}) \quad (4.11)$$

$$\text{where } h^{(l)} \sim q_\phi(h|x, y); \quad z^{(l)} \sim q_\phi(z|x, d).$$

However, similar to VAE [32], using this estimate would require gradients to backpropagate through the stochastic samples for training the neural networks of $q_\phi(h|x, y)$ and $q_\phi(z|x, y)$, which is not realizable in practice because the sampling operation is not differentiable. To solve this problem, we apply the *reparametrization trick* in [32].

Specifically, denote $r(h, z) = \log p_\theta(x|h, z)$, we can rewrite the expectation in (4.11) into

$$\begin{aligned} & \mathbb{E}_{H, Z \sim q_\phi(h, z|x, y, d)} [r(H, Z)] \\ &= \mathbb{E}_{H \sim q_\phi(h|x, y), Z \sim q_\phi(z|x, d)} [r(H, Z)] \\ &= \mathbb{E}_{E_h \sim \mathcal{N}(0, \mathbf{I}), E_z \sim \mathcal{N}(0, \mathbf{I})} [r(\hat{\mu}_{h|x, y} + \mathbf{L}_{h|x, y} E_h, \hat{\mu}_{z|x, d} + \mathbf{L}_{z|x, d} E_z)] \\ & \text{where } \mathbf{L}_{h|x, y} \mathbf{L}_{h|x, y}^T = \hat{\Sigma}_{h|x, y}; \quad \mathbf{L}_{z|x, d} \mathbf{L}_{z|x, d}^T = \hat{\Sigma}_{z|x, d}. \end{aligned}$$

Here, $\hat{\mu}_{h|x, y}$ is the mean and $\mathbf{L}_{h|x, y}$ is the Cholesky factor to the covariance of $q_\phi(h|x, y)$, both are outputs from f_ϕ^h ; similarly for $\hat{\mu}_{z|x, d}$ and $\mathbf{L}_{z|x, d}$. The derivation above follows from a change of variable from E_h to H and E_z to Z , e.g. $E_h \sim \mathcal{N}(0, \mathbf{I}) \implies H \sim \mathcal{N}(\hat{\mu}_{h|x, y}, \mathbf{L}_{h|x, y} \mathbf{L}_{h|x, y}^T)$ for $H = \hat{\mu}_{h|x, y} + \mathbf{L}_{h|x, y} E_h$. This results in the below equivalent

estimator to (4.11):

$$\mathbb{E}_{H,Z \sim q_\phi(h,z|x,y,d)} [\log p_\theta(x|H, Z)] \approx \frac{1}{L} \sum_{l=1}^L r(\hat{\mu}_{h|x,y} + \mathbf{L}_{h|x,y} \epsilon_h^{(l)}, \hat{\mu}_{z|x,d} + \mathbf{L}_{z|x,d} \epsilon_z^{(l)})$$

where $\epsilon_h^{(l)} \sim \mathcal{N}(0, \mathbf{I}); \quad \epsilon_z^{(l)} \sim \mathcal{N}(0, \mathbf{I}),$

which is differentiable with respect to $(\hat{\mu}_{h|x,y}, \mathbf{L}_{h|x,y}, \hat{\mu}_{z|x,d}, \mathbf{L}_{z|x,d})$, enabling gradients to backpropagate to $q_\phi(h|x, y)$ and $q_\phi(z|x, y)$.

4.3.3 Computing KL-Divergence

The lower-bounds $\mathcal{L}_s(x, y|d)$ and $\mathcal{L}_u(x|d)$ contain KL-divergence terms between the variational posteriors and the prior distributions, which are all normal distributions. The KL-divergence between two N -dimensional multivariate normal distributions has a closed-form and is given by [46]:

$$D_{\text{KL}}[\mathcal{N}(\mu_1, \Sigma_1) \parallel \mathcal{N}(\mu_2, \Sigma_2)] = \frac{1}{2} \left\{ \text{tr}(\Sigma_2^{-1} \Sigma_1) + (\mu_2 - \mu_1) \Sigma_2^{-1} (\mu_2 - \mu_1) + \log \frac{|\Sigma_2|}{|\Sigma_1|} - N \right\}.$$

4.3.4 Domain-Specific Batch Normalization

The gradient of (4.9) exhibits high-variance due to the stochastic sampling involved in estimating the expectations in $\mathcal{L}_s(x, y|d)$ and $\mathcal{L}_u(x|d)$, which negatively impact convergence rate of the various neural networks in our framework. One way to combat this effect is to employ batch normalization [47]. Consider a d -dimensional input $\mathbf{x} = [x_1, \dots, x_d]$ to a layer within our neural network. Let $\mathbf{X} = [\mathbf{x}^{(1)}; \dots; \mathbf{x}^{(N)}]$ denotes a mini-batch, where each \mathbf{x}_i is a row of \mathbf{X} . Batch normalization is summarized in Algorithm 2.

However, directly applying batch normalization in UDA uniformly across source and target mini-batches results in $\boldsymbol{\mu}$ being the mean over all samples in both domains, and similarly for \mathbf{v} . Hence, the activations for the samples deviates from zero mean and unit variance when considering the domains individually, which causes large activations on average when the domains have significantly disperse feature distributions.

To resolve this, we utilize a batch normalization scheme similar to [48], where the batch normalization is simply performed on a per-domain basis. Denote the domain-specific quantities by subscript s or t respectively, the revised batch normalization is shown in Algorithm 3.

Algorithm 2 Batch Normalization [47]

Require: Parameters to be learned: $\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}^n$.

Input: Mini-batch $\mathbf{X} \in \mathbb{R}^{N \times d}$.

Output: Batch normalized mini-batch $\mathbf{Y} \in \mathbb{R}^{N \times d}$.

$$\begin{aligned}\boldsymbol{\mu} &\leftarrow \frac{1}{N} \sum_{i=1}^N \mathbf{X}_{i,:} \\ \mathbf{v} &\leftarrow \frac{1}{N} \sum_{i=1}^N (\mathbf{X}_{i,:} - \boldsymbol{\mu})^2 \\ \bar{\mathbf{X}}_{i,:} &\leftarrow \frac{\mathbf{X}_{i,:} - \boldsymbol{\mu}}{\sqrt{\mathbf{v} + \epsilon}} \quad \forall i \in \{1, \dots, N\} \\ \mathbf{Y}_{i,:} &\leftarrow \boldsymbol{\alpha}^T \bar{\mathbf{X}}_{i,:} + \boldsymbol{\beta}\end{aligned}$$

Algorithm 3 Domain-Specific Batch Normalization

Require: Parameters to be learned: $\alpha_s, \alpha_t \in \mathbb{R}^n, \beta_s, \beta_t \in \mathbb{R}^n$.

Input: Mini-batch $\mathbf{X}_s, \mathbf{X}_t \in \mathbb{R}^{N \times d}$.

