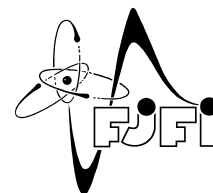




CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Nuclear Sciences and Physical
Engineering



Hybrid Discriminative-Generative Training for Set data

Hybridní diskriminativně-generativní modely pro množinová data

Master's Thesis

Author: **Bc. Jakub Bureš**
Supervisor: **doc. Ing. Václav Šmídl, Ph.D.**
Academic year: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Jakub Bureš
Studijní program: Aplikované matematicko-stochastické metody
Název práce (česky): Hybridní diskriminativně-generativní modely pro množinová data
Název práce (anglicky): Hybrid Discriminative-Generative Training for Set Data

Pokyny pro vypracování:

- 1) Seznamte se s problémem diskriminativního učení množinových dat, tj. kde jeden záznam je tvořen množinou vektorů. Na vybraných datech ukažte základní vlastnosti přístupu, například problém s přetrénováním.
- 2) Seznamte se s metodami generativního učení vektorů. Zaměřte se na techniku variační autoencoder. Možnosti metody generovat vzorky ukažte na jednoduchých datech.
- 3) Seznamte se s metodikou kombinovaného generativního a diskriminativního učení pomocí kontrastního učení, která nabízí kombinaci obou přístupů. Na jednoduchých datech ověřte funkčnost metody pro klasifikační úlohy s množinovými daty.
- 4) Navrhněte metodu nahrazující generativní model na bázi kontrastivního učení modelem založeným na variačním autoencoderu. Srovnajte jeho vlastnosti se základním modelem.
- 5) Všechny experimenty provádějte na standardních datech pro množinové úlohy z databáze UCI.

Doporučená literatura:

- 1) H. Liu, P. Abbeel, Hybrid discriminative-generative training via contrastive learning. arXiv preprint arXiv:2007.09070, 2020.
- 2) T. Pevny and P. Somol, Discriminative models for multi-instance problems with tree structure. In Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security, 2016, 83-91.
- 3) K. Jinwoo, J. Yoo, J. Lee and S. Hong. SetVAE: Learning Hierarchical Composition for Generative Modeling of Set-Structured Data. In 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', IEEE, 2021, 15059-15068.

Jméno a pracoviště vedoucího diplomové práce:

doc. Ing. Václav Šmídl Ph.D.

Ústav teorie informace a automatizace, Pod Vodárenskou věží 4, 182 00, Praha 8

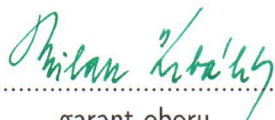
Jméno a pracoviště konzultanta:

Datum zadání diplomové práce: 31.10.2021

Datum odevzdání diplomové práce: 2.5.2022

Doba platnosti zadání je dva roky od data zadání.

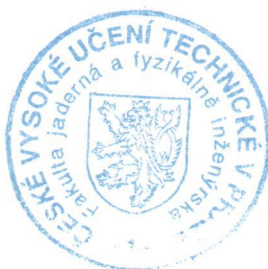
V Praze dne 01.11.2021



garant oboru



vedoucí katedry




děkan

Acknowledgment:

I would like to thank doc. Ing. Václav Šmídl, Ph.D. for his expert guidance and patience during this academic year. In addition, I would like to thank my family for all the support I have received on my academic journey.

Author's declaration:

I declare that this master's thesis is entirely my own work and I have listed all the used sources in the bibliography.

Prague, May 2, 2022

Bc. Jakub Bureš

A handwritten signature in black ink, consisting of a stylized 'B' followed by a horizontal line and a small flourish.

Název práce:

Hybridní diskriminativně-generativní modely pro množinová data

Autor: Bc. Jakub Bureš

Obor: Matematické inženýrství

Zaměření: Aplikované matematicko-stochastické metody

Druh práce: Diplomová práce

Vedoucí práce: doc. Ing. Václav Šmídl, Ph.D.

Abstrakt: Tato diplomová práce se zabývá hybridními diskriminativními a generativními modely a jejich možným využitím v multi–instančním učení, kde je jeden vzorek tvořen množinou vektorů. Doposud byly tyto modely trénovány pouze diskriminativně. Ukazuje se však, že samotný diskriminativní přístup může mít svoje limity a přidáním generativní složky je možno tyto limity posunovat. Způsobů, jak zahrnout generativní složku do tréninku modelu, je vícero. My se zaměříme na dva, jeden pomocí kontrastivního učení a druhý na bázi variačního autoencoderu. Pilířem práce s množinovými daty je pak struktura HMill s knihovnou Mill.jl implementována v programovacím jazyku Julia, která dovoluje jednoduše a efektivně trénovat modely na těchto datech. My se ji pokusíme rozšířit o generativní složku.

Klíčová slova: diskriminativní, generativní, modelování, multi–instanční učení, množinová data, variační autoencoder

Title:

Hybrid Discriminative-Generative Training for Set data

Author: Bc. Jakub Bureš

Abstract: This master's thesis deals with hybrid discriminative and generative models and their possible use in multi–instance learning, where one sample consists of a set of vectors. So far, these models have only been trained discriminatively. However, it turns out that the discriminative approach alone can have downsides and, by adding a generative component, these downsides can be minimized. There are several ways to include a generative component in model training. We focus on two, one using contrastive learning and the other based on a variational autoencoder. The mainstay of working with set data is then the HMill framework with the Mill.jl library, implemented in the Julia programming language, which allows us to train models on these data simply and efficiently. We will try to extend it with a generative component.

Key words: discriminative, generative, modeling, multiple instance learning, set data, variational autoencoder

Contents

Introduction	17
1 Theoretical Introduction	19
1.1 Mathematical notation	19
1.2 Probability theory	20
1.3 Supervised learning	20
1.3.1 Prediction	21
1.4 Unsupervised learning	24
1.5 Bayesian inference	24
1.5.1 Choice of prior distribution	25
1.5.2 Prediction	26
2 Discriminative vs. Generative Models	27
2.1 Overview	27
2.2 Discriminative modeling	27
2.2.1 Connection to Kullback–Leibler divergence	28
2.2.2 One-hot encoding	30
2.3 Generative modeling	30
2.3.1 Variational autoencoder	30
2.3.2 Semi-supervised variational autoencoder	36
2.3.3 Noise-contrastive estimation	38
3 Hybrid Generative and Discriminative Models	43
3.1 Energy-based models	43
3.1.1 Joint energy models	44
3.2 Contrastive learning	45
3.3 Hybrid discriminative and generative models	45
3.3.1 Toy problem - polynomial regression	47
3.3.2 Experiment setup and results	49
3.4 Hybrid VAE	52
3.4.1 Toy problem	52

4 Multiple Instance Learning	55
4.1 Fundamentals	55
4.2 Embedded-space paradigm	56
4.3 Training	57
4.3.1 Cross-validation	57
4.4 Application of HDGM to the MIL problems	59
4.4.1 Setup and results	59
4.4.2 Discussion	62
4.5 Hybrid VAE for MIL	63
4.5.1 Setup and results	63
4.5.2 Discussion	64
Conclusion	67
Bibliography	69
A Computational formulas	73
A.1 Solution of $D_{\text{KL}}(\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \sigma^2 \mathbb{I}_p) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbb{I}_p))$	73
A.2 Derivation of $\mathbb{E}_{p_{\theta}(\mathbf{x})}[-\nabla_{\theta} E_{\theta}(\mathbf{x})]$	75

List of Figures

1.1	Splitting the data into $K = 5$ roughly equal-sized parts.	22
1.2	Illustrative sketch of prediction error as a function of model complexity.	23
2.1	Discriminative approach.	28
2.2	Generative approach.	28
2.3	Reparametrization trick.	33
2.4	VAE diagram, where the input x is passed through VAE architecture to create reconstructed input \hat{x}	34
2.5	True and estimated samples using VAE.	36
2.6	Results of the NCE experiment for one-dimensional and two-dimensional Gaussian case.	40
3.1	Sensitivity of the polynomial model to the order of the polynomial $s - 1$, specifically $s - 1 \in \{2, 4\}$	50
3.2	Sensitivity of the polynomial model to the order of the polynomial $s - 1$, specifically $s - 1 \in \{5, 6\}$	50
3.3	Sensitivity of the polynomial model to the order of the polynomial $s - 1$, specifically $s - 1 \in \{8, 10\}$	50
3.4	Sensitivity of the polynomial model to the parameter τ , specifically $\tau \in \{0.1, 1\}$	51
3.5	Sensitivity of the polynomial model to the parameter τ , specifically $\tau \in \{10, 100\}$	51
3.6	Sensitivity of the polynomial model to the parameter τ , specifically $\tau \in \{10^3, 10^4\}$	51
3.7	AUC-ROC for both SM and HM.	53
3.8	Comparison of the generated samples of the SM and HM models. The train data are identical in both cases.	53
4.1	The difference between standard ML and MIL. Standard ML is special case of MIL with $ b = 1$	56
4.2	MIL diagram, where b is the input bag, f_θ is the classifier applied on the resulting embedded space representation $\Lambda_\Omega(b)$ of the bag.	57
4.3	Evaluation of prediction error with the use of training data and testing data on MIL data sets Musk1, Musk2, Fox, and Tiger.	58
4.4	Dependence of the prediction error $\text{err}(\alpha)$ on the hyper-parameter α	60
4.5	Comparison of prediction errors for HDGM and standard discriminative training.	62

4.6 Comparison of the dependencies of the average AUC on $\log \nu$ with st. deviation for hybrid VAE.	65
--	----

List of Tables

1.1	Confusion matrix.	23
2.1	Transformation of a label encoding (left) to the one-hot encoding (right).	30
4.1	Results of CV evaluated on the testing data.	59
4.2	Prediction error summary for the HDGM loss in case of $\xi = 10$	61
4.3	Comparison of prediction errors for HDGM $\alpha = 0.5$ and discriminative part only. Pay special attention to the last column $\text{err}(\xi = 10)$ in each approach.	61
4.4	Average AUCs and st. deviations of individual data sets for both approaches with $r = 50$	62
4.5	Average AUCs and st. deviations of individual data sets for hybrid VAE over $r = 10$ runs.	64

List of Acronyms and Abbreviations

CE	Cross-Entropy
CV	Cross-Validation
EBM	Energy-Based Model
ELBO	Evidence Lower Bound
FP	False Positive
FN	False Negative
FPR	False Positive Rate
HDGM	Hybrid Discriminative Generative energy-based Model
i.i.d.	Independent and Identically Distributed
JEM	Joint Energy-based Model
KL	Kullback-Leibler (divergence)
MCMC	Markov Chain Monte Carlo
MIL	Multiple Instance Learning
ML	Machine Learning
MLE	Maximum Likelihood Estimation
NN	Neural Network
PDF	Probability Density Function
PR	Precision-Recall (curve)
ROC	Receiver Operator Characteristic (curve)
SL	Supervised Learning
SSVAE	Semi-Supervised Variational AutoEncoder

TP	True Positive
TPR	True Postive Rate
TN	True Negative
UL	Unsupervised Learning
VAE	Variational AutoEncoder

Introduction

In the field of supervised learning, tremendous progress and success have been achieved in recent years. Examples of such successes include speech recognition [1], protein structure prediction [2, 3] or anomaly detection [4, 5, 6, 7]. Anomaly detection, also known as outlier analysis, attempts to identify data points, events, and/or observations that deviate from the normal behavior of a data set. At the level of Internet security, anomaly detection often operates with set data that are processed using multiple instance learning [8, 9, 10, 11]. So far, the parameters of these models have only been trained discriminatively.

Multiple instance learning tasks are of the type that we call classification. Such tasks are typically addressed by minimizing a cross-entropy loss, which is defined as an expected value of logarithm of the Softmax function. It is actually a generalization of the logistic function into multiple dimensions.

It turns out that cross-entropy has its limits and can be extended on the basis of contrastive learning, which is used in generative modeling. Contrastive learning [12, 13] is a machine learning method that is often used in representation learning for image classification or video understanding. For training such models is, most of the time, minimized the contrastive loss, which reduces the distance between representations of different augmented views of the same image and increases the distance between representations of augmented views of different images.

In this master's thesis, these two objectives, that is, cross-entropy and contrastive loss, are brought together and used in the form of a hybrid combination [14] and thus the model parameters can be trained both discriminatively and generatively. This study provides important insight into possible cooperation of these two approaches. However, contrastive learning is not the only generative approach that we use to improve discriminative modeling. An interesting way to combine generative and discriminative modeling is also offered by the semi-supervised variational autoencoder [15, 16, 17]. This is an extension of the standard variational autoencoder [18, 19] which is nothing more than an artificial neural network that compresses the input into the latent space using an encoder. The decoder then receives as input the information sampled from the latent space and produces an estimate of the input.

All experiments are performed in the Julia programming language [20] for several reasons. For the first, Julia is considered by many to be a rising star among all programming languages. It is a fast, dynamically typed language suitable for computational science. More essentially, the HMill unified framework and the Mill.jl library [21, 22] implemented in Julia provide an easy and effective way to work with multiple instance learning problems. The corresponding scripts are listed in <https://github.com/KubaBury/hybridVAE.jl>.

This work is organized into four chapters in a logical sequence. The first chapter contains a theoretical introduction, which is necessary to better understand the whole work. The second chapter consists of discriminative and generative modeling [23, 24, 25]. The theoretical aspects of these two approaches are described in detail. Part of the generative models is also the already mentioned variational autoencoder and its extended semi-supervised version. This is followed by Chapter 3, where a method is presented to define hybrid learning using energy-based models [26, 27, 28, 29]. In this chapter, a brief introduction to contrastive learning is provided. The last chapter covers multiple instance learning, i.e., how to work with multiple data and its training method. Many experiments are carried out here, which point out some shortcomings of the discriminative approach itself and include its improvement using a hybrid approach.

The primary goal of this thesis is to test the hybrid approach in real set data and ultimately to show its benefits or downsides compared to discriminative learning.

1

Theoretical Introduction

It is customary that vast academic paper, for better understanding, in its beginning outlines miscellaneous theoretical aspects, which are used during the other chapters. This thesis is no exception.

1.1 Mathematical notation

It will be most appropriate to begin by introducing the basic notation that will be used throughout this thesis. This will ensure that any confusion will be avoided, even though the notation is quite standard.

Notation for random variables using upper case letters of the Latin alphabet is widely used. Typically, the letters used are from the end of the alphabet, i.e. X, Y or Z . The realization of a random variable, also known as an observed value or simply an observation, will be denoted by the appropriate lower case letters. Thus, the realization $x \in \mathbb{R}$ corresponds to the random variable X , which holds by analogy for other random variables. However, significant simplification can be achieved if one uses the same notation for random variables and realizations.

Bold symbols, for instance, $\mathbf{x} \in \mathbb{R}^D$ or $\mathbf{y} \in \mathbb{R}^D$, will be used to distinguish vectors and scalars. All vectors are assumed to be column vectors, so $\mathbf{x} = (x_1, x_2, \dots, x_D)^\top$ and hence \mathbf{x}^\top is a row vector.

Any matrices will be denoted by blackboard bold Latin letters, for example, if one has N values of D -dimensional vector of observations $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, it can be simply combined into a $D \times N$ data matrix \mathbb{X} in which the j^{th} row of \mathbb{X} corresponds to the row vector \mathbf{x}_j^\top . The symbol \mathbb{I}_N denotes the square $N \times N$ identity matrix, i.e., matrix with ones on the main diagonal and zeros elsewhere. The set of observations will be denoted by bold uppercase letter, for example $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$.

1.2 Probability theory

Mathematical models are very well described by probability and for this reason, this section will look at some of the basic concepts of probability theory that we will need. The most important such concept is probability density function (PDF). The symbol $p(x)$ will be used predominantly for the PDF, which is a function of x . In addition, this will be used for both discrete and continuous x . In this way can be achieved significant simplification and unification of all formulas and equations. Any PDF is a non-negative function and its integration over the entire space is equal to 1. This applies to multivariate case as well as it applies to univariate case, therefore integration of joint PDFs $p(\mathbf{x}) = p(x_1, x_2, \dots, x_D)$ over the entire space is also equal to 1. In mathematical terms one can express it as follows

$$\int_{\mathbb{R}^D} p(\mathbf{x}) d\mathbf{x} = 1. \quad (1.1)$$

In other parts of this thesis we will use conditional PDFs such as $p_{\boldsymbol{\theta}}(\mathbf{x}) \equiv p(\mathbf{x}|\boldsymbol{\theta})$ that are conditioned by known parameters $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^s$, where Θ is called parameter space. The constraint (1.1) can be always fulfilled by redefining the PDF as

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}^0(\mathbf{x})}{Z(\boldsymbol{\theta})}, \quad Z(\boldsymbol{\theta}) = \int_{\mathbb{R}^D} p_{\boldsymbol{\theta}}^0(\mathbf{x}) d\mathbf{x}, \quad (1.2)$$

where $p_{\boldsymbol{\theta}}^0(\mathbf{x})$ specifies the functional form of the $p_{\boldsymbol{\theta}}(\mathbf{x})$ and does not need to integrate to 1. The normalization constant $Z(\boldsymbol{\theta})$ is often called the partition function.

The average value of some function $g(x)$ under a probability distribution $p(x)$ is typically denoted by $\mathbb{E}[g(x)]$ and it is called expected value or mean [30]. For a continuous variable, expected value are expressed in terms of an integration with respect to the corresponding probability density

$$\mathbb{E}[g(x)] = \int_{\mathbb{R}} p(x)g(x)dx.$$

In the case of a discrete variable, one has to keep in mind that an integration turns into a sum over all x . To specify over which PDF the expectation is calculated, the notation $\mathbb{E}_{p(x)}[g(x)]$ can be used.

1.3 Supervised learning

Supervised learning (SL), in less academic terms called "learning with a teacher", is one of the machine learning tasks [24]. The goal of this approach is to make a good prediction of the output y (sometimes also called target variable), denoted by the symbol \hat{y} , with given input \mathbf{x} . This prediction is obtained through learning a model $f_{\boldsymbol{\theta}}(\mathbf{x}) \equiv f(\mathbf{x};\boldsymbol{\theta})$ that minimizes a loss function $\mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{x}), y)$ (also known as the error function), where $\boldsymbol{\theta} \in \Theta$ are the parameters of the model.

To construct this prediction one needs data, hence it is supposed that we have available set of independent and identically distributed (i.i.d.) observations, input–output paired

samples denoted by $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, eventually, this may in fact be

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N, \quad \mathbf{x}_i \in \mathbb{R}^D, \quad y_i \in \mathbb{R}, \quad \forall i = 1, \dots, N. \quad (1.3)$$

The index i will be omitted whenever it is clear that we are referring to terms associated with a single data point. Such setting is usually known as training data and its applications are regression problems. As an example, we can mention a couple of typically used loss functions for such problems. They are squared error and absolute error

$$\mathcal{L}(\hat{y}, y) = \begin{cases} (y - \hat{y})^2 \\ |y - \hat{y}| \end{cases}$$

where $\hat{y} = \hat{f}_{\boldsymbol{\theta}}(\mathbf{x}) = f(\mathbf{x}; \hat{\boldsymbol{\theta}})$. As we are not quite interested in regression problems in this thesis, we will mainly deal with the second approach. That is classification problems, i.e. when $y \in \mathcal{C}$ is qualitative output and where \mathcal{C} is a finite set. A typical example is binary classification, where $\mathcal{C} = \{-1, +1\}$. However, classification will be object of interest later in Section 2.

Here it is clear why the term "learning with a teacher" is used. This metaphor means that the student presents output \hat{y} and the teacher provides either a correct answer and/or an error that corresponds to the student's answer.

1.3.1 Prediction

The generalization performance [34], i.e. the performance on out-of-sample data of the models learned by the algorithm relates to its prediction capability on independent test data \mathcal{T} . Assessment of this performance is essentially important in practice, since it conducts the choice of learning method or model, and provides a measure of the quality of the hereafter chosen model. In fact, there are two separate objectives that need to be achieved:

1. *Model selection* - estimating the performance of different models in order to choose the best one.
2. *Model assessment* - having chosen a final model, estimating its prediction error (generalization error) on new data.

Cross-validation

The simplest and most widely used method for estimating prediction error of the model $\hat{f}_{\boldsymbol{\theta}}$ is called *cross-validation* (CV). We review this method using the notation of the authors of [34]. It is used for direct estimating of the expected extra-sample error

$$\text{err} = \mathbb{E}[\mathcal{L}(y, \hat{y})],$$

the measure how accurately is the model able to predict output values for previously unseen data - independent test sample. In an ideal case, if sufficient number of data is available, a test set can be set aside and used to assess the performance of the employed prediction model. Since data are often scarce, this is usually not possible.

Very elegant solution to this problem is via *K-fold cross-validation*. It uses part of the available data for fitting the model, and a different part for testing. We split the data into K roughly equal-sized parts, for example, when $K = 5$, the scenario is shown in Figure 1.1. For the j^{th} part (third in Figure 1.1), we train the model to the other $K - 1$ parts of the data,

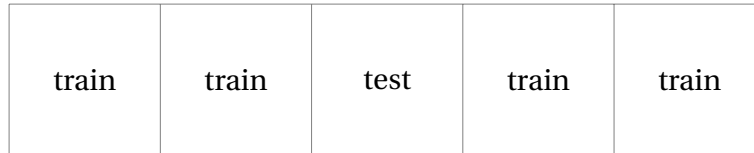


Figure 1.1: Splitting the data into $K = 5$ roughly equal-sized parts.

and calculate the prediction error of the fitted model when predicting the j^{th} part of the data. We repeat this process for $j \in \{1, 2, \dots, K\}$ and combine the K estimates of prediction error. Let $\gamma : \{1, \dots, N\} \rightarrow \{1, \dots, K\}$ be an indexing function that indicates the partition to which observation j is allocated by the randomization. Symbol $\hat{f}_{\theta}^{-j}(x)$ denotes the fitted model, computed with the j^{th} part of the data removed. Then the cross-validation estimate of prediction error is defined by

$$\text{CV}(\hat{f}_{\theta}) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y_i, \hat{f}_{\theta}^{-\gamma(i)}(x_i)). \quad (1.4)$$

Typical choices of K are 5 or 10 and even case $K = N$ that is known as *leave-one-out CV*. Generally, there is not an universal way of choosing K , since it strongly depends on the available number of data. The biggest problem of this method is a fact that it is computationally very expensive, because we usually train many models with different complexity and assess their performance. Let us now analyze the problem of the model complexity. Consider a polynomial regression problem, where the model is defined by

$$f_{\theta}(x) = \sum_{i=0}^{s-1} \theta_i x^i.$$

Here, over-fitting occurs very frequently. The complexity of the model of this case is very intuitive, as it is just the order of the polynomial, $s - 1$. Smaller orders of the polynomial (may) give rather poor fits to the data in contrast to a much higher order polynomial giving an excellent fit. However, such a polynomial passes exactly through each data point, oscillates wildly, and gives a poor prediction for the new input variable $x_0 \in \mathbb{R}$.

To obtain some quantitative insight into the dependence of the generalization performance on model complexity, consider a separate test set of data (testing data) used to assess the performance of the model. In general, the prediction error evaluated on the training data for increasing the complexity of the model approaches zero. On the other hand, the prediction error evaluated on the testing data for increasing model complexity is (from a certain point) increasing as well. The typical scenario is illustrated in Figure 1.2. The goal is then to choose a model that performs best on testing data. For extremely complicated and complex models that are trained for hours or days, is cross-validation inconvenient approach of estimating the prediction error as we need to train numerous models of this complexity.

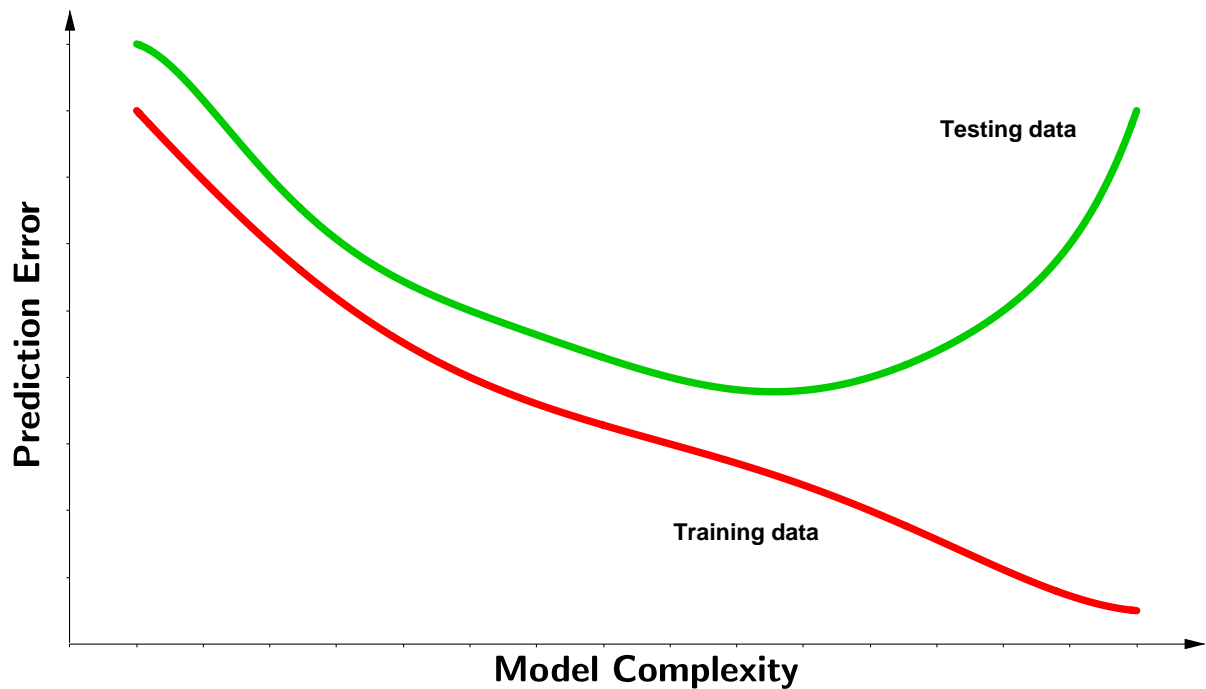


Figure 1.2: Illustrative sketch of prediction error as a function of model complexity.

ROC and PR curves

In binary classification problems, a classifier labels observations as either positive or negative. The decision made by the classifier can be represented via a confusion matrix¹, which has four categories [46]. True positives (TP) are observations correctly labeled as positives, on the other hand true negatives (TN) correspond to negative observations correctly labeled as negative. False positives (FP) refer to negative observations incorrectly labeled as positive and finally, false negatives (FN) refer to positive observations incorrectly labeled as negative. Following Table 1.1 illustrates a confusion matrix. Notice the great similarities with

	actual positive	actual negative
predicted positive	TP (#true positives)	FP (#false positives)
predicted negative	FN(#false negatives)	TN (#true negatives)

Table 1.1: Confusion matrix.

error types of hypothesis testing in statistics. Based on the confusion matrix, a point for either ROC or PR space is constructed and in addition a set of evaluation metrics is defined.

¹The confusion matrix is not only a matter of binary classification, but can also be extended to multi-class classification problem. For L classes, the confusion matrix takes the form of $L \times L$ matrix.

Among the best known and most frequently used are the following metrics

$$\begin{aligned}
 \text{Recall} &= \frac{TP}{TP + FN}, \\
 \text{Precision} &= \frac{TP}{TP + FP}, \\
 \text{True Positive Rate} &= \frac{TP}{TP + FN}, \\
 \text{False Positive Rate} &= \frac{FP}{FP + TN}.
 \end{aligned} \tag{1.5}$$

To construct Receiver Operator Characteristic (ROC) curve one needs to plot the True Positive Rate (TPR) against the False Positive Rate (FPR). The resulting curve depicts relative trade-offs between true positive and false positive. Classifiers that give curves closer to the top-left corner indicate a better performance. When dealing with data sets that are highly skewed, Precision–Recall (PR) curves give a more informative picture. Obviously, the PR curve is nothing more than plot of Precision against Recall. As opposition to the ROC, classifiers performs better for curves closer to the right–top corner. To get one number that describes the model performance, one can calculate area under either the ROC curve or PR curve, typically denoted by AUC. The symbols AUC–ROC or AUC–PR are used to distinguish the AUC association for individual curves. Generally, the higher the AUC score, the better a classifier performs.