Output: Batch normalized mini-batch $\mathbf{Y}_s, \mathbf{Y}_t \in \mathbb{R}^{N \times d}$.

$$\begin{aligned}\boldsymbol{\mu}_s &\leftarrow \frac{1}{N} \sum_{i=1}^N \mathbf{X}_{s,i,:} \\ \mathbf{v}_s &\leftarrow \frac{1}{N} \sum_{i=1}^N (\mathbf{X}_{s,i,:} - \boldsymbol{\mu}_s)^2 \\ \bar{\mathbf{X}}_{s,i,:} &\leftarrow \frac{\mathbf{X}_{s,i,:} - \boldsymbol{\mu}_s}{\sqrt{\mathbf{v}_s + \epsilon}} \quad \forall i \in \{1, \dots, N\} \\ \mathbf{Y}_{s,i,:} &\leftarrow \boldsymbol{\alpha}_s^T \bar{\mathbf{X}}_{s,i,:} + \boldsymbol{\beta}_s \\ &\vdots\end{aligned}$$

▷ Repeat the above with subscript 's' replaced by 't'

4.3.5 Overall Architecture

Figure 4.4 shows an overview of the architecture of our framework. We denote the domain of the i -th sample by d_i . There are two data-paths for computing $\mathcal{L}_s(x_i, y_i|d_i)$ and $\mathcal{L}_u(x_i|d_i)$ respectively:

- When computing $\mathcal{L}_s(x_i, y_i|d_i)$: y_i is available and directly inputted into f_ϕ^h , bypassing the classifier network f_ϕ^y .
- When computing $\mathcal{L}_u(x_i|d_i)$: y_i is unavailable/masked. Each x_i is inputted into the classifier network f_ϕ^y first to obtain $q_\phi(y|x_i)$. Then $q_\phi(y|x_i)$ is used to compute the expectation with respect to Y in $\mathcal{L}_u(x_i|d_i)$.

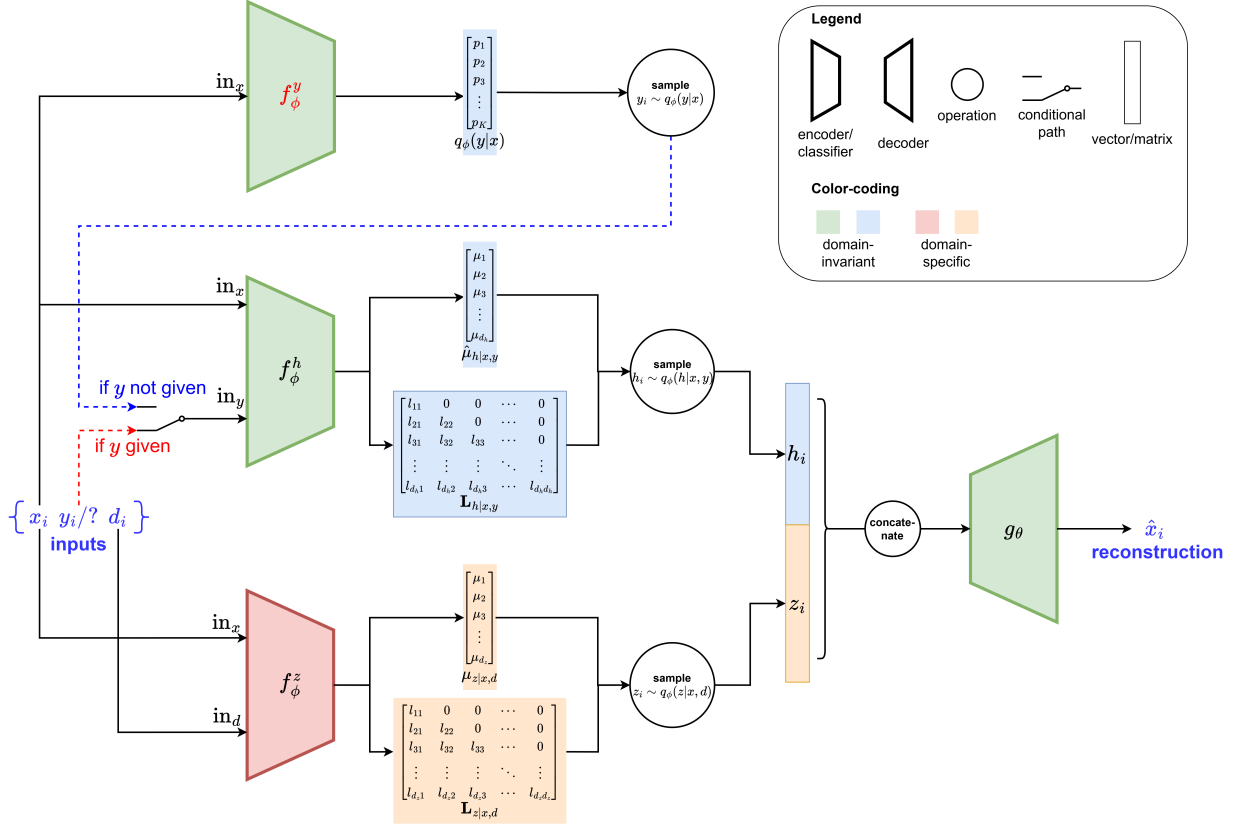


Figure 4.4: The overall architecture of our proposed framework. There are two data-paths depending on whether y_i is given, depicted by dashed red and blue lines respectively. The data-path for missing y_i involves computing an expectation over $q_\phi(y|x_i)$ for each label. We replace this expectation by its sample estimate here to better illustrate the relationship between f_ϕ^y and the rest of the components.

Chapter 5

Empirical Results

In this chapter, we describe the experiment to validate our approach on example UDA scenarios against other common UDA approaches and present the related results. We follow up with a brief demonstration of our approach to solve the related domain translation problem. Lastly, we discuss the limitations of our approach we discovered. Implementation specifics of the experiment are presented at the end.

The code to our experiment is available at: <https://github.com/xuanrui-work/vdat-pyro>.

5.1 Setup of Experiment

To test our proposed approach empirically, we use image classification as the supervisory task, where the task consists of classifying the digit (0-9) presented in an 32×32 image. For this task, the input space $\mathbb{X} \subset \mathbb{R}^{3072}$ is the space of all RGB images of size 32×32 containing a digit; and the output space $\mathbb{Y} = \{0, \dots, 9\}$ is the space of possible labels. We follow the maximum-likelihood paradigm introduced in Example 2.2.2 where the model is a conditional probability distribution $f : \mathbb{X} \rightarrow \Delta^{10}$ that seeks to approximate $p(y|x)$, and the final classification rule is given by

$$\hat{y} = \arg \max_{y \in \mathbb{Y}} f(x)_y$$

as in (2.5). See Figure 5.1 for an illustration of this task.