1.4 Unsupervised learning

The previous section dealt with input–output paired samples \mathcal{D} . The second approach is a logical modification of SL, based on data without labels. Such setting is called unsupervised learning (UL) or "learning without a teacher". Unlike SL, one has a set of N observations in the form of $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ and nothing more. In this case, the student learns without any feedback from a supervisor or teacher providing correct answers. The goal is to directly infer the properties of $p(\mathbf{x})$.

1.5 Bayesian inference

The Bayesian methodology is a well established approach to statistical inference and became very important technique in statistics and data analysis. As its name suggests, Bayesian statistics is based on application of Bayes' rule. In this chapter, we briefly review basic concept of this approach, which was suggested here [31].

Let the measured data be denoted by \mathcal{D} , defined according to previous Section 1.1. A parametric probabilistic model of the data \mathcal{D} is given by the probability density function $p(\mathcal{D}|\boldsymbol{\theta})$, where again $\boldsymbol{\theta} \in \Theta$ denotes parameters of the model. The main idea behind Bayesian theory is the treatment of the unknown parameters $\boldsymbol{\theta}$ as a random variable. Bayes' rule is applied to infer model parameters $\boldsymbol{\theta}$, therefore

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\boldsymbol{\theta}, \mathcal{D})}{p(\mathcal{D})} = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\Theta} p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}}. \tag{1.6}$$

Since $p(\mathcal{D})$ is just the normalization constant, Equation (1.6) is often simplified to

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}). \quad (1.7)$$

Symbol \propto means equal up to the normalization constant. The term $p(\boldsymbol{\theta}|\mathcal{D})$ is known as the *posterior* distribution, $p(\mathcal{D}|\boldsymbol{\theta})$ as the *observation model*, and $p(\boldsymbol{\theta})$ is called the *prior* distribution of the $\boldsymbol{\theta}$. Note that evaluation of the normalization constant can be computationally expensive, in higher dimension even intractable.

There is of course many possible options how to obtain $\hat{\boldsymbol{\theta}}$ from posterior. Popular choices for an optimal value of the point estimate are:

1. Maximum A posteriori estimate (MAP)

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} p(\boldsymbol{\theta}|\mathcal{D}) \quad (1.8)$$

This method estimates $\boldsymbol{\theta}$ as the mode of the posterior distribution. It appears to be computationally attractive, as it is not necessary to evaluate the normalization constant.

2. Mean or expected value

$$\hat{\boldsymbol{\theta}}_{\text{B}} = \int_{\Theta} \boldsymbol{\theta} p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})} [\boldsymbol{\theta}] \quad (1.9)$$

Mean value, unlike MAP estimate, may be very expensive to compute because of the required integration. This may lead to further approximations such as EM algorithm [32].

1.5.1 Choice of prior distribution

For the posterior computation, it is necessary to specify the prior distribution $p(\boldsymbol{\theta})$, unfortunately, this might not be easily determined. This can be achieved through knowledge of previous models, expert knowledge, their combination, or even uncertainty about $\boldsymbol{\theta}$ being a viable option.

There are also many practical aspects of priors:

- *Regularization* - supplementing the data if there are scarce, insufficient data, or poorly defined models.
- *Restrictive conditions* - imposing various restrictions on the parameters $\boldsymbol{\theta}$ reflecting physical constraints. The choice of a prior distribution with bounded support will also result in a posterior distribution with bounded support.
- *Non-informative prior* - if the data are informative enough to make a prediction, it is proposed to choose a prior with minimal impact on the posterior distribution, such as uniform distribution. However, typical choices of non-informative priors are the so-called *Jeffreys priors* [33].

1.5.2 Prediction

We are usually not interested in the value of $\hat{\boldsymbol{\theta}}$ itself, but rather, once the model is estimated, we are interested in making a prediction of the output variable y_0 for the new input variable \mathbf{x}_0 . Note that the symbol \mathcal{D} contains all previously given data \mathbf{x} and y . The posterior predictive distribution is then determined by the distribution of y_0 , marginalized over the posterior

$$p(y_0|\mathbf{x}_0, \mathcal{D}) = \int_{\Theta} p(y_0|\mathbf{x}_0, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta}. \quad (1.10)$$

When the distribution $p(\boldsymbol{\theta}|\mathcal{D})$ is not available, we have to approximate leveraging the Dirac delta function $\delta(x)$ for which the property

$$\int_{\mathbb{R}} g(x) \delta(x - x_0) dx = g(x_0) \quad (1.11)$$

holds. Once property (1.11) is applied to Equation (1.10), we get

$$p(y_0|\mathbf{x}_0, \mathcal{D}) = \int_{\Theta} p(y_0|\mathbf{x}_0, \boldsymbol{\theta}) \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) d\boldsymbol{\theta} = p(y_0|\mathbf{x}_0, \hat{\boldsymbol{\theta}}),$$

causing an error. In typical MAP, this is known as *over-fitting*.

2

Discriminative vs. Generative Models

2.1 Overview

Machine learning models can be classified into two main categories, *discriminative* and *generative* models. Simply put, a discriminative model makes predictions based on conditional probability $p(y|\mathbf{x})$ and is used for classification or regression problems. In other words, discriminative models distinguish the decision boundary between the classes. It corresponds to learning parameters that maximize the conditional probability distribution $p(y|\mathbf{x})$. On the contrary, a generative model revolves around the distribution of a data set to return a probability for a given example. Rather than looking at classes and trying to find something to separate them, it focuses only on the one class at the time and builds a model what that certain class looks like, then turns attention to the other class. To express it more formally, generative models learn parameters that maximize $p(\mathbf{x}|y)$ and $p(y)$. Since

$$p(\mathbf{x}, y) = p(\mathbf{x}|y) \cdot p(y), \quad (2.1)$$

with joint PDF it is possible to generate new $\{\mathbf{x}', y'\}$ pairs. In some cases, the use of the second decomposition $p(\mathbf{x}, y) = p(y|\mathbf{x}) \cdot p(\mathbf{x})$ is also an option. Note that in an unsupervised setting, the task is reduced to inferring only $p(\mathbf{x})$. To provide more insight into this problem, simple illustrations are available in Figures 2.1 and 2.2.

2.2 Discriminative modeling

In this section, we review the basic concepts of discriminative modeling proposed in [14]. Given data \mathcal{D} according to (1.3), with the empirical distribution of \mathbf{x} being referenced by $\tilde{p}(\mathbf{x})$

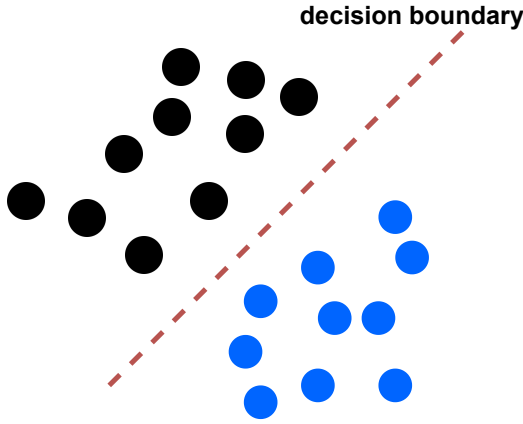


Figure 2.1: Discriminative approach.

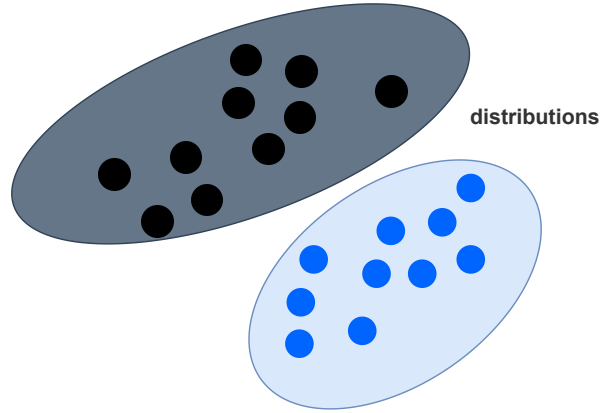


Figure 2.2: Generative approach.

and the empirical label distribution $\tilde{p}(y|\mathbf{x})$ containing L categories. In this thesis, we focus on classification problems, where the variable y is now a qualitative variable called a class label, taking on L possible values, and comes from a finite set \mathcal{C} . A classification problem is typically solved using a parametric function $f_{\theta} : \mathbb{R}^D \rightarrow \mathcal{C}$, where θ denotes the parameters of the model. In practice, the function f_{θ} is often used in the form of $\mathbb{R}^D \rightarrow \mathbb{R}^L$. This function maps each data point $\mathbf{x} \in \mathbb{R}^D$ to L real-valued numbers known as logits. It should be noted that \mathbb{R}^L is allowed here due to the utilization of *one-hot encoding*, which will be explained in Section 2.2.2. Logits are used to parameterize a categorical distribution through the transfer function

$$q_{\theta}(y|\mathbf{x}) = \frac{\exp(f_{\theta}(\mathbf{x})[y])}{\sum_{y \in \mathcal{C}} \exp(f_{\theta}(\mathbf{x})[y])}, \quad (2.2)$$

which is known as Softmax. In other words, the true data distribution $\tilde{p}(y|\mathbf{x})$ is modeled by a parameterized family of functions $\{q_{\theta}(y|\mathbf{x}) | \theta \in \Theta\}$ and thus $\tilde{p}(y|\mathbf{x})$ is assumed to belong to this family. Note that the convention $f_{\theta}(\mathbf{x})[y]$ means the y^{th} element of $f_{\theta}(\mathbf{x})$, that is, the logit corresponding to the y^{th} class label. For learning f_{θ} is usually minimized total cross-entropy (CE) loss

$$\text{CE}(\theta) = -\mathbb{E}_{\tilde{p}(y,\mathbf{x})} [\log q_{\theta}(y|\mathbf{x})] = -\mathbb{E}_{\tilde{p}(\mathbf{x})} \left[\mathbb{E}_{\tilde{p}(y|\mathbf{x})} [\log q_{\theta}(y|\mathbf{x})] \right] \approx -\frac{1}{N \cdot L} \sum_{i=1}^N \sum_{y \in \mathcal{C}} \log q_{\theta}(y|\mathbf{x}_i), \quad (2.3)$$

as it is relatively easy to compute and has several other justifications. These justifications will be addressed in the following text. Notice that the last approximation of (2.3) holds using the law of large numbers.

2.2.1 Connection to Kullback–Leibler divergence

The rationale for objective (2.3) comes from minimizing the Kullback-Leibler (KL) divergence with a target distribution $\tilde{p}(y|\mathbf{x})$ [35]. In general, the KL divergence (or KL distance)

from $\tilde{p}(y|\mathbf{x})$ to $q_{\boldsymbol{\theta}}(y|\mathbf{x})$ is defined as

$$D_{\text{KL}}(\tilde{p}(y|\mathbf{x})\|q_{\boldsymbol{\theta}}(y|\mathbf{x})) = \int \tilde{p}(y|\mathbf{x}) \log \frac{\tilde{p}(y|\mathbf{x})}{q_{\boldsymbol{\theta}}(y|\mathbf{x})} dy = \mathbb{E}_{\tilde{p}(y|\mathbf{x})} \left[\log \frac{\tilde{p}(y|\mathbf{x})}{q_{\boldsymbol{\theta}}(y|\mathbf{x})} \right] \quad (2.4)$$

and has the following properties:

1. $D_{\text{KL}}(\tilde{p}(y|\mathbf{x})\|q_{\boldsymbol{\theta}}(y|\mathbf{x})) \geq 0$,
2. $D_{\text{KL}}(\tilde{p}(y|\mathbf{x})\|q_{\boldsymbol{\theta}}(y|\mathbf{x})) = 0$ iff $\tilde{p}(y|\mathbf{x}) = q_{\boldsymbol{\theta}}(y|\mathbf{x})$ almost everywhere,
3. $D_{\text{KL}}(\tilde{p}(y|\mathbf{x})\|q_{\boldsymbol{\theta}}(y|\mathbf{x})) \neq D_{\text{KL}}(q_{\boldsymbol{\theta}}(y|\mathbf{x})\|\tilde{p}(y|\mathbf{x}))$ and KL divergence does not obey the triangle inequality.

The third property indicates that care is needed in the syntax describing KL divergence. We say that (2.4) is from $\tilde{p}(y|\mathbf{x})$ to $q_{\boldsymbol{\theta}}(y|\mathbf{x})$. Using the logarithmic property, (2.4) can be further rewritten in the form

$$\mathbb{E}_{\tilde{p}(y|\mathbf{x})} \left[\log \frac{\tilde{p}(y|\mathbf{x})}{q_{\boldsymbol{\theta}}(y|\mathbf{x})} \right] = \mathbb{E}_{\tilde{p}(y|\mathbf{x})} [\log \tilde{p}(y|\mathbf{x})] - \mathbb{E}_{\tilde{p}(y|\mathbf{x})} [\log q_{\boldsymbol{\theta}}(y|\mathbf{x})], \quad (2.5)$$

where the first term is called entropy, often denoted by $H(\tilde{p}(y|\mathbf{x}))$ and the second term is called CE. The subscript $\boldsymbol{\theta}$ emphasizes that $q_{\boldsymbol{\theta}}(y|\mathbf{x})$ is the approximate density we get to control. This gives us KL distance for single data point \mathbf{x} , but for optimization we need to include all data points [36]. Let

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N D_{\text{KL}}^{(i)}(\tilde{p}(y|\mathbf{x}_i)\|q_{\boldsymbol{\theta}}(y|\mathbf{x}_i)) \quad (2.6)$$

be the sum of KL distances over all data points. Since the entropy of $\tilde{p}(y|\mathbf{x}_i)$ does not depend on $\boldsymbol{\theta}$ therefore by minimizing (2.6) with respect to $\boldsymbol{\theta}$ we obtain

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} - \sum_{i=1}^N \mathbb{E}_{\tilde{p}(y|\mathbf{x}_i)} [\log q_{\boldsymbol{\theta}}(y|\mathbf{x}_i)] = \min_{\boldsymbol{\theta}} - \mathbb{E}_{\tilde{p}(y,\mathbf{x})} [\log q_{\boldsymbol{\theta}}(y|\mathbf{x})], \quad (2.7)$$

which corresponds to minimizing the objective CE($\boldsymbol{\theta}$) defined in Equation (2.3). This part deserves further discussion for a few reasons:

- Maximum likelihood estimation (MLE) of $\boldsymbol{\theta}$ is equivalent to minimizing the KL distance.
- One may encounter the concepts of minimization or maximization of CE.

To address these reasons, it is necessary to briefly review the MLE. The MLE principle assumes that the most reasonable values for $\boldsymbol{\theta}$ are those for which the probability of the observed sample is highest. Since $q_{\boldsymbol{\theta}}(y|\mathbf{x})$ is the PDF model, we have to follow the log-likelihood function

$$\mathcal{L}_{\text{ML}}(\boldsymbol{\theta}) = \sum_{i=1}^N \sum_{y \in \mathcal{C}} \log q_{\boldsymbol{\theta}}(y|\mathbf{x}_i),$$

which is up to the factor $-\frac{1}{N \cdot L}$ the same as the objective (2.3). The value of this factor does not affect the actual optimization, but the negative sign is important because it makes the difference between minimizing and maximizing. Further optimization of $\mathcal{L}_{\text{ML}}(\boldsymbol{\theta})$ gives the point estimate

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{ML}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^N \sum_{y \in \mathcal{C}} \log q_{\boldsymbol{\theta}}(y|\mathbf{x}_i) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \sum_{i=1}^N \sum_{y \in \mathcal{C}} \log q_{\boldsymbol{\theta}}(y|\mathbf{x}_i)\end{aligned}\tag{2.8}$$

It is much more common to minimize a function than to maximize it in practice, and therefore the log-likelihood function is inverted by adding a negative sign to the front of the first part of Equation (2.8) yielding a negative log-likelihood or simply cross-entropy.

2.2.2 One-hot encoding

Machine learning (ML) algorithms can misinterpret the numeric values of labels if there exists a hierarchy between them. One-hot encoding is a very common approach for dealing with this issue in order to improve the algorithm performance. Each unique category value is transformed into a new column, and then these dummy variables are filled with 0 or 1 (0 for FALSE and 1 for TRUE). For the sake of clarity, the transformation of a label encoding into a one-hot encoding is illustrated in the following table 2.1.

However, this method has its own downsides. For example, it creates new variables and if there exist many unique category values, the models have to deal with a large number of predictors, leading to the so-called *Big-p problem* [39]. Also, one-hot encoding causes multicollinearity between the individual variables, which may lead to reducing the model's accuracy.

Food Name	Categorical #	Calories		Pizza	Hamburger	Caviar	Calories
Pizza	1	266	⇒	1	0	0	266
Hamburger	2	295		0	1	0	295
Caviar	3	264		0	0	1	264

Table 2.1: Transformation of a label encoding (left) to the one-hot encoding (right).

2.3 Generative modeling

2.3.1 Variational autoencoder

The first approach to generative modeling that will be discussed is the variational autoencoder (VAE) [18, 19, 42], which is classified as an UL method. In this section, motivation will be addressed and individual mathematical aspects will be discussed in detail.

Problem scenario

Assume that the data $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ are generated by some random process involving an unobserved continuous variable \mathbf{z} , which will be referenced as a latent variable or code. The objective is again to find the PDF of the given data in parametric form $p_{\boldsymbol{\theta}}(\mathbf{x})$. One can choose an approximate distribution in the form of

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z}) d\mathbf{z}, \quad (2.9)$$

But such an approximation is usually very expensive to compute or can even be intractable. Intractability of the $p_{\boldsymbol{\theta}}(\mathbf{x})$ makes posterior PDF $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ also intractable.

Naive approach

One of the simplest ways to solve this problem may seem to be to build a model depending on the latent variable $f_{\boldsymbol{\theta}}(\mathbf{z})$ and try to train its parameters. For simplicity, let the prior distribution be $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbb{I}_P)$, where P denotes the dimension of the latent space \mathbf{z} , and also let

$$\mathbf{x} = f_{\boldsymbol{\theta}}(\mathbf{z}) + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{\varepsilon}; \mathbf{0}, \sigma^2 \cdot \mathbb{I}_D)$$

which actually gives

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_{\boldsymbol{\theta}}(\mathbf{z}), \sigma^2 \cdot \mathbb{I}_D). \quad (2.10)$$

It may be noted that the subscript D represents the dimension of the data point \mathbf{x} . The true PDF of the given data can be cleverly written using the empirical PDF, i.e., in the form of $\tilde{p}(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \delta(\mathbf{x} - \mathbf{x}_i)$, which can be exploited by finding the parameters $\boldsymbol{\theta}$ by minimizing $D_{\text{KL}}(\tilde{p}(\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x}))$. Since minimizing the KL distance is equivalent to MLE and using the approximate form (2.9), the following holds

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \sum_{i=1}^N \log p_{\boldsymbol{\theta}}(\mathbf{x}_i) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \sum_{i=1}^N \log \int \mathcal{N}(\mathbf{x}_i; f_{\boldsymbol{\theta}}(\mathbf{z}), \sigma^2 \cdot \mathbb{I}_D) \cdot \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbb{I}_P) d\mathbf{z} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \sum_{i=1}^N \log \sum_{j=1}^P \exp\left(-\frac{1}{2\sigma^2} (\mathbf{x}_i - f_{\boldsymbol{\theta}}(\mathbf{z}_j))^{\top} (\mathbf{x}_i - f_{\boldsymbol{\theta}}(\mathbf{z}_j))\right). \end{aligned} \quad (2.11)$$

Integration over \mathbf{z} is represented by sampling. In iterations, for an incorrect value of $\boldsymbol{\theta}$, all the generated samples may be away from the samples of \mathbf{x} , and the gradient is poor.

Variational Bayes approach

To solve this problem, it is necessary to introduce a further approximate posterior distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \approx p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ with parameters $\boldsymbol{\phi}$, preferably Gaussian. Standard terminology

refers to the model $q_\phi(\mathbf{z}|\mathbf{x})$ as probabilistic *encoder* and $p_\theta(\mathbf{x}|\mathbf{z})$ is called probabilistic *decoder*. For VAE, the idea is to use the KL distance from $q_\phi(\mathbf{z}|\mathbf{x})$ to $p_\theta(\mathbf{z}|\mathbf{x})$, which produces

$$\begin{aligned}
D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} \tag{2.12} \\
&= \log p_\theta(\mathbf{x}) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p(\mathbf{x}|\mathbf{z}) \right] \\
&= \log p_\theta(\mathbf{x}) + D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z})].
\end{aligned}$$

Using the last equality of (2.16), it is possible to rewrite the equation in its typical form

$$\log p_\theta(\mathbf{x}) - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}))}_{= -L(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})}, \tag{2.13}$$

where the right-hand side is called *Variational Lower Bound* for a single data point. There is no uniformity in terminology, and thus one can also encounter the name Evidence Lower Bound (ELBO) with associated maximizing ELBO. The first term on the right-hand side is known as reconstruction loss, and the second term is often called a regularization term. In order to have more control over optimization, a multiplicative hyper-parameter β is often added in front of the regularization term. ELBO is subsequently rewritten as

$$-L^\beta(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta \cdot D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})), \tag{2.14}$$

which is more formally known as β -VAE [19]. Varying β allows us to control the increment of KL distance to be in numbers similar to the reconstruction loss. Clearly, the original VAE framework is achieved for $\beta = 1$. Since a KL distance is always non-negative, it holds

$$\log p_\theta(\mathbf{x}) \geq -L(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}). \tag{2.15}$$

The objective is to maximize the log-likelihood $\log p_\theta(\mathbf{x})$ which is equivalent to minimizing the negative log-likelihood and that is what will be used here. At this point, we have a lower bound for one data point \mathbf{x} , but we need to include all observations in the lower bound. The joint log-likelihood can be rewritten as a sum over the marginal log-likelihoods of individual observations $\log p_\theta(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) = \sum_{i=1}^N \log p_\theta(\mathbf{x}_i)$ that completes all the building blocks needed to determine the optimization equation. This formulation provides one major advantage, which is that it is now possible to jointly optimize both the generative parameters $\boldsymbol{\theta}$

and the variational parameters ϕ as follows

$$\begin{aligned}
\hat{\theta}, \hat{\phi} &= \operatorname{argmin}_{\theta, \phi} - \sum_{i=1}^N \log p_{\theta}(x_i) \\
&= \operatorname{argmin}_{\theta, \phi} \sum_{i=1}^N L(\theta, \phi; x_i) \\
&= \operatorname{argmin}_{\theta, \phi} - \sum_{i=1}^N \mathbb{E}_{q_{\phi}(z|x_i)} [\log p_{\theta}(x_i|z)] - D_{\text{KL}}(q_{\phi}(z|x_i) \| p_{\theta}(z)).
\end{aligned} \tag{2.16}$$

For a better understanding of the problem, a VAE diagram is shown in Figure 2.4. Note that the latent space is usually much smaller than the input space, and for this reason, it is also sometimes called the bottleneck.

Reparameterization trick

The key success of VAE lies in the fact that Equation (2.16) can be efficiently computed using *reparameterization trick*. We express z as a deterministic variable

$$z = g_{\phi}(\epsilon, x), \tag{2.17}$$

where ϵ stands for an auxiliary variable with independent marginal $p(\epsilon)$ and $g_{\phi}(\cdot)$ is a function parameterized by ϕ .

A common explanation for this trick is that during the optimization the gradient cannot back-propagate through a random node. So, in the case of VAE, the reparameterization trick shifts the source of randomness to another variable different from z and allows differentiation with respect to z . However, this explanation may not be sufficient and for this reason we will state a more formal justification [38]. Consider taking the gradient with respect to θ of $\mathbb{E}_{p(z)}[f_{\theta}(z)]$. It can be easily computed as

$$\begin{aligned}
\nabla_{\theta} \mathbb{E}_{p(z)}[f_{\theta}(z)] &= \nabla_{\theta} \int p(z) f_{\theta}(z) dz \\
&= \int p(z) \nabla_{\theta} f_{\theta}(z) dz \\
&= \mathbb{E}_{p(z)}[\nabla_{\theta} f_{\theta}(z)].
\end{aligned} \tag{2.18}$$

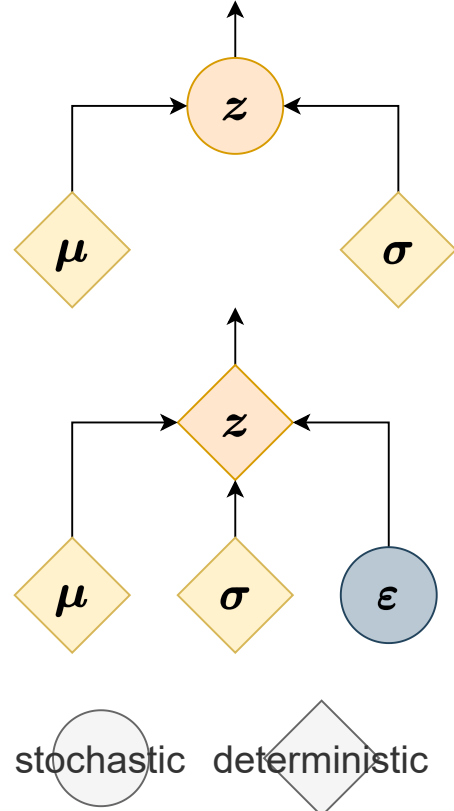


Figure 2.3: Reparametrization trick.

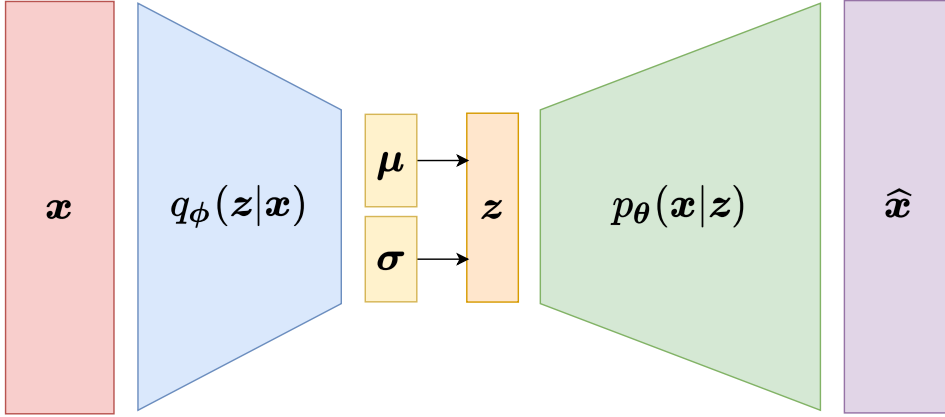


Figure 2.4: VAE diagram, where the input \mathbf{x} is passed through VAE architecture to create reconstructed input $\hat{\mathbf{x}}$.