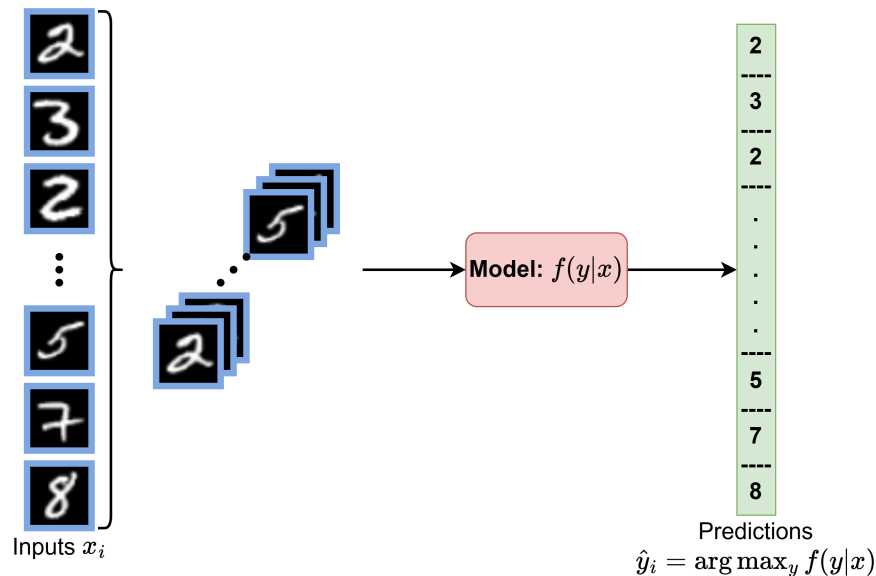


Figure 5.1: An illustration of our digit classification task for experiments and comparisons with different UDA approaches. The figure depicts the desired result of correctly predicting the digits in the given set of images.

With regards to the dataset, the training dataset consists of pairs (x_i, y_i) where x_i is the input image representing the digit; and y_i is the label, the desired categorization of the digit generated by human labor.

We frame the above supervised learning task into an UDA problem by designating the above dataset as the source dataset \mathcal{D}_s , and introduce an unlabeled target dataset \mathcal{D}_t which consists images of digit that contain visually distinct features from the source dataset. Details are presented in Section 5.1.1.

5.1.1 Domains/Datasets

As discussed in Section 3.1, UDA concerns with two datasets with different input distributions $p(x|d)$ while assuming identical labeling distribution $p(y|x)$, where the source dataset \mathcal{D}_s is labeled and the target dataset \mathcal{D}_t is unlabeled. To facilitate this idea, we choose three public datasets for digit recognition, MNIST [49], USPS [38], and SVHN [50]. In addition, we modify the MNIST dataset to generate two new datasets: MNIST-r is the MNIST dataset with all digits rotated 45-degrees counterclockwise; and MNIST-s is the MNIST dataset with all digits shrunk by a scale factor of 0.5. Samples from each dataset

are illustrated in Figure 5.2. Each dataset contains visually distinct features from each others, which induces different input distributions and creates the notion of domains.

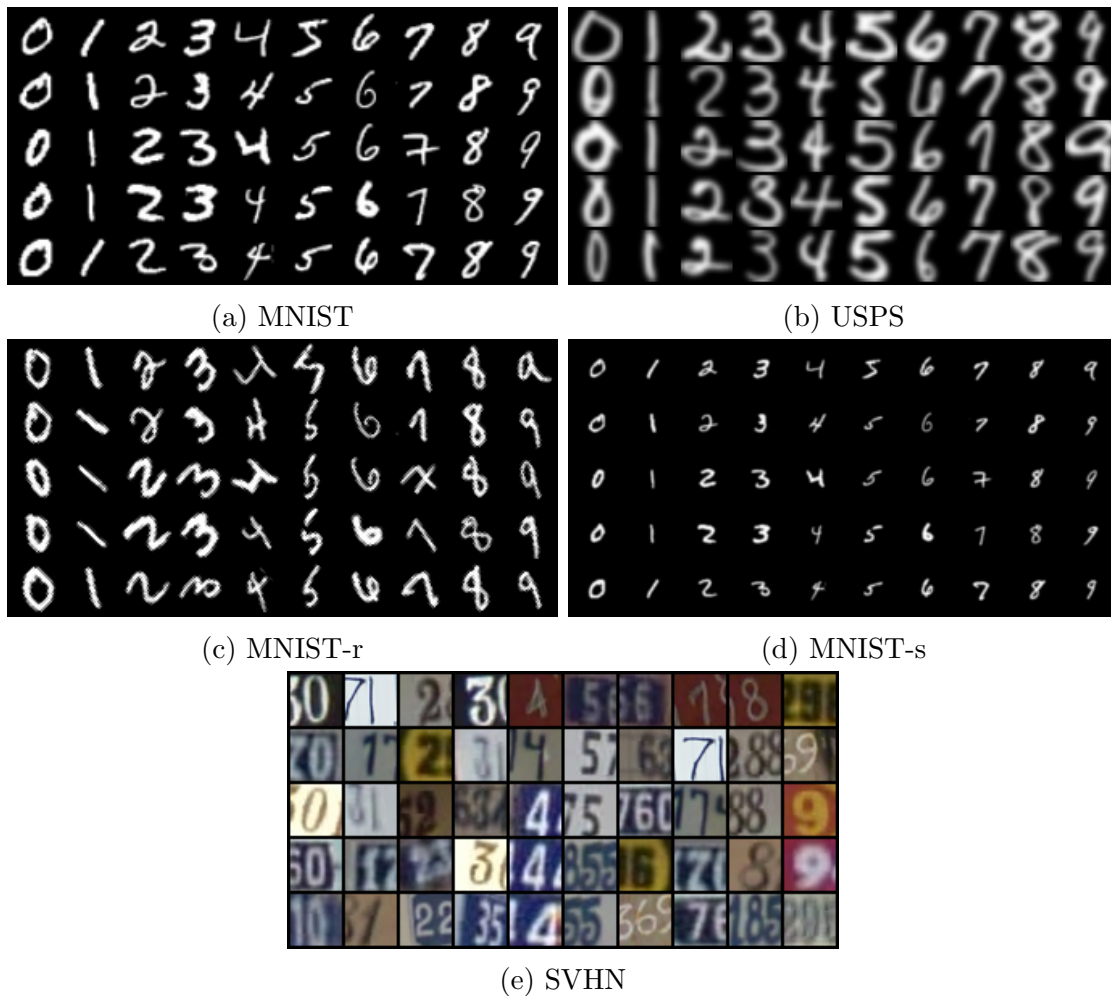


Figure 5.2: Input samples from the five datasets used in our experiment.

By picking one of the datasets as the source dataset and another as the target, different UDA scenarios can be simulated by discarding the labels in the chosen target dataset during training. Based off this, we generate the following list of scenarios for UDA: MNIST→USPS, MNIST→MNIST-r, MNIST→MNIST-s, SVHN→MNIST, where we denote each UDA scenario by the notation *source domain*→*target domain*. For example, MNIST→USPS means to pick the MNIST dataset as samples from the source domain

and the USPS dataset as samples from the target domain, under which case we have access to the labels on MNIST but not for USPS.

With regards to pre-processings, we resize all input images to be of size 32×32 with bilinear interpolation, and convert grayscale image to RGB by repeating the grayscale channel for each component of the RGB. The pixels in the final image are then normalized from the range $(0 - 255)$ to the range $(0 - 1)$ by rescaling. Most importantly, no transformation is used to help reduce the domain shift for the model in any scenarios.