The result is obvious; the gradient of the expectation is equal to the expectation of the gradient. However, the gradient of the expectation becomes much more interesting if the PDF $p_{\theta}(\mathbf{z})$ is also parameterized by θ , resulting in

$$\begin{aligned}
 \nabla_{\theta} \mathbb{E}_{p_{\theta}(\mathbf{z})} [f_{\theta}(\mathbf{z})] &= \nabla_{\theta} \int p_{\theta}(\mathbf{z}) f_{\theta}(\mathbf{z}) d\mathbf{z} \\
 &= \int p_{\theta}(\mathbf{z}) \nabla_{\theta} f_{\theta}(\mathbf{z}) d\mathbf{z} + \int f_{\theta}(\mathbf{z}) \nabla_{\theta} p_{\theta}(\mathbf{z}) d\mathbf{z} \\
 &= \mathbb{E}_{p_{\theta}(\mathbf{z})} [\nabla_{\theta} f_{\theta}(\mathbf{z})] + \int f_{\theta}(\mathbf{z}) \nabla_{\theta} p_{\theta}(\mathbf{z}) d\mathbf{z}.
 \end{aligned} \tag{2.19}$$

The second term of (2.19) is not guaranteed to be an expectation and this very fact indicates that back-propagation would not compute an estimate of $\nabla_{\theta} \mathbb{E}_{p_{\theta}(\mathbf{z})} [f_{\theta}(\mathbf{z})]$. That being the case, if we apply the reparameterization trick $\mathbf{z} = g_{\theta}(\boldsymbol{\varepsilon}, \mathbf{x})$ to this simple example, we get

$$\mathbb{E}_{p_{\theta}(\mathbf{z})} [f_{\theta}(\mathbf{z})] = \mathbb{E}_{p(\boldsymbol{\varepsilon})} [f(g_{\theta}(\boldsymbol{\varepsilon}, \mathbf{x}))]. \tag{2.20}$$

At this point, it is possible to take the gradient $\nabla_{\theta} \mathbb{E}_{p(\boldsymbol{\varepsilon})} [f(g_{\theta}(\boldsymbol{\varepsilon}, \mathbf{x}))]$ analogously to that in (2.18). To be perfectly clear, the authors of [18] proposed an easy exercise. Take the univariate Gaussian case $p(z|x) = \mathcal{N}(z; \mu, \sigma^2)$. In such a case, proper reparameterization takes the form of

$$z = \mu + \sigma \varepsilon, \tag{2.21}$$

where $\varepsilon \sim \mathcal{N}(0, 1)$ and, therefore, the expectation

$$\mathbb{E}_{\mathcal{N}(z; \mu, \sigma^2)} [f(z)] = \mathbb{E}_{\mathcal{N}(\varepsilon; 0, 1)} [f(\mu + \sigma \varepsilon)] \approx \frac{1}{M} \sum_{j=1}^M f(\mu + \sigma \varepsilon_j).$$

Note that this is nothing more than a transformation of a random variable. If we look closer at (2.16), the expectation on the right-hand side, i.e. $\mathbb{E}_{q_{\phi}(z|x_i)}$, is taken over the parameterized PDF. This must be rewritten using the reparameterization trick so that the Monte Carlo estimate of the expected value is differentiable with respect to ϕ .

Variational autoencoder

So far, we have only dealt with VAE in general. In this section, we put everything together and specify the individual parts of the ELBO (2.16). Let the probabilistic encoder be a multivariate Gaussian with a diagonal covariance matrix

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_{\phi}, \boldsymbol{\sigma}_{\phi}^2 \mathbb{I}_P) \quad (2.22)$$

and let the probabilistic decoder $p_{\theta}(\mathbf{x}|\mathbf{z})$ take the form depending on the type of given data and model. This is typically either multivariate Gaussian or Bernoulli. Finally, let the prior $p_{\theta}(\mathbf{z})$ be the centered isotropic multivariate Gaussian, i.e.

$$p_{\theta}(\mathbf{z}) = p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbb{I}_P), \quad (2.23)$$

where the generative parameters θ are omitted, since the chosen prior distribution lacks parameters. When using (2.22), $\boldsymbol{\mu}_{\phi}$ and $\boldsymbol{\sigma}_{\phi}$ are non-linear functions of the data point \mathbf{x} and the variational parameters ϕ . For further simplification of the notation, the index ϕ will be omitted. This setting actually allows us to take the reparameterization trick in a form similar to that of Equation (2.21), which means that

$$\mathbf{z}_{i,j} = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\varepsilon}_j, \quad (2.24)$$

where the symbol \odot denotes the Hadamard product, i.e. the element product and the auxiliary variable $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbb{I}_P)$. Another major fact is that the KL distance from a Gaussian distribution to a Gaussian distribution has an analytical solution (for a full derivation, see the Appendix A.1), so $D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))$ can be expressed in closed form:

$$\begin{aligned} D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z})) &= D_{\text{KL}}(\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \mathbb{I}_P) \| \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbb{I}_P)) \\ &= \frac{1}{2} \sum_{j=1}^P \left(-1 - \log \sigma_j^2 + \mu_j^2 + \sigma_j^2 \right). \end{aligned} \quad (2.25)$$

Now all that is left is to plug everything into equation (2.16), which leads to the final form for optimization

$$\begin{aligned} \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\theta}, \boldsymbol{\phi}}{\operatorname{argmin}} - \sum_{i=1}^N \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}_i)} [\log p_{\theta}(\mathbf{x}_i|\mathbf{z})] - D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}_i) \| p_{\theta}(\mathbf{z})) \\ &= \underset{\boldsymbol{\theta}, \boldsymbol{\phi}}{\operatorname{argmin}} - \sum_{i=1}^N \left(\frac{1}{P} \sum_{j=1}^P \log p_{\theta}(\mathbf{x}_i|\mathbf{z}_{i,j}) + \frac{1}{2} \sum_{j=1}^P \left(1 + \log \sigma_{i,j}^2 - \mu_{i,j}^2 - \sigma_{i,j}^2 \right) \right). \end{aligned} \quad (2.26)$$

Toy problem

The objective of this example is to verify that VAE can be utilized to generate new data points. Assume that we have a data set of 2D i.i.d. observations $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ generated from the unknown distribution and that we would like to sample new observations from this distribution. We should see similar patterns of the true and estimated samples. Let the

probabilistic decoder be in Gaussian form 2.10, but with the identity matrix as a covariance matrix, therefore,

$$p_{\theta}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_{\theta}(\mathbf{z}), \mathbb{I}_2). \quad (2.27)$$

This allows us to once again rewrite Equation (2.26) in a concrete optimizable form

$$\begin{aligned} \hat{\theta}, \hat{\phi} &= \underset{\theta, \phi}{\operatorname{argmin}} - \sum_{i=1}^N \left(\frac{1}{P} \sum_{j=1}^P \log \mathcal{N}(\mathbf{x}_i; f_{\theta}(\mathbf{z}_{i,j}), \mathbb{I}_2) + \frac{\beta}{2} \sum_{j=1}^P (1 + \log \sigma_{i,j}^2 - \mu_{i,j}^2 - \sigma_{i,j}^2) \right) \\ &= \underset{\theta, \phi}{\operatorname{argmin}} - \sum_{i=1}^N \left(\frac{1}{P} \sum_{j=1}^P (\mathbf{x}_i - f(\mathbf{z}_{i,j}))^{\top} (\mathbf{x}_i - f(\mathbf{z}_{i,j})) + \frac{\beta}{2} \sum_{j=1}^P (1 + \log \sigma_{i,j}^2 - \mu_{i,j}^2 - \sigma_{i,j}^2) \right), \end{aligned} \quad (2.28)$$

where f_{θ}, μ and σ are represented via the neural network (NN). These are in fact dense layers, i.e., the NN layer, where neurons are connected to every neuron of its preceding layer. For the non-linearity of μ and σ we choose SELU and for f_{θ} it is identity. Clearly, new data points $\hat{\mathbf{x}}$ (reconstructed input) are then sampled using $\hat{\mathbf{x}} = \hat{f}_{\theta}(\mathbf{z})$ with $\mathbf{z} \sim \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbb{I}_P)$, thus there is no need to transform the output. The input layer is the encoder, which is again a dense layer with SELU non-linearity. The hyper-parameter $\beta = 0.2$ is added to ensure a similar increment of KL, see

(2.14). For the training of parameters $\hat{\theta}$ and $\hat{\phi}$, the Adam¹ optimization algorithm [37] is used. It is sufficient to initialize optimization with standard default values, learning rate $\alpha = 0.001$, decay rates $\beta_1 = 0.9$, $\beta_2 = 0.999$, and finally $\epsilon = 10^{-8}$. The results are shown in Figure 2.5, where the true data and estimated data are depicted. The estimated distribution is very close to the true distribution, as the pattern of the samples is indistinguishable. This experiment revealed that VAE is a good way of successfully generating new data points.

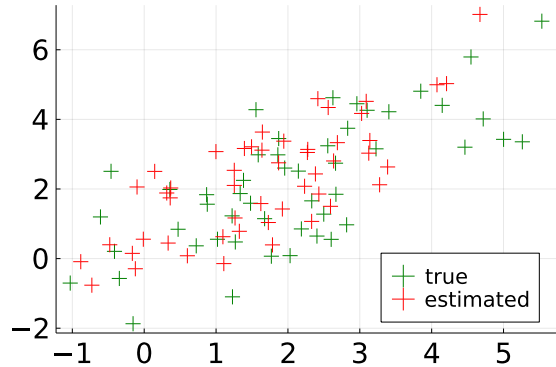


Figure 2.5: True and estimated samples using VAE.

2.3.2 Semi-supervised variational autoencoder

Semi-Supervised Variational Autoencoder (SSVAE) copes with input-output pair samples \mathcal{D} as defined in (1.3). Each pair sample (\mathbf{x}_i, y_i) has its corresponding latent variable \mathbf{z}_i . The authors of [15] propose a probabilistic model that describes the data as generated by a latent class variable y in addition to a continuous latent variable \mathbf{z} . However, only a subset of observations \mathbf{x} has the corresponding class labels. Observe that these latent variables are marginally independent. Empirical distributions of labeled and unlabeled subsets are denoted by $\tilde{p}_l(\mathbf{x}, y)$ and $\tilde{p}_u(\mathbf{x})$, respectively. As with standard VAE, the data is generated by

¹The Adam optimization algorithm is a standard optimization tool implemented in most of the programming languages. As opposed to stochastic gradient descent, this method computes individual adaptive learning rates for different parameters from estimates of the first and second moments of the gradients.

some random process which can be described as follows

$$p(y) = \text{Cat}(y; \boldsymbol{\pi}), \quad p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbb{I}_p), \quad p_{\boldsymbol{\theta}}(\mathbf{x}|y, \mathbf{z}) = h_{\boldsymbol{\theta}}(\mathbf{x}; y, \mathbf{z}).$$

The symbol $\text{Cat}(y; \boldsymbol{\pi})$ denotes the multinomial distribution with probability vector $\boldsymbol{\pi}$, and $h_{\boldsymbol{\theta}}$ is a suitable likelihood function depending on the type of given data parameterized by a non-linear transformation of the latent variables. Predictions of missing labels are obtained from the inferred posterior distribution $p_{\boldsymbol{\theta}}(y|\mathbf{x})$. The standard VAE employs an inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, however, in the case of semi-supervised learning, this model should also contain class labels. Ideally, for a given observation \mathbf{x} , the model should predict the class label y and, in addition, be able to construct the latent space \mathbf{z} for given \mathbf{x} and y . Under these circumstances, the model is introduced in the factorized form $q_{\boldsymbol{\phi}}(\mathbf{z}, y|\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{z}|y, \mathbf{x}) q_{\boldsymbol{\phi}}(y|\mathbf{x})$. This factorization is specified further as

$$q_{\boldsymbol{\phi}}(\mathbf{z}|y, \mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_{\boldsymbol{\phi}}(y, \mathbf{x}), \text{diag}\left(\boldsymbol{\sigma}_{\boldsymbol{\phi}}^2(\mathbf{x})\right)\right), \quad q_{\boldsymbol{\phi}}(y|\mathbf{x}) = \text{Cat}(y; \boldsymbol{\pi}_{\boldsymbol{\phi}}(\mathbf{x})),$$

where $\boldsymbol{\mu}_{\boldsymbol{\phi}}$, $\boldsymbol{\sigma}_{\boldsymbol{\phi}}$ and $\boldsymbol{\pi}_{\boldsymbol{\phi}}$ are represented as NNs. For VAE, we derived the variational lower bound

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})) = -L(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}),$$

from which we now derive lower bounds for SSVAE. This derivation consists of two steps.

Latent Feature Discriminative Model (M1): First, consider observation \mathbf{x} that has its class label y . The variational lower bound is then easily extended as

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\mathbf{x}, y) &\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, y)} [\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, y) + \log p_{\boldsymbol{\theta}}(y)] - D_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, y) \| p_{\boldsymbol{\theta}}(\mathbf{z})) \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, y)} [\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, y) + \log p_{\boldsymbol{\theta}}(y) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, y) + \log p_{\boldsymbol{\theta}}(\mathbf{z})] \\ &= -J(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}, y), \end{aligned} \quad (2.29)$$

where $\log p_{\boldsymbol{\theta}}(y)$ is a constant that determines the ratio between classes. If the classes are similarly large, this term becomes irrelevant. When necessary, it is again possible to weigh the KL distance using the parameter β (2.14).

Generative Semi-supervised Model Objective (M2): In the case of observation \mathbf{x} lacking its class label y , it is treated as another latent variable over which posterior inference is performed. We get

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\mathbf{x}) &\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(y, \mathbf{z}|\mathbf{x})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, y) + \log p_{\boldsymbol{\theta}}(y)] - D_{\text{KL}}(q_{\boldsymbol{\phi}}(y, \mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})) \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(y, \mathbf{z}|\mathbf{x})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, y) + \log p_{\boldsymbol{\theta}}(y) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, y) + \log p_{\boldsymbol{\theta}}(\mathbf{z}) + \log q_{\boldsymbol{\phi}}(y|\mathbf{x})] \\ &= \sum_y q_{\boldsymbol{\phi}}(y|\mathbf{x}) J(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}, y) + H(q_{\boldsymbol{\phi}}(y|\mathbf{x})) = -U(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}), \end{aligned} \quad (2.30)$$

where $H(q_{\boldsymbol{\phi}}(y|\mathbf{x}))$ indicates the entropy of $q_{\boldsymbol{\phi}}(y|\mathbf{x})$.

To include the entire data set in the bound, we sum (2.29) over the labeled subset and (2.30) over the unlabeled subset, so that we can write

$$\tilde{J}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{(x, y) \sim \tilde{p}_l(x, y)} J(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}, y) + \sum_{\mathbf{x} \sim \tilde{p}_u(\mathbf{x})} U(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}). \quad (2.31)$$

This objective lacks the predictive distribution of the label $q_\phi(y|\mathbf{x})$ in its first expression. Since the purpose of this distribution is to use it as a classifier, we need to ensure that its parameters are learned in all cases. Currently, the objective (2.31) would be completely lacking $q_\phi(y|\mathbf{x})$ if all data were labeled. To fix this problem, it is suggested to add the classification loss to (2.31), which yields the concise form

$$\tilde{J}^\nu(\boldsymbol{\theta}, \boldsymbol{\phi}) = \tilde{J}(\boldsymbol{\theta}, \boldsymbol{\phi}) + \nu \cdot \mathbb{E}_{\tilde{p}_l(\mathbf{x}, y)} [-\log q_\phi(y|\mathbf{x})]. \quad (2.32)$$

This gives a hybrid combination of generative and purely discriminative modeling. The hyper-parameter ν weighs the discriminative counterpart with a common value of $\nu = 0.1\tilde{N}$, where \tilde{N} denotes the size of the supervised data set. Following Equation (2.26), optimization of (2.32) can be performed jointly

$$\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}} = \operatorname{argmin}_{\boldsymbol{\theta}, \boldsymbol{\phi}} \tilde{J}^\nu(\boldsymbol{\theta}, \boldsymbol{\phi}).$$

The key factor in this formulation is that the optimization of the predictor and the generator is linked and proceeds simultaneously. The trained predictor should perform better than a predictor that is trained separately without the generator.