5.1.2 Procedures

All approaches in our experiment are implemented in PyTorch [51], including our approach and the chosen baselines. In addition, we make use of Pyro [52] for the implementation of variational inference and the optimization of (4.9) in our approach.

For the source and target datasets. Each dataset is split into three subsets: the training set, the validation set, and the test set, where the training set is the set of samples used in training the model; the validation set is isolated from training, used to measure the accuracy of the model on unseen samples for model tuning; and the test set is isolated from both training and tuning, it is the set of samples we test the final model and report the accuracy upon.

For training, each approach is trained for a fixed number of epochs, where an epoch is the number of steps needed to process the smaller of the source and target training set, and both the source and target training sets are shuffled at the start of each epoch. After each epoch, the accuracy of the classifier in the target domain is measured through the target validation set. In the end, the model with the highest accuracy on the validation set is used as the final model for testing on the target test set, which we report as the classification accuracy on the target domain. We also report the accuracy of this model on the source test set.

Lastly, tuning parameters of each approach are tuned to maximize the accuracy on the target validation set. These include: number of epochs, the learning rate, other parameters to the optimizer, and parameters tailored to each approach. The ranges of tuning parameters used in our approach are listed in Appendix B.

We compare our approach against the following list of UDA approaches:

1. *Re-weight*: Learning and Evaluating Classifiers under Sample Selection Bias [11].
2. *CORAL*: Deep CORAL: Correlation alignment for deep domain adaptation [15].

3. *DANN*: Domain-adversarial training of neural networks [31].

And the following methods are used as the baselines for UDA:

1. *Src-Only*: Model trained using supervised learning on the source dataset only.
2. *Joint*: Model trained using supervised learning on *jointly the labeled source and target datasets*.

In particular, the Src-Only method is used to obtain a basemark performance of the model in the target domain without adaptation, and the Joint method estimates the best achievable performance in the target domain through UDA.

Details on the implementation of our approach and each baseline is available in Section 5.3.

5.2 Results

The test accuracies on the target domain for all approaches are included in Table 5.1. From the results, we see that our approach achieves the best accuracy on the simpler UDA scenarios. Especially for MNIST→MNIST-r, where the domain shift involves a rotation on the image plane that is difficult for the neural network to generalize to without explicit training on labeled data, our approach has covered 86% of the performance gap between the Src-Only model and the Joint model.

5.2.1 Capability on Domain Translation

Interestingly, our approach can be easily adopted to perform domain translation as mentioned in Section 1.2, where the goal is to make the data from the source domain to appear as that from the target domain. Normally, this means to find a transformation $M_{x_s \rightarrow x_t} : \mathbb{X} \rightarrow \mathbb{X}$ that maps an input X with a source distribution $p(x|d=0)$ to distribute alike the target distribution $p(x|d=1)$, i.e. we desire

$$\mathbb{P}(M_{x_s \rightarrow x_t}(X) \leq x | D = 0) = \mathbb{P}(X \leq x | D = 1).$$

We consider our approach to domain translation here as semi-supervised, since our framework does utilize labeled source data but does not need paired data across the domains as in the fully-supervised case.

Method	M→U	M→M-s	M→M-r	S→M
Src-Only*	96.09% / 73.96%	99.15% / 67.85%	99.29% / 58.22%	94.07% / 64.65%
Joint*	95.47% / 95.21%	99.10% / 99.03%	99.07% / 98.76%	93.37% / 98.96%
Re-weight	98.11% / 77.67%	99.21% / 70.78%	99.39% / 58.93%	95.91% / 71.72%
CORAL	98.76% / 87.04%	99.42% / 87.83%	99.37% / 74.55%	96.40% / 68.78%
DANN	98.70% / 90.10%	99.90% / 88.65%	99.04% / 74.28%	96.57% / 74.98%
Ours	97.53% / 95.72%	98.85% / 92.07%	98.90% / 93.04%	DNC

Table 5.1: Classification accuracies of our UDA approach and other UDA approaches for different UDA scenarios. Each entry is “accuracy on source test set / accuracy on target test set”. Different UDA scenarios are abbreviated by: MNIST→USPS (M→U), MNIST→MNIST-s (M→M-s), MNIST→MNIST-r (M→M-r), SVHN→MNIST (S→M). *DNC* denotes “did not converge”. * marks baseline methods without the use of UDA.

To establish domain translation using our framework, we note that the variation across the domains are solely introduced from the domain-specific factors Z in our latent variable model. Hence, to align $p_\theta(x|d=0)$ and $p_\theta(x|d=1)$, it suffices to find a transformation $M_{z_s \rightarrow z_t} : \mathbb{Z} \rightarrow \mathbb{Z}$ that aligns $p_\theta(z|d=0)$ and $p_\theta(z|d=1)$. Given that both of these distributions are restricted to be multivariate normal, specifically

$$p_\theta(z|d=0) = \mathcal{N}(\mu_{z|d}[0], \Sigma_{z|d}[0]); \quad p_\theta(z|d=1) = \mathcal{N}(\mu_{z|d}[1], \Sigma_{z|d}[1]),$$

$p_\theta(z|d=0)$ can be transformed into $p_\theta(z|d=1)$ via an affine transformation.

Denote the eigendecomposition of $\Sigma_{z|d}[0]$ and $\Sigma_{z|d}[1]$ respectively by

$$\Sigma_{z|d}[0] = \mathbf{Q}_s \mathbf{\Lambda}_s \mathbf{Q}_s^T; \quad \Sigma_{z|d}[1] = \mathbf{Q}_t \mathbf{\Lambda}_t \mathbf{Q}_t^T.$$

Since $\Sigma_{z|d}[0]$ and $\Sigma_{z|d}[1]$ are symmetric positive-definite, the above decomposition is guaranteed to exist, where $\mathbf{Q}_s, \mathbf{Q}_t$ are orthogonal and $\mathbf{\Lambda}_s, \mathbf{\Lambda}_t$ have positive diagonals. Without loss of generality, we assume eigenvalues placed in descending order on the diagonal of $\mathbf{\Lambda}_s, \mathbf{\Lambda}_t$. The principal components of $p_\theta(z|d=0)$ and $p_\theta(z|d=1)$ can then be aligned through the affine transformation given by

$$M_{z_s \rightarrow z_t}(z) = \mathbf{R}z + \mathbf{b}$$

where $\mathbf{R} = \mathbf{Q}_t (\mathbf{\Lambda}_t^{1/2} \mathbf{\Lambda}_s^{-1/2}) \mathbf{Q}_s^T; \quad \mathbf{b} = \mu_{z|d}[1] - \mathbf{R} \mu_{z|d}[0].$

Based on this, we develop the procedure for domain translation described in Algorithm 4. The same procedure can be used to translate target samples into source by interchanging the roles of the two domains.

Algorithm 4 Domain Translation with Our Approach

Require: Trained neural networks: $f_\phi^h, f_\phi^z, f_\phi^y, g_\theta$.