2.3.3 Noise-contrastive estimation

Suppose one has to estimate a model that is specified by a non-normalized probability density function $q_\theta^0(\mathbf{x})$. In such a case, one can utilize noise-contrastive estimation (NCE) [43, 44]. The first step is to introduce another parameter c among the estimated parameters $\boldsymbol{\theta}$. For clarity, the symbol $\boldsymbol{\theta}^* = \{\boldsymbol{\theta}, c\}$ is introduced for the set of estimated parameters, including c . Using this notation, we can write the following equality

$$\log q_{\boldsymbol{\theta}^*}(\mathbf{x}) = \log q_{\boldsymbol{\theta}^*}^0(\mathbf{x}) + c, \quad (2.33)$$

which means that the newly introduced parameter c is an estimate of the negative logarithm of the normalization constant $Z(\boldsymbol{\theta})$ (1.2). As the name suggests, we use noise to estimate. By our convention, let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ be observations and $\boldsymbol{\Xi} = \{\boldsymbol{\varepsilon}_1, \boldsymbol{\varepsilon}_2, \dots, \boldsymbol{\varepsilon}_N\}$ be artificially generated noise data with known distribution $\psi(\boldsymbol{\varepsilon})$. The estimate $\hat{\boldsymbol{\theta}}^*$ is then defined as

$$\begin{aligned} \hat{\boldsymbol{\theta}}^* &= \operatorname{argmax}_{\boldsymbol{\theta}^*} \mathcal{L}^{\text{NC}}(\boldsymbol{\theta}^*) \\ &= \operatorname{argmax}_{\boldsymbol{\theta}^*} \frac{1}{2N} \sum_{i=1}^N \log S_{\boldsymbol{\theta}^*}(\mathbf{x}_i) + \log(1 - S_{\boldsymbol{\theta}^*}(\boldsymbol{\varepsilon}_i)) \\ &= \operatorname{argmin}_{\boldsymbol{\theta}^*} -\frac{1}{2N} \sum_{i=1}^N \log S_{\boldsymbol{\theta}^*}(\mathbf{x}_i) + \log(1 - S_{\boldsymbol{\theta}^*}(\boldsymbol{\varepsilon}_i)) \end{aligned} \quad (2.34)$$

where $S_{\boldsymbol{\theta}^*}$ stands for a logistic function,

$$S_{\boldsymbol{\theta}^*}(\mathbf{x}) = \frac{1}{1 + \exp(-G_{\boldsymbol{\theta}^*}(\mathbf{x}))} \quad (2.35)$$

and finally, the function G_{θ^*} represents the difference of the log-likelihoods of q_{θ^*} and ψ , hence

$$G_{\theta^*}(\mathbf{x}) = \log q_{\theta^*}(\mathbf{x}) - \log \psi(\mathbf{x}). \quad (2.36)$$

It may be noted that equation (2.34) also appears in SL tasks and is called binary CE loss. It is actually a special case of CE itself. Thus, it is used for the classification of two classes. This gives an intuitive insight into how noise-contrastive estimation really works. When data and noise are compared, the model is learned, so this method can be called learning by comparison. To make the connection with SL more explicit, denote $U = \{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{2N}\}$ the union of two sets \mathbf{X} and $\mathbf{\Xi}$. Then each data point \mathbf{u}_i is assigned a binary class label y_i , where $y_i = 1$ if $\mathbf{u}_i \in \mathbf{X}$ and $y_i = 0$ if $\mathbf{u}_i \in \mathbf{\Xi}$. The aim is to estimate the posterior probabilities of the classes given the data \mathbf{u}_i . To do this, one needs the class-conditional PDFs that are given by

$$p(\mathbf{u} | y = 1) = q_{\theta^*}(\mathbf{u}) \quad p(\mathbf{u} | y = 0) = \psi(\mathbf{u}).$$

Class labels are equally likely, so that $\Pr(y = 1) = \Pr(y = 0) = \frac{1}{2}$ and the posteriors are determined as follows

$$\Pr(y = 1 | \mathbf{u}) = \frac{q_{\theta^*}(\mathbf{u})}{q_{\theta^*}(\mathbf{u}) + \psi(\mathbf{u})} = S_{\theta^*}(\mathbf{u}), \quad (2.37)$$

$$\Pr(y = 0 | \mathbf{u}) = 1 - S_{\theta^*}(\mathbf{u}). \quad (2.38)$$

The class labels y_i are Bernoulli-distributed so that for the log-likelihood of Bernoulli with probabilities (2.37) and (2.38), we get

$$\begin{aligned} \mathcal{L}^{\text{NC}}(\boldsymbol{\theta}) &= \sum_{i=1}^{2N} y_i \log \Pr(y = 1 | \mathbf{u}_i) + (1 - y_i) \log \Pr(y = 0 | \mathbf{u}_i) \\ &= \sum_{i=1}^N \log S_{\theta^*}(\mathbf{x}_i) + \log(1 - S_{\theta^*}(\boldsymbol{\epsilon}_i)), \end{aligned} \quad (2.39)$$

which is the equation (up to the extrinsic factor $\frac{1}{2N}$) that is optimized in Equation (2.34).

Choice of the contrastive noise PDF

The noise distribution $\psi(\boldsymbol{\epsilon})$ can be considered as a design parameter. But this choice is not completely arbitrary, because in practice the noise distribution should meet certain conditions. These are:

1. It is easy to sample from, because NCE approach relies on artificially generated noise data $\boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2, \dots, \boldsymbol{\epsilon}_N$.
2. In order to smoothly evaluate (2.36), closed form for $\log \psi(\cdot)$ is requisite.
3. It leads to a small mean squared error $\mathbb{E} \left[\left(\widehat{\boldsymbol{\theta}^*} - \boldsymbol{\theta}^* \right)^2 \right]$.

The authors of [44] suggest using a Gaussian or uniform distribution, eventually a Gaussian mixture.

EXAMPLE 2.1 (One-dimensional Gaussian distribution). To test this approach, we performed a simple experiment. There are a total of $N = 100$ i.i.d. and one-dimensional observations x_1, x_2, \dots, x_N from an unknown distribution that is assumed to be Gaussian. Therefore, it is of the form

$$q_{\theta^*}(x) = \exp\left(-\frac{1}{2} \cdot \frac{(x - \mu)^2}{\sigma^2} + c\right), \quad (2.40)$$

where $\theta^* = \{\mu, \sigma^2, c\}$. Since c is an estimate of the negative logarithm of the normalization constant $Z(\theta)$, we can include it into the exponent. Next, we artificially generate noise data e_1, e_2, \dots, e_N , which is again easier to do using a Gaussian distribution. This means that it can be chosen, for example,

$$\psi(e) = \frac{1}{\sqrt{2\pi}10} \exp\left(-\frac{1}{2} \cdot \frac{e^2}{10}\right). \quad (2.41)$$

We choose the noise PDF intentionally so widely spread from its mean value because these two PDFs, i.e. (2.40) and (2.41), should at least partially overlap. At this point, we have all the components available and it is possible to construct a function (2.37) that is minimized using the Adam optimization algorithm [37]. Figure 2.6 shows the comparison between the estimated distribution and the true one. The estimate is very close to the actual

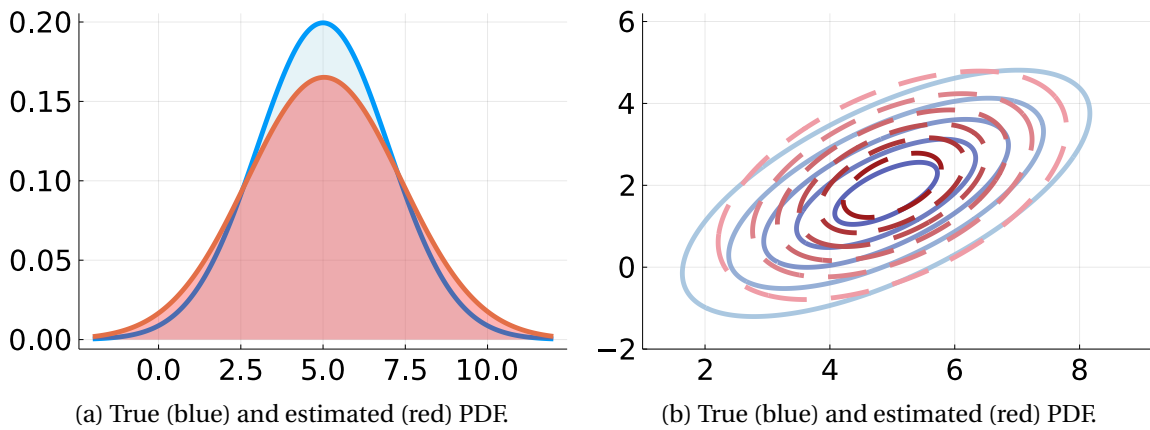


Figure 2.6: Results of the NCE experiment for one-dimensional and two-dimensional Gaussian case.

distribution, and for an increasing number of observations, the estimate would get closer and closer.

EXAMPLE 2.2 (Two-dimensional Gaussian distribution). The one-dimensional case may seem too simple, and therefore an example with a two-dimensional Gaussian distribution was performed. The experimental setup remains nearly the same; only the dimensionality of the problem differs. Recall that the multivariate Gaussian distribution in \mathbb{R}^2 can be written as

$$q_{\theta^*}(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) + c\right),$$

where $\boldsymbol{\mu} \in \mathbb{R}^2$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{2 \times 2}$ is a symmetric and positive semidefinite covariance matrix. As the noise PDF is chosen $\psi(\mathbf{e}) = \mathcal{N}(\mathbf{e}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$, where $\boldsymbol{\mu}_1 = (2, 2)^\top$ and $\boldsymbol{\Sigma}_1 = 10 \cdot \mathbb{I}_2$. Figure 2.6 shows the results in a vein similar to that in the previous case.

3

Hybrid Generative and Discriminative Models

In the previous chapter, we have introduced the basics of discriminative and generative modeling. An interesting and key approach is SSSVAE, which combines both types of modeling. This chapter attempts to do the same, but from a slightly different perspective. In total, we review two methods. The first method combining discriminative and generative models will be based on contrastive learning, and the second method will use the aforementioned SSSVAE.

3.1 Energy-based models

In the first part of this chapter, we review the theory of energy-based models (EBM) [26, 27, 28, 29]. It is motivated by statistical physics and aims at PDF estimation. Given a large data set, we want to estimate the PDF over the entire data space. Assuming that we are modeling images from, for example, the CIFAR-10¹ data set [41], the goal is to estimate a PDF over all possible images of size $32 \times 32 \times 3$, where those images have a high likelihood of looking realistic and are one of the CIFAR classes. Nowadays, images are extremely high-dimensional and that is why simple methods such as interpolation between images fail.

The fundamental idea of EBMs is to transform any function that predicts values larger than zero into a PDF by dividing by its volume (normalization constant). This implies that the probability densities $p_{\theta}(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^D$ in EBM are assumed to be expressed in the form

$$p_{\theta}(\mathbf{x}) = \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z(\theta)}, \quad (3.1)$$

¹The CIFAR-10 data set consists of 60000 32x32 color images in 10 different classes.

where $Z(\boldsymbol{\theta})$ is a normalization constant and $E_{\boldsymbol{\theta}} : \mathbb{R}^D \rightarrow \mathbb{R}$ is called the *energy* function with parameters $\boldsymbol{\theta}$, which maps each data point \mathbf{x} to a scalar. In front of the objective $E_{\boldsymbol{\theta}}(\mathbf{x})$ there is a negative sign because we want data points with high likelihood to have low energy, while data points with low likelihood should have high energy. The significant advantage of the density formulation of EBM is that there are a relatively large number of ways to choose energy $E_{\boldsymbol{\theta}}$. The exponential function captures major variations in probability, and log-likelihood is a natural scale to work with. Unfortunately, we cannot compute $Z(\boldsymbol{\theta})$ for the vast majority of them (not even numerically, since computing time scales exponentially in the number of dimensions of \mathbf{x}), and we have to rely on the Monte Carlo estimate. The idea here is to take the gradient with respect to $\boldsymbol{\theta}$ from the log-likelihood $\log p_{\boldsymbol{\theta}}(\mathbf{x})$, which decomposes as the sum of two terms

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) = -(\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}) + \nabla_{\boldsymbol{\theta}} \log Z(\boldsymbol{\theta})). \quad (3.2)$$

The second term can be rewritten as expectation $\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}[-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})]$, which yields the following (for the full derivation, see Appendix A.2)

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}[-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})] - \nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}). \quad (3.3)$$

Drawing samples from $p_{\boldsymbol{\theta}}(\mathbf{x})$ is far from trivial, thus we exploit a well-established Monte Carlo method called stochastic gradient Langevin dynamics (SGLD) [45]. For any continuous $p_{\boldsymbol{\theta}}(\mathbf{x})$ we can compute the score function

$$\nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\mathbf{x}} E_{\boldsymbol{\theta}}(\mathbf{x}), \quad (3.4)$$

and generate samples using the following stochastic process

$$\mathbf{x}_0 \sim \psi(\mathbf{x}), \quad \mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla_{\mathbf{x}} E_{\boldsymbol{\theta}}(\mathbf{x}) + \sqrt{2\eta} \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbb{I}_D), \quad t \in \{0, 1, \dots, T-1\}, \quad (3.5)$$

where $\psi(\mathbf{x})$ is the prior distribution (that is easy to sample from, such as Gaussian) used to generate the initial sample \mathbf{x}_0 and $\eta \in \mathbb{R}$ is a step size. This algorithm ensures that for $\eta \rightarrow 0$ and $T \rightarrow \infty$, \mathbf{x}_T is distributed as $p_{\boldsymbol{\theta}}(\mathbf{x})$ [28].