Trained prior parameters: $\psi = [\mu_{z|d}, \Sigma_{z|d}]$.

Input: Source sample x_s .

Output: Translated source sample $\hat{x}_{s \rightarrow t}$.

$\mathbf{Q}_s, \mathbf{\Lambda}_t \leftarrow$ Eigendecomposition of $\Sigma_{z|d}[0]$.

$\mathbf{Q}_t, \mathbf{\Lambda}_t \leftarrow$ Eigendecomposition of $\Sigma_{z|d}[1]$.

$\mathbf{R} \leftarrow \mathbf{Q}_t(\mathbf{\Lambda}_t^{1/2}\mathbf{\Lambda}_s^{-1/2})\mathbf{Q}_s^T$

$\mathbf{b} \leftarrow \mu_{z|d}[1] - \mathbf{R}\mu_{z|d}[0]$

$\hat{y} \leftarrow \arg \max_{y \in \mathbb{Y}} q_\phi(y|x = x_s)$

$h \sim q_\phi(h|x = x_s, y = \hat{y})$

$z_s \sim q_\phi(z|x = x_s, d = 0)$

$z_t \leftarrow \mathbf{R}z_s + \mathbf{b}$

$\hat{x}_{s \rightarrow t} \leftarrow g_\theta(h, z_t)$

Figure 5.3 shows our results for domain translation using MNIST-style digits as the source domain and the other datasets as the target domains. Result for each domain translation scheme is obtained using the framework trained on the corresponding domain adaptation scenario.

5.2.2 Limitations

Although our approach has demonstrated promising results for both domain adaptation and translation on toy scenarios involving simple domain shifts, our approach has two major limitations that prevent its boarder use to more realistic UDA scenarios. We provide a discussion of them in this section.

1. Numerical Instability In the SVHN \rightarrow MNIST scenario, where the domain shift is perceptually large, we observe tendency in f_ϕ^h and f_ϕ^z towards outputting singular covariance matrices during training, despite our effort in enforcing the positive definiteness constraint in Section 4.3.1. These singularities prevent the computation of the KL-divergence terms in our objective in (4.9) and the log-probability of the variational approximation q_ϕ , causing optimization with stochastic gradient descent to halt.

We suspect the above to be caused by our assumption on the intrinsic dimensions of

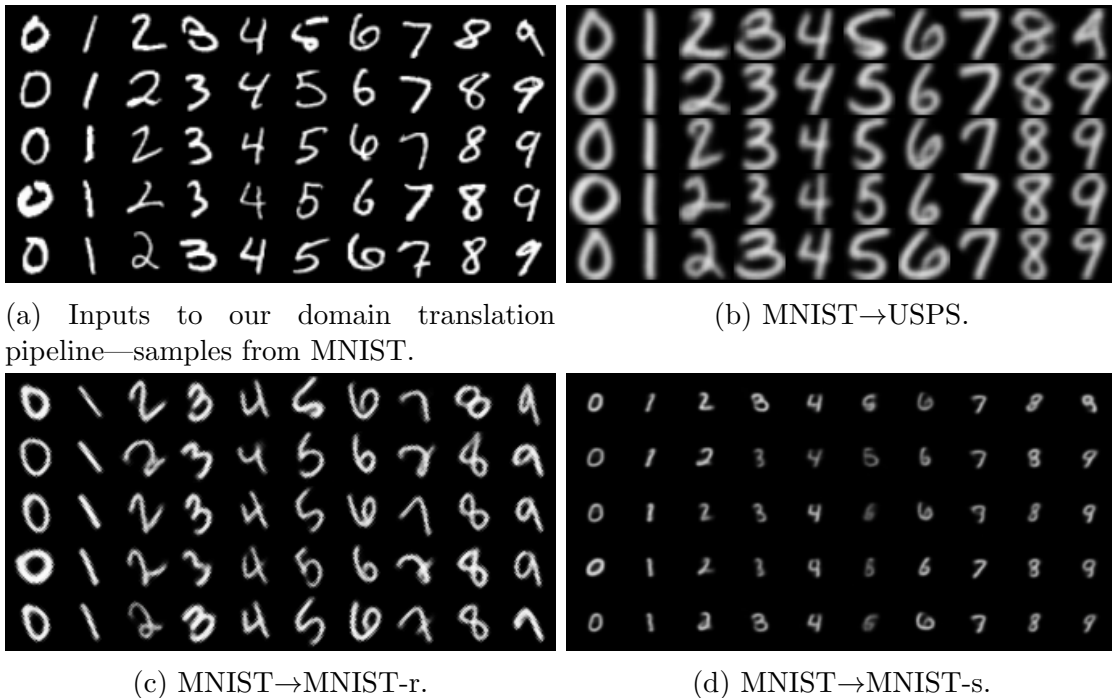


Figure 5.3: Examples demonstrating the capability of our framework for domain translation. The input samples are from the MNIST dataset. Samples in the other domains are generated by our framework using Algorithm 4.

H and Z in the generative process. Specifically, we have assumed H and Z to each follow multivariate normal distribution with the same dimension across the domains. However in the SVHN→MNIST scenario, presumably the domain-invariant and domain-specific factors of the MNIST domain are a subset of that in the SVHN domain, as seen from Figure 5.2. In such cases, the variations among factors of the MNIST samples likely lies within a subspace of that of the SVHN, hence prompting for a low-rank covariance matrix.

2. Computational Cost As can be seen from (4.7), the unsupervised term $\mathcal{L}_u(x_i|d)$ in our objective contains an expectation over Y that is computed through summing over each label $y \in \mathbb{Y}$, where for each y , one forward pass through the neural networks f_ϕ^h and g_θ is needed to obtain $q_\phi(h|x, y)$ and $p_\theta(x|h, z)$.¹ Hence, the computational cost associated with this expectation scales linearly with the number of possible labels. This makes our

¹Assume $L = 1$.

existing approach computationally expensive at handling UDA for classification tasks with a large label set, which is undesirable.

5.3 Implementation Details

This section is devoted into the implementation details of our experiment, including the specific construct of the neural networks involved, the optimization method used, and our selection of the tuning parameters for each approach.