Note that for two different data points \mathbf{x} and \mathbf{x}^* , computing $p_{\boldsymbol{\theta}}(\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x}^*)$ requires $Z(\boldsymbol{\theta})$, however, the ratio $p_{\boldsymbol{\theta}}(\mathbf{x}) / p_{\boldsymbol{\theta}}(\mathbf{x}^*)$ does not involve $Z(\boldsymbol{\theta})$ and one can easily check which data point is more likely.

3.1.1 Joint energy models

Recall, that in Section 2.2 we described the Softmax function, which is used to model the true data distribution. It is defined as

$$q_{\boldsymbol{\theta}}(y|\mathbf{x}) = \frac{\exp(f_{\boldsymbol{\theta}}(\mathbf{x})[y])}{\sum_{y \in \mathcal{C}} \exp(f_{\boldsymbol{\theta}}(\mathbf{x})[y])}.$$

Crucial observation is made by the authors in [27], where they show that supervised learning classifiers are secretly EBMs on $p_{\boldsymbol{\theta}}(\mathbf{x}, y)$, i.e., the logits $f_{\boldsymbol{\theta}}(\mathbf{x})[y]$ in objective (2.2) can be seen as defining an EBM so that the joint PDF can be expressed as

$$p_{\boldsymbol{\theta}}(\mathbf{x}, y) = \frac{\exp(f_{\boldsymbol{\theta}}(\mathbf{x})[y])}{Z(\boldsymbol{\theta})}. \quad (3.6)$$

This objective is called *joint energy model* (JEM), and it is obvious that $f_{\theta}(\mathbf{x})[y] = -E_{\theta}(\mathbf{x}, y)$. The desirable model $p_{\theta}(\mathbf{x})$ can be obtained by marginalizing (3.6) over y , resulting in the following density

$$p_{\theta}(\mathbf{x}) = \frac{\sum_{y \in \mathcal{C}} \exp(f_{\theta}(\mathbf{x})[y])}{Z(\theta)}, \quad (3.7)$$

where the energy is given by $E_{\theta}(\mathbf{x}) = -\log \sum_{y \in \mathcal{C}} \exp(f_{\theta}(\mathbf{x})[y])$. A very useful property appears when computing $p_{\theta}(y|\mathbf{x})$, because we can take advantage of the definition of a conditional distribution $p_{\theta}(y|\mathbf{x}) = p_{\theta}(\mathbf{x}, y) / p_{\theta}(\mathbf{x})$. Substituting (3.6) and (3.7) into this conditional distribution results in

$$p_{\theta}(y|\mathbf{x}) = q_{\theta}(y|\mathbf{x}) = \frac{\exp(f_{\theta}(\mathbf{x})[y])}{\sum_{y \in \mathcal{C}} \exp(f_{\theta}(\mathbf{x})[y])}. \quad (3.8)$$

Note that the normalization constant $Z(\theta)$ canceled out and we ended up with the same function, which was introduced in (2.2).

3.2 Contrastive learning

Contrastive learning [12, 13] is an ML technique used to learn the so-called general features of a data set by teaching the model which data points are similar or different. All this happens without labels; therefore, contrastive learning is frequently called the *self-supervised* technique of ML. Given $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, in contrastive learning problems, it is very common to optimize an objective often called contrastive loss, which can be written in the form as follows

$$\text{CL}(\theta) = -\mathbb{E}_{\tilde{p}(\mathbf{x})} \left[\log \frac{\exp(m_{\theta}(\mathbf{x}) \cdot m_{\theta}(\mathbf{x}'))}{\sum_{i=1}^M \exp(m_{\theta}(\mathbf{x}) \cdot m_{\theta}(\mathbf{x}_i))} \right], \quad (3.9)$$

where $M < N$ denotes the number of normalization samples. The function $m_{\theta} : \mathbb{R}^D \rightarrow \mathbb{R}^H$ maps each data point to a representation space of dimension H , while \mathbf{x} and \mathbf{x}' are two different augmented views of the same data point. If \mathbf{x} is an image, then an augmented view of \mathbf{x} can be obtained, for example, by rotating or colorizing that image. Note that the inner product between two vectors can be replaced with any distance metric, for instance, the Euclidean distance.

This objective tries to maximally distinguish an input \mathbf{x}_i from an alternative input \mathbf{x}'_i . In other words, (3.9) reduces the distance between the representations of different augmented views of the same image \mathbf{x}, \mathbf{x}' (positive pairs) and increases the distance between the representations of augmented views of different images (negative pairs). This means that the model should be able to distinguish between different types of image without even knowing what these images really are.

3.3 Hybrid discriminative and generative models

In this section, we will put everything together and present an approach to combine both types of models. The authors of article [14] proposed a solution, however, the rationale for

this objective originates from [23], where the authors show that hybrid models can outperform their purely generative or purely discriminative counterparts.

To achieve this goal, a hybrid model consists of a discriminative conditional and a generative conditional, and it is trained by minimizing the negative sum of both conditional log-likelihoods, concretely

$$\min_{\theta} -\mathbb{E}_{\tilde{p}(\mathbf{x},y)} [\log q_{\theta}(y|\mathbf{x}) + \log q_{\theta}(\mathbf{x}|y)], \quad (3.10)$$

where the first term is a standard Softmax NN classifier (as mentioned in Equation (2.2) or (3.8)), while the second term differs from Softmax in its denominator, so that

$$q_{\theta}(\mathbf{x}|y) = \frac{\exp(f_{\theta}(\mathbf{x})[y])}{\sum_{i=1}^N \exp(f_{\theta}(\mathbf{x}_i)[y])}. \quad (3.11)$$

The objective (3.11) can cause serious problems with the unknown normalization constant $\sum_{i=1}^N \exp(f_{\theta}(\mathbf{x}_i)[y])$, which is often intractable, as we have already mentioned in previous considerations. The authors of [14] recommend resolving this obstacle using an approximation via contrastive loss

$$\mathbb{E}_{\tilde{p}(\mathbf{x},y)} [\log q_{\theta}(\mathbf{x}|y)] = \mathbb{E}_{\tilde{p}(\mathbf{x},y)} \left[\log \frac{\exp(f_{\theta}(\mathbf{x})[y])}{\sum_{i=1}^N \exp(f_{\theta}(\mathbf{x}_i)[y])} \right] \approx \mathbb{E}_{\tilde{p}(\mathbf{x},y)} \left[\log \frac{\exp(f_{\theta}(\mathbf{x})[y])}{\sum_{i=1}^M \exp(f_{\theta}(\mathbf{x}_i)[y])} \right], \quad (3.12)$$

where $M < N$ denotes the number of normalization samples. This loss is related to (3.9), which was mentioned in the previous section, but there are few distinctions to discuss. We use labels in this formulation as we assume a supervised setting and, more importantly, we do not use any mapping m_{θ} or augmented views. The main contribution of (3.9) is the proposed approximation in the denominator. To have an adequate approximation, M must be sufficiently large, becoming exact in the limit $M \rightarrow N$. In practice, increasing M is not straightforward as it requires a larger memory. However, this does not apply to our experiments.

Now it is possible to substitute the approximation (3.12) into Equation (3.10), which results in a hybrid combination of supervised learning and constrastive learning in the form of

$$\begin{aligned} & \min_{\theta} \text{HDGM}(\theta; \alpha) \\ & = \min_{\theta} -\mathbb{E}_{\tilde{p}(\mathbf{x},y)} [\alpha \log q_{\theta}(y|\mathbf{x}) + (1 - \alpha) \log q_{\theta}(\mathbf{x}|y)] \\ & \approx \min_{\theta} -\mathbb{E}_{\tilde{p}(\mathbf{x},y)} \left[\alpha \log \frac{\exp(f_{\theta}(\mathbf{x})[y])}{\sum_{y \in \mathcal{C}} \exp(f_{\theta}(\mathbf{x})[y])} + (1 - \alpha) \log \frac{\exp(f_{\theta}(\mathbf{x})[y])}{\sum_{i=1}^M \exp(f_{\theta}(\mathbf{x}_i)[y])} \right], \end{aligned} \quad (3.13)$$

called the Hybrid Discriminative Generative energy-based Model (HDGM). The newly introduced hyper-parameter α is a weight between $[0, 1]$. It is obvious that in the case of $\alpha = 1$, the objective is reduced to the standard cross-entropy loss, while in $\alpha = 0$, the objective is reduced to a case called *end-to-end supervised version of contrastive learning*. The choice of hyper-parameter α is a decision of the experiment designer, however, the authors of [14] evaluated many possible variants in their experiments and found that the choice of $\alpha = 0.5$

produces the highest classification accuracy performance. Unfortunately, these experiments involved only image classification.

The HDGM (3.13) is absolutely crucial for us as we extend this approach to the multi-instance learning problem, but this is discussed in further sections.

3.3.1 Toy problem - polynomial regression

First and foremost, we test the HDGM in a simple example before moving on to more difficult cases. Assume that we have generated the synthetic data $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$, where $x_i, y_i \in \mathbb{R}$, therefore, this is only a two-dimensional problem. However, the data are poor. The goal is to train a model using the form (3.13) derived in the previous section and observe the effect of the generative part on the resultant model.

According to the energy-based models 3.1, we know that for the joint distribution, it holds

$$p_{\theta}(x, y) = \frac{\exp(f_{\theta}(x)[y])}{Z(\theta)}, \quad (3.14)$$

where the energy is given by $f_{\theta}(x)[y] = -E_{\theta}(x, y)$. The general model of polynomial regression is written as

$$f_{\theta}(x) = \sum_{i=0}^{s-1} \theta_i x^i. \quad (3.15)$$

At this point, we transform this problem into a polynomial regression. We must be aware of the discriminative term in Equation (3.10), because we do not want to classify, but our objective is to find the best fit to the given data. For this reason, we replace $\log q_{\theta}(y|x)$ with the typical regression loss, i.e., the residual sum of squares

$$S = S(\theta) = \sum_{k=1}^N \left(y_k - \sum_{i=0}^{s-1} \theta_i x_k^i \right)^2. \quad (3.16)$$

Generally, any joint probability distribution can be broken down into parts by the chain rule. In this case, a decomposition of the form

$$p_{\theta}(x, y) = p_{\theta}(y, x) = p_{\theta}(y|x) \cdot p(x) \quad (3.17)$$

is appropriate. Therefore, we need to find $p_{\theta}(y|x)$ and $p(x)$. From polynomial regression, we can obtain the conditional PDF

$$p_{\theta}(y|x) = \mathcal{N} \left(y; \sum_{i=0}^{s-1} \theta_i x^i, \sigma^2 \right), \quad (3.18)$$

where the variance σ^2 is considered the known parameter. In this example, we also need to determine the prior distribution of x . To keep this example simple, let the prior be Gaussian as well as (3.18), so that

$$p(x|\tau) = \mathcal{N}(x; 0, \tau^2), \quad (3.19)$$

where the choice of parameter τ is based on the fact that we do not want to influence the given observations too much. This leads to a non-informative prior and thus τ should be

adequately high. If the value of τ is high, the data are spread very far from their expected value. Substituting equations (3.19) and (3.18) into (3.17) results in

$$p_{\boldsymbol{\theta}}(x, y) = \mathcal{N}(x; 0, \tau^2) \cdot \mathcal{N}\left(y; \sum_{i=0}^{s-1} \theta_i x^i, \sigma^2\right) = \frac{1}{2\pi\sigma\tau} \exp\left(-\frac{\left(y - \sum_{i=0}^{s-1} \theta_i x^i\right)^2}{2\sigma^2} - \frac{x^2}{2\tau^2}\right), \quad (3.20)$$

whereas logits of the model are given by

$$f_{\boldsymbol{\theta}}(x)[y] = -\frac{\left(y - \sum_{i=0}^{s-1} \theta_i x^i\right)^2}{2\sigma^2} - \frac{x^2}{2\tau^2}. \quad (3.21)$$

We can now substitute (3.21) and (3.16) in Equation (3.10), resulting in

$$\begin{aligned} & \min_{\boldsymbol{\theta}} \left\{ \alpha S(\boldsymbol{\theta}) - \mathbb{E}_{\tilde{p}(x,y)} \left[(1 - \alpha) \log q_{\boldsymbol{\theta}}(x|y) \right] \right\} = \\ & \min_{\boldsymbol{\theta}} \left\{ \alpha S(\boldsymbol{\theta}) - \mathbb{E}_{\tilde{p}(x,y)} \left[(1 - \alpha) \log \frac{\exp(f_{\boldsymbol{\theta}}(x)[y])}{\sum_{i=1}^N \exp(f_{\boldsymbol{\theta}}(x_i)[y])} \right] \right\} = \\ & \min_{\boldsymbol{\theta}} \left\{ \alpha S(\boldsymbol{\theta}) - \mathbb{E}_{\tilde{p}(x,y)} \left[(1 - \alpha) \log \frac{\exp\left(-\frac{\left(y - \sum_{i=0}^{s-1} \theta_i x^i\right)^2}{2\sigma^2} - \frac{x^2}{2\tau^2}\right)}{\sum_{k=1}^N \exp\left(-\frac{\left(y - \sum_{i=0}^{s-1} \theta_i x_k^i\right)^2}{2\sigma^2} - \frac{x_k^2}{2\tau^2}\right)} \right] \right\}. \end{aligned} \quad (3.22)$$

Note that for $\alpha = 1$ we get purely polynomial regression, and for $\alpha = 0$, the term $S(\boldsymbol{\theta})$ is not involved at all. The estimated parameters are then obtained as

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \alpha S(\boldsymbol{\theta}) - \mathbb{E}_{\tilde{p}(x,y)} \left[(1 - \alpha) \log \frac{\exp\left(-\frac{\left(y - \sum_{i=0}^{s-1} \theta_i x^i\right)^2}{2\sigma^2} - \frac{x^2}{2\tau^2}\right)}{\sum_{k=1}^N \exp\left(-\frac{\left(y - \sum_{i=0}^{s-1} \theta_i x_k^i\right)^2}{2\sigma^2} - \frac{x_k^2}{2\tau^2}\right)} \right] \right\}. \quad (3.23)$$

Now, we have everything we need to carry out the experiment, since Equation (3.22) is the final optimization form. For optimization of Equation (3.23), Adam with default values is used.