5.3.1 Our Approach

There are four neural networks involved in our framework: the encoder for domain-invariant factors $f_\phi^h : \mathbb{X} \times \mathbb{Y} \rightarrow \mathbb{H} \times \mathbb{R}^{n_h \times n_h}$, the encoder for domain-specific factors $f_\phi^z : \mathbb{X} \times \{0, 1\} \rightarrow \mathbb{Z} \times \mathbb{R}^{n_z \times n_z}$, the decoder $g_\theta : \mathbb{H} \times \mathbb{Z} \rightarrow \mathbb{X}$, and most importantly the classifier network $f_\phi^y : \mathbb{X} \rightarrow \Delta^K$. With regards to the dimensions of the latent factors, we pick $\mathbb{H} = \mathbb{R}^{32}$ and $\mathbb{Z} = \mathbb{R}^{32}$ for all experiments. The specific construction of each network is shown in Table 5.2 and Table 5.3, where each type of layers is defined below:

1. **Linear(i,o)** – linear layer with i input dimension and o output dimension.
2. **Conv(c,k,s)** – convolution layer with c output channels, k kernel size, and s stride.
3. **MaxPool(k,s)** – max pooling layer with k kernel size and s stride.
4. **Upsample(s)** – upsample layer with s scale factor using nearest-neighbor interpolation.
5. **DSBatchNorm** – domain-specific batch normalization layer in Algorithm 3.
6. **LeakyReLU** – the LeakyReLU activation function.
7. **ConvBlock(c,k,s)** – a composition of the following layers in sequence: [Conv(c,k,s), DSBatchNorm, MaxPool(k=2,s=2), LeakyReLU].
8. **ConvBlock'(c,k,s)** – a composition of the following layers in sequence: [Conv(c,k,s), DSBatchNorm, Upsample(s=2), LeakyReLU]

Layer #	Type	Output Shape	# Parameters
1	Input+Embedding	(4, 32, 32)	11,264
2	ConvBlock(c=64, k=3, s=1)	(64, 16, 16)	2,496
3	ConvBlock(c=128, k=3, s=1)	(128, 8, 8)	74,112
4	ConvBlock(c=256, k=3, s=1)	(256, 4, 4)	295,680
5	ConvBlock(c=512, k=3, s=1)	(512, 2, 2)	1,181,184
6	Linear(i=2048, o=560)	32 + 528	1,147,440
			2,712,176

(a) Construction of f_ϕ^h .

Layer #	Type	Output Shape	# Parameters
1	Input+Embedding	(4, 32, 32)	3,072
2	ConvBlock(c=64, k=3, s=1)	(64, 16, 16)	2,496
3	ConvBlock(c=128, k=3, s=1)	(128, 8, 8)	74,112
4	ConvBlock(c=256, k=3, s=1)	(256, 4, 4)	295,680
5	ConvBlock(c=512, k=3, s=1)	(512, 2, 2)	1,181,184
6	Linear(i=2048, o=560)	32 + 528	1,147,440
			2,703,984

(b) Construction of f_ϕ^z .

Table 5.2: Construction of the neural networks f_ϕ^h and f_ϕ^z in our framework.

With regards to training, we set $\gamma_s = 0.5$, $\gamma_t = 0.5$, and $L = 1$ across all scenarios, and use the Adam [53] optimizer for stochastic gradient descent. The schedule for TRAINNETWORKS() and TRAINPRIORS() in Algorithm 1 is set to 100 optimization steps each, i.e. the neural networks are updated for 100 steps then the priors are updated for another 100 steps. Other tuning parameters are tuned for each scenario to maximize the accuracy on the target validation set.

5.3.2 Baselines

To enable fair comparisons, we make an effort towards using the same architecture as f_ϕ^y in our approach for the classifier networks in the baselines. However, similar to our approach, most baselines require add-on components aside from the classifier network. In this section, we briefly outline the implementation of the neural networks and the training procedures for the baselines.

For training, the Adam [53] optimizer is used across all baselines, and the tuning

Layer #	Type	Output Shape	# Parameters
1	Input from Encoder	32 + 32	0
2	Linear(i=64, o=2048)	2048	133,120
3	ConvBlock'(c=512, k=3, s=1)	(512, 4, 4)	2,360,832
4	ConvBlock'(c=256, k=3, s=1)	(256, 8, 8)	1,180,416
5	ConvBlock'(c=128, k=3, s=1)	(128, 16, 16)	295,296
6	ConvBlock'(c=64, k=3, s=1)	(64, 32, 32)	73,920
7	ConvBlock(c=3, k=3, s=1)	(3, 32, 32)	1,731
			4,045,315

(a) Construction of g_θ .

Layer #	Type	Output Shape	# Parameters
1	Input	(3, 32, 32)	0
2	ConvBlock(c=64, k=3, s=1)	(64, 16, 16)	1,920
3	ConvBlock(c=128, k=3, s=1)	(128, 8, 8)	74,112
4	ConvBlock(c=256, k=3, s=1)	(256, 4, 4)	295,680
5	ConvBlock(c=512, k=3, s=1)	(512, 2, 2)	1,181,184
6	Linear(i=2048, o=1024)	1024	2,098,176
7	Linear(i=1024, o=512)	512	524,800
8	Linear(i=512, o=10)	10	5,130
			4,181,002

(b) Construction of f_ϕ^y .Table 5.3: Construction of the neural networks g_θ and f_ϕ^y in our framework.

parameters for each baseline are tuned to maximize the accuracy on the target validation set.

Src-Only & Joint The Src-Only baseline and the Joint baseline both involve solely the classifier network. For each baseline, a neural network with the same architecture as f_ϕ^y is used.

Re-weight [11] We apply the bias correction principle given in [11], where the risk with respect to the target domain is obtained through a re-weighted expected loss on the source

domain:

$$\mathbb{E}_{X,Y \sim p(x,y|d=1)} [L(f_\theta(X), Y)] = \mathbb{E}_{X,Y \sim p(x,y|d=0)} \left[\frac{p(d=1|x)}{p(d=0|x)} L(f_\theta(X), Y) \right].$$

The above equality holds assuming no concept shift and $p(d=1) = p(d=0)$. We estimate the ratio $\frac{p(d=1|x)}{p(d=0|x)}$ by training a separate neural network with the same architecture as f_ϕ^y to predict the domain of a given input sample.

CORAL [15] We use the same network architecture as f_ϕ^y for its classifier network. The CORAL-loss is computed on the output of the last ConvBlock (Layer #5 in Table 5.3b). As in the original work, during training we schedule the weight of the CORAL-loss to gradually increase from 0 to 1 using the schedule proposed in [31]:

$$\lambda_t = \frac{2}{1 + \exp(-\alpha t)} - 1, \tag{5.1}$$

where λ_t is the weight, t is the training progress linearly scaled to $[0, 1]$, and α is a tuning parameter.

DANN [31] Again, the same network architecture as f_ϕ^y is used for the main classifier network, where we choose to focus on performing domain alignment on the output feature map of the last ConvBlock. DANN involves an additional *domain classifier* component that attempts to classify the domain of this feature map. For this, a three-layers neural network that has the architecture of the last three layers (Layer #6-8 in Table 5.3b) of f_ϕ^y is used, similar to that in the original work. For the weight on the domain adversarial loss, (5.1) is used to gradually increase it from 0 to 1 during training.

Chapter 6

Conclusion and Future Work

In this thesis, we explored an alternative generative view to the unsupervised domain adaptation (UDA) problem and proposed a novel solution based on latent variable models.

The UDA problem has represented one fundamental limitation in machine learning with neural networks, and has limited the applicability of various neural network techniques to a much boarder scope. Although many previous works were proposed to tackle the UDA problem, there still existed a large performance gap between models adapted using the various UDA approaches compared to that trained natively on labeled-data from all domains.

Within the literature, the use of generative modeling to solve the UDA problem had not yet been widely explored, and this thesis aims to fill in this gap. In this thesis, we proposed to treat the UDA problem as a generative modeling problem, and formulated a novel framework to solve it based on probabilistic modeling and variational inference. Under such framework, we had motivated our assumption on the data generating process for the UDA problem, and demonstrated its direct incorporation into a generative model.

Empirical results validated the efficacy of our framework, showing superior performance compared to previous UDA methods on simpler scenarios, along with the capability of solving the domain translation problem. This demonstrated generative modeling for UDA as a promising direction. However, despite its successes, our framework has limitations, particularly in modeling highly complex variations across domains and its computational demands. With these in mind, we propose the below list as future work.

1. Improve Numerical Stability It was demonstrated that our approach suffers from numerical instability when deployed in UDA scenarios involving more complex domain

shifts. This likely resulted from the generative assumption that the distributions associated with the latent factors share the same intrinsic dimension between domains, causing some distributions to become ill-conditioned when one domain lies within a subspace of the other. Hence, one direct improvement to the existing framework is a generalization that supports different dimension of the factors' distribution across domains; while another is a robust method for automatic selection of the intrinsic dimension of the factors' distribution that avoids singularities.

2. Reduce Computational Cost Our current approach had shown a computation cost that scales approximately linearly with the size of the label set. This undesirable property makes it unrealistic to apply to tasks with a large number of possible labels. Fortunately, recent advances in generative modeling had proposed an efficient sample estimator to expectations taken with respect to a categorical distribution [54]. It is worth investigating the validity of our framework when the expectation over the labels is replaced with such sample estimate, in which case the computational cost is decoupled from the label set size.

3. Extend to Regression Problems We had focused on solving the UDA problem under the setting of classification in this work. Aside from classification, another common supervised learning setting is regression, where the label set becomes the uncountable set of real numbers. One immediate avenue of research is to extend the existing framework towards solving regression tasks and investigate its effectiveness using some regression datasets.

4. Semi-supervised Domain Adaptation Our approach can be directly extended to tackle the problem of semi-supervised domain adaptation where a few labeled target examples are available along with or without an abundance of unlabeled target examples. It is worth investigating the sample efficiency in terms of the target examples of our approach under both of these settings, and compare against other mainstream approaches designed for these settings.

References

- [1] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser Nam Lim, and Rama Chellappa. Learning from synthetic data: Addressing domain shift for semantic segmentation. In IEEE Conference on Computer Vision and Pattern Recognition, pages 3752–3761, 2018. [2](#)
- [2] Hao Guan and Mingxia Liu. Domain adaptation for medical image analysis: A survey. IEEE Transactions on Biomedical Engineering, pages 1173–1185, 2021. [3](#)
- [3] Christian S Perone, Pedro Ballester, Rodrigo C Barros, and Julien Cohen-Adad. Unsupervised domain adaptation for medical imaging segmentation with self-ensembling. NeuroImage, pages 1–11, 2019. [3](#)
- [4] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. Learning to adapt structured output space for semantic segmentation. In IEEE Conference on Computer Vision and Pattern Recognition, pages 7472–7481, 2018. [3](#)
- [5] Tuan-Hung Vu, Himalaya Jain, Maxime Bucher, Matthieu Cord, and Patrick Pérez. Advent: Adversarial entropy minimization for domain adaptation in semantic segmentation. In IEEE Conference on Computer Vision and Pattern Recognition, pages 2517–2526, 2019. [3](#)
- [6] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In IEEE Conference on Computer Vision and Pattern Recognition, pages 3723–3732, 2018. [3](#), [4](#), [5](#)
- [7] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. Using simulation and domain adaptation to improve

- efficiency of deep robotic grasping. In IEEE International Conference on Robotics and Automation, pages 4243–4250, 2018. [3](#), [4](#), [6](#)
- [8] Yong Dai, Jian Liu, Xiancong Ren, and Zenglin Xu. Adversarial training based multi-source unsupervised domain adaptation for sentiment analysis. In AAAI Conference on Artificial Intelligence, pages 7618–7625, 2020. [3](#)
- [9] Minhø Ryu, Geonseok Lee, and Kichun Lee. Knowledge distillation for bert unsupervised domain adaptation. Knowledge and Information Systems, pages 3113–3128, 2022. [3](#)
- [10] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. Journal of Machine Learning Research, 2007. [4](#)
- [11] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In International Conference on Machine Learning, page 114, 2004. [4](#), [49](#), [56](#)
- [12] Jiayuan Huang, Arthur Gretton, Karsten Borgwardt, Bernhard Schölkopf, and Alex Smola. Correcting sample selection bias by unlabeled data. Advances in Neural Information Processing Systems, 2006. [4](#)
- [13] Takafumi Kanamori, Shohei Hido, and Masashi Sugiyama. A least-squares approach to direct importance estimation. Journal of Machine Learning Research, pages 1391–1445, 2009. [4](#)
- [14] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. arXiv preprint arXiv:1412.3474, 2014. [4](#), [5](#)
- [15] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In European Conference on Computer Vision Workshops, pages 443–450, 2016. [4](#), [5](#), [49](#), [57](#)
- [16] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In International Conference on Machine Learning, pages 97–105, 2015. [4](#), [5](#)
- [17] Yongchun Zhu, Fuzhen Zhuang, Jindong Wang, Guolin Ke, Jingwu Chen, Jiang Bian, Hui Xiong, and Qing He. Deep subdomain adaptation network for image classification. IEEE Transactions on Neural Networks and Learning Systems, pages 1713–1722, 2020. [4](#), [5](#)

- [18] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-adversarial training of neural networks. Journal of Machine Learning Research, pages 1–35, 2016. 4, 5
- [19] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In IEEE International Conference on Computer Vision, pages 2223–2232, 2017. 4, 6
- [20] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In International Conference on Machine Learning, pages 1989–1998, 2018. 4, 6
- [21] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. Advances in Neural Information Processing Systems, 2017. 4, 6, 27
- [22] Hsin-Ying Lee, Hung-Yu Tseng, Jia-Bin Huang, Maneesh Singh, and Ming-Hsuan Yang. Diverse image-to-image translation via disentangled representations. In European Conference on Computer Vision, pages 35–51, 2018. 4, 6
- [23] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. Deep reconstruction-classification networks for unsupervised domain adaptation. In European Conference on Computer Vision, pages 597–613, 2016. 4, 6, 7, 27
- [24] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. Advances in Neural Information Processing Systems, 2016. 4, 6, 7, 26, 27
- [25] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. Advances in Neural Information Processing Systems, 2016. 4, 6
- [26] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, pages 559–572, 1901. 4
- [27] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. Advances in Neural Information Processing Systems, 2006. 5, 26

- [28] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. Machine Learning, pages 151–175, 2010. [5](#), [26](#)
- [29] Dino Sejdinovic, Bharath Sriperumbudur, Arthur Gretton, and Kenji Fukumizu. Equivalence of distance-based and rkhs-based statistics in hypothesis testing. The Annals of Statistics, pages 2263–2291, 2013. [5](#)
- [30] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. Advances in Neural Information Processing Systems, 2014. [5](#)
- [31] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In International Conference on Machine Learning, pages 1180–1189, 2015. [5](#), [50](#), [57](#)
- [32] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013. [6](#), [37](#), [42](#)
- [33] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, pages 504–507, 2006. [6](#)
- [34] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In International Conference on Machine Learning, pages 1096–1103, 2008. [6](#)
- [35] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair autoencoder. arXiv preprint arXiv:1511.00830, 2015. [7](#)
- [36] Maximilian Ilse, Jakub M Tomczak, Christos Louizos, and Max Welling. Diva: Domain invariant variational autoencoders. In Medical Imaging with Deep Learning, pages 322–348. PMLR, 2020. [7](#)
- [37] Mark Davenport. A first model of learning, 2024. [19](#)
- [38] Jonathan J. Hull. A database for handwritten text recognition research. IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 550–554, 1994. [21](#), [47](#)
- [39] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, pages 2278–2324, 1998. [21](#)

- [40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 2012. 22
- [41] Julia A Lasserre, Christopher M Bishop, and Thomas P Minka. Principled hybrids of generative and discriminative models. In 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), volume 1, pages 87–94. IEEE, 2006. 26
- [42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011. 29
- [43] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. arXiv preprint arXiv:1907.02893, 2019. 32
- [44] MTCAJ Thomas and A Thomas Joy. Elements of Information Theory. Wiley-Interscience, 2006. 32
- [45] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. Advances in Neural Information Processing Systems, 2014. 37, 39
- [46] John Duchi. Derivations for linear algebra and optimization. Berkeley, California, pages 2325–5870, 2007. 43
- [47] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015. 43, 44
- [48] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. arXiv preprint arXiv:1603.04779, 2016. 43
- [49] Li Deng. The mnist database of handwritten digit images for machine learning research. IEEE signal processing magazine, pages 141–142, 2012. 47
- [50] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In Neural Information Processing Systems Workshop, page 4, 2011. 47

- [51] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. Neural Information Processing Systems Workshop, 2017. [49](#)
- [52] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. Journal of Machine Learning Research, 2019. [49](#)
- [53] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2017. [55](#)
- [54] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In International Conference on Learning Representations, 2017. [59](#)
- [55] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: Transfer learning from unlabeled data. In Proceedings of the 24th international conference on Machine learning, pages 759–766, 2007.
- [56] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. A kernel method for the two-sample problem. Advances in Neural Information Processing Systems, 2006.
- [57] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. Advances in Neural Information Processing Systems, 2007.
- [58] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In IEEE Conference on Computer Vision and Pattern Recognition, pages 7167–7176, 2017.
- [59] Ryuhei Takahashi, Atsushi Hashimoto, Motoharu Sonogashira, and Masaaki Iiyama. Partially-shared variational auto-encoders for unsupervised domain adaptation with target shift. In European Conference on Computer Vision, pages 1–17, 2020.
- [60] Zongyao Li, Ren Togo, Takahiro Ogawa, and Miki Haseyama. Variational autoencoder based unsupervised domain adaptation for semantic segmentation. In IEEE International Conference on Image Processing, pages 2426–2430, 2020.

- [61] Robert M Gray and Lee D Davisson. An Introduction to Statistical Signal Processing. Cambridge University Press, 2004.
- [62] Difei Gao, Ke Li, Ruiping Wang, Shiguang Shan, and Xilin Chen. Multi-modal graph neural network for joint reasoning on vision and scene text. In IEEE Conference on Computer Vision and Pattern Recognition, pages 12746–12756, 2020.
- [63] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In International Conference on Machine Learning, pages 8748–8763, 2021.

APPENDICES

Appendix A

Derivation of Variational Lower-Bounds

Here we give the derivations to the variational lower-bound in (4.8).

$$\begin{aligned}\log p_\theta(x, y|d) &= \log \mathbb{E}_{H, Z \sim p_\theta(h, z|y, d)} [p_\theta(x|H, Z)p_\theta(y)] \\ &= \log \int_{\mathbb{H}, \mathbb{Z}} p_\theta(x, h, z, y|d) \frac{q_\phi(h, z|x, y, d)}{q_\phi(h, z|x, y, d)} dh dz \\ &= \log \mathbb{E}_{H, Z \sim q_\phi(h, z|x, y, d)} \left[\frac{p_\theta(x, H, Z, y|d)}{q_\phi(H, Z|x, y, d)} \right] \\ &\geq \mathbb{E}_{H, Z \sim q_\phi(h, z|x, y, d)} \left[\log \frac{p_\theta(x, H, Z, y|d)}{q_\phi(H, Z|x, y, d)} \right] \quad \because \text{Jensen's Inequality} \\ &= \mathbb{E}_{H, Z \sim q_\phi(h, z|x, y, d)} [\log p_\theta(x|H, Z)] - D_{\text{KL}}[q_\phi(h|x, y) \| p_\theta(h|y)] \\ &\quad - D_{\text{KL}}[q_\phi(z|x, d) \| p_\theta(z|d)] + \log p_\theta(y).\end{aligned}$$

$$\begin{aligned}
\log p_\theta(x|d) &= \log \mathbb{E}_{H,Z \sim p_\theta(h,z|d)} [p_\theta(X|H, Z)] \\
&= \log \sum_y \int_{\mathbb{H}, \mathbb{Z}} p_\theta(x, h, z, y|d) \frac{q_\phi(h, z, y|x, d)}{q_\phi(h, z, y|x, d)} dh dz \\
&= \log \mathbb{E}_{H,Z,Y \sim q_\phi(h,z,y|x,d)} \left[\frac{p_\theta(x, H, Z, Y|d)}{q_\phi(H, Z, Y|x, d)} \right] \\
&\geq \mathbb{E}_{H,Z,Y \sim q_\phi(h,z,y|x,d)} \left[\log \frac{p_\theta(x, H, Z, Y|d)}{q_\phi(H, Z, Y|x, d)} \right] \quad \because \text{Jensen's Inequality} \\
&= \sum_y q_\phi(y|x) \left\{ \mathbb{E}_{H,Z \sim q_\phi(h,z|x,y,d)} [\log p_\theta(x|H, Z)] - D_{\text{KL}}[q_\phi(h|x, y) \| p_\theta(h|y)] \right\} \\
&\quad - D_{\text{KL}}[q_\phi(z|x, d) \| p_\theta(z|d)] - D_{\text{KL}}[q_\phi(y|x) \| p_\theta(y)].
\end{aligned}$$

Appendix B

List of Tuning Parameters

Here we give the ranges of tuning parameters used in our experiment for our approach, shown in Table B.1.

Tuning Param.	List of Values
(γ_s, γ_t)	(0.5, 0.5), (0.9, 0.1), (0.1, 0.9)
NN/priors update steps	10, 100, 1000
learning rate (ADAM)	2.0e-4, 2.0e-3, 2.0e-2
betas (ADAM)	(0.5, 0.999), (0.9, 0.999)
batch size	16, 32, 64
gradient clipping	0.1, 1.0, 10

Table B.1: Ranges of tuning parameters used in our experiment for our approach.