3.3.2 Experiment setup and results

We generate synthetic data \mathcal{D} , two clusters consisting of five data points each. This makes the data very poor. Standard polynomial regression is probably not the best tool here; in the best possible way, the generative component should improve the polynomial regression itself. We have three parameters, α , $s - 1$ and τ , which will be varied during the experiment. We want to observe how the estimated model behaves in relation to the hyper-parameter α . The model is then fitted for different weights $\alpha \in \{0.0, 0.1, 0.2, \dots, 1.0\}$, giving 11 different models in total. The models estimated for $\alpha \in \{0.0, 0.5, 1.0\}$ are highlighted, as they are more important to us than the other models. Let this experiment be further divided into two parts according to $s - 1$ and τ .

Fixed order of the polynomial

For the first part, we train the models for the fixed order of the polynomial $s - 1 = 10$, but for six different values of the parameter $\tau \in \{0.1, 1, 10, 100, 1000, 10000\}$. The results obtained (Figures 3.4, 3.5 and 3.6) for small τ barely vary from those for high τ and one can observe a mere effect of τ . This phenomenon is exactly what we hoped for, as the prior distribution should be non-informative (3.19). Additionally, it can be observed that the polynomial model itself, i.e., the $\alpha = 1$ curve, is probably not the best for prediction. From this point of view, we would prefer to choose the model trained for $\alpha = 0.2$ or $\alpha = 0.1$, since these models exhibit a lower degree of oscillation.

Fixed prior parameter

In the second part of this experiment, we train our polynomial models for the fixed value of $\tau = 100$, but this time with 6 different values of the order of the polynomial, concretely, $s - 1 \in \{2, 4, 5, 6, 8, 10\}$. The goal of this part is to observe how the contrastive part of Equation (3.22) affects the polynomial regression for different values of $s - 1$. As can be seen in Figures 3.1, 3.2, and 3.3, for small $s - 1$, such as $s - 1 = 2$, the term $\log q_{\theta}(x|y)$ does not have a crucial influence. However, we get a considerable difference between models for higher orders of the polynomial. Furthermore, the curve for $\alpha = 0$ prefers not to oscillate. It seems that due to the low complexity of the model, the generative component is not important. However, with increasing complexity, its importance increases. From this point of view, we can say that we have managed to improve the polynomial model with the help of the generative component.

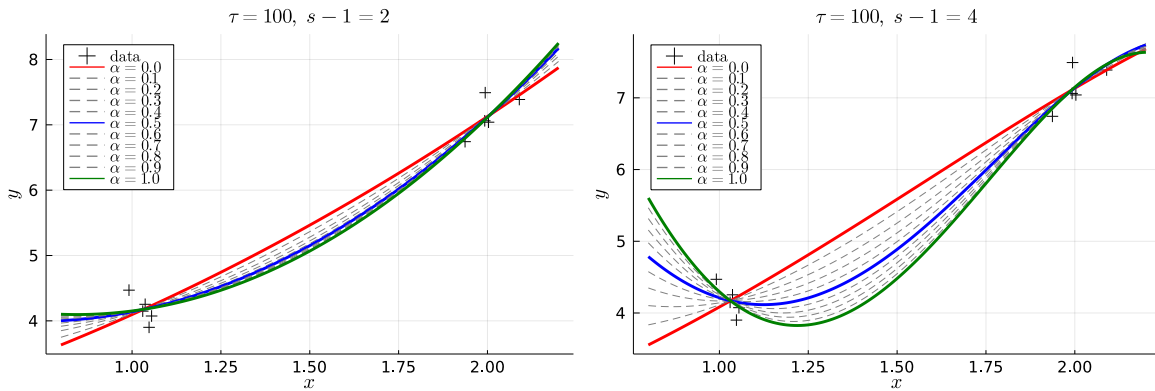


Figure 3.1: Sensitivity of the polynomial model to the order of the polynomial $s - 1$, specifically $s - 1 \in \{2, 4\}$.

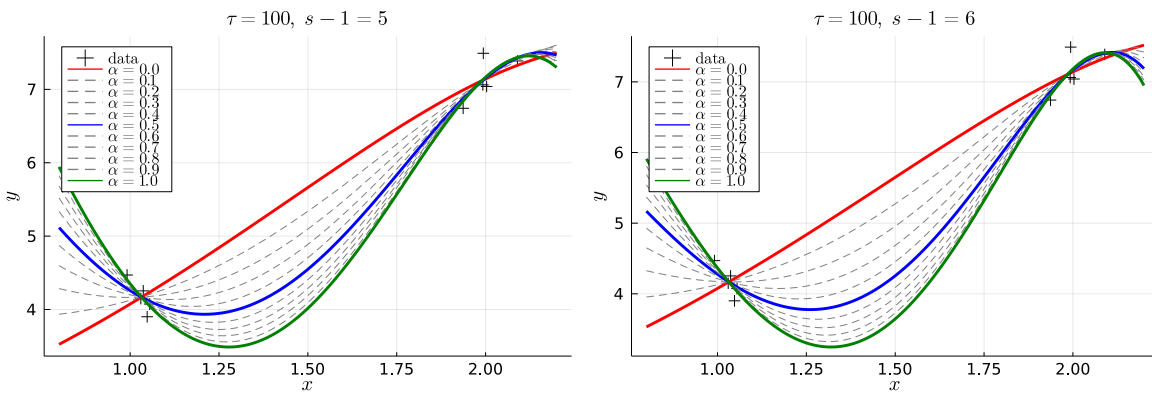


Figure 3.2: Sensitivity of the polynomial model to the order of the polynomial $s - 1$, specifically $s - 1 \in \{5, 6\}$.

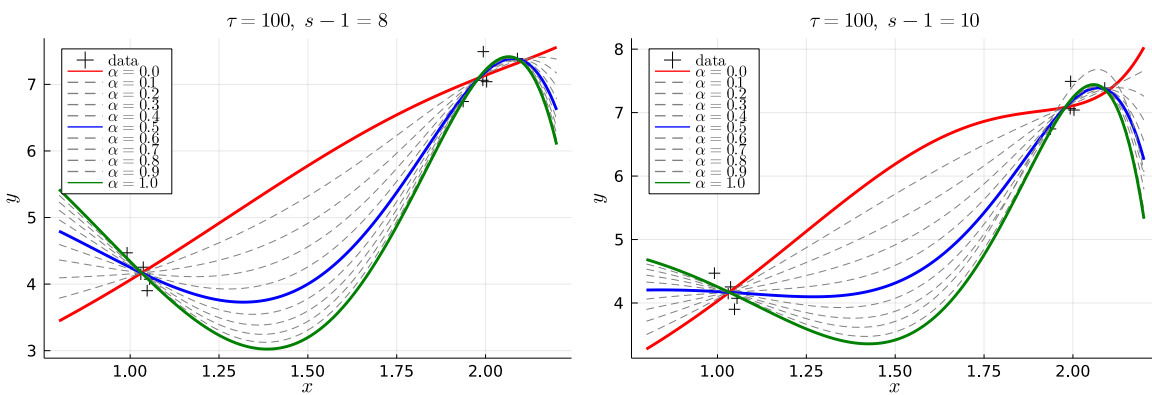


Figure 3.3: Sensitivity of the polynomial model to the order of the polynomial $s - 1$, specifically $s - 1 \in \{8, 10\}$.

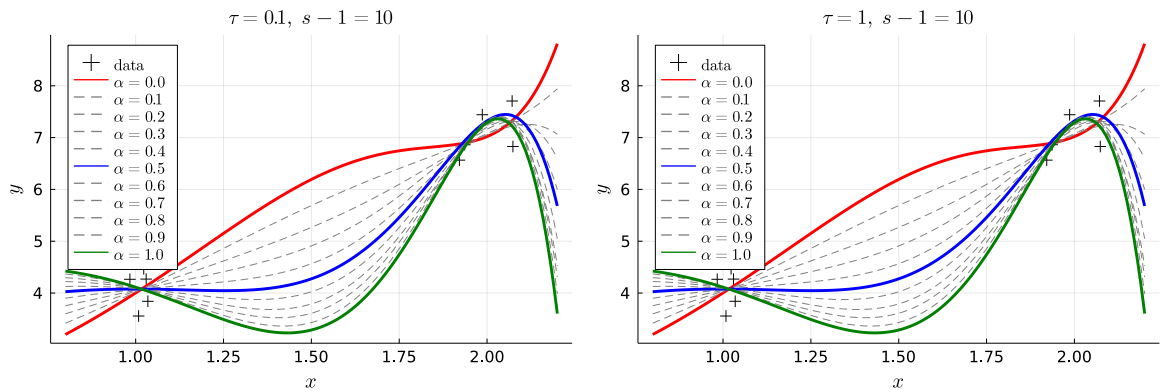


Figure 3.4: Sensitivity of the polynomial model to the parameter τ , specifically $\tau \in \{0.1, 1\}$.

+

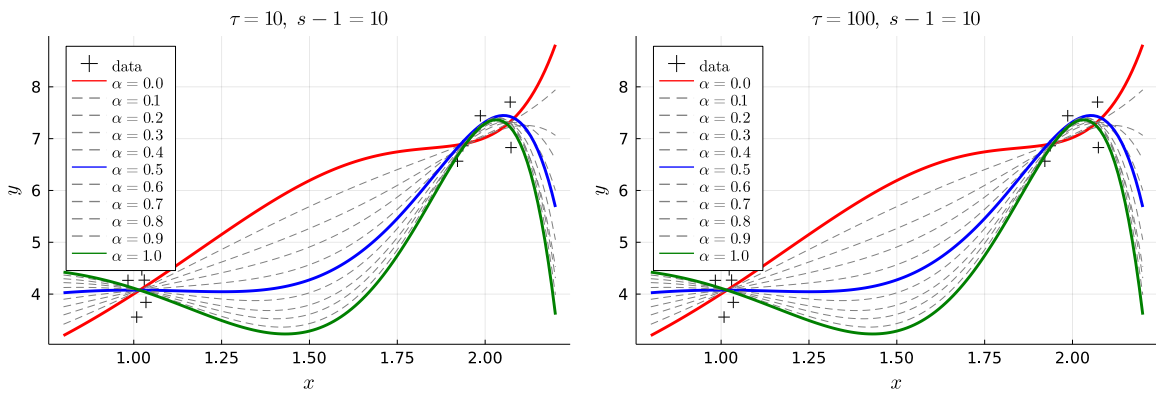


Figure 3.5: Sensitivity of the polynomial model to the parameter τ , specifically $\tau \in \{10, 100\}$.

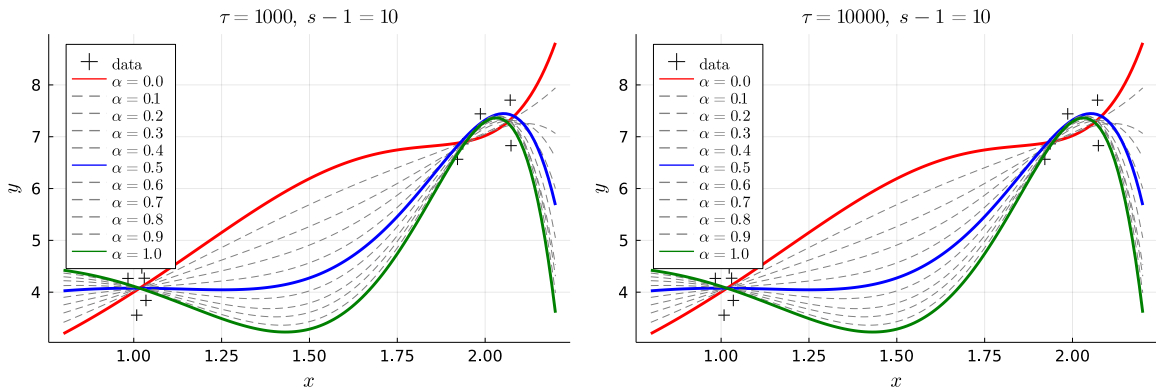


Figure 3.6: Sensitivity of the polynomial model to the parameter τ , specifically $\tau \in \{10^3, 10^4\}$.

3.4 Hybrid VAE

In section 2.3.2, we derived SSSVAE, whose predictor can fill in missing data labels. To achieve this, we constructed two lower bounds (2.29) and (2.30). We can slightly reinterpret this methodology for a fully supervised data set, where the predictor is used to predict labels for new data. Based on [15], we can split this approach into two models. The first model optimizes the predictor and generator separately, while the second model has the predictor and generator linked. This ensures that optimization is performed jointly.

Separate model (SM): For training of SM, we use only the lower bound $-J(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}, y)$ and standard cross-entropy loss. The total loss that is minimized is then

$$J_{SM}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{(\mathbf{x}, y) \sim \tilde{p}_l(\mathbf{x}, y)} J(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}, y) + \mathbb{E}_{\tilde{p}_l(\mathbf{x}, y)} [-\log q_{\boldsymbol{\phi}}(y|\mathbf{x})]. \quad (3.24)$$

Observe that the lower bound (2.29) does not contain the predictor $q_{\boldsymbol{\phi}}(y|\mathbf{x})$ and the optimization of these two expressions proceeds independently of each other.

Hybrid model (HM): The second phase deals with the inclusion of the predictor $q_{\boldsymbol{\phi}}(y|\mathbf{x})$ in both summands. To do this, we use $-U(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})$, which does not participate in SM. Since we operate with a completely supervised data set, we cannot sum over \mathbf{x} from an unlabeled subset. However, the summation is performed over \mathbf{x} and y from the labeled data set. We get the modified version of Equation (2.31), that is

$$\begin{aligned} \tilde{J}_{HM}^v(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \sum_{(\mathbf{x}, y) \sim \tilde{p}_l(\mathbf{x}, y)} (J(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}, y) + U(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}, y)) + v \cdot \mathbb{E}_{\tilde{p}_l(\mathbf{x}, y)} [-\log q_{\boldsymbol{\phi}}(y|\mathbf{x})] \\ &= \tilde{J}_{HM}(\boldsymbol{\theta}, \boldsymbol{\phi}) + v \cdot \mathbb{E}_{\tilde{p}_l(\mathbf{x}, y)} [-\log q_{\boldsymbol{\phi}}(y|\mathbf{x})]. \end{aligned} \quad (3.25)$$

From Equation (2.30) we know that $-U(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})$ contains the predictor $q_{\boldsymbol{\phi}}(y|\mathbf{x})$. Unlike (3.24), the optimization of (3.25) trains the predictor that is linked to the generator. A very important factor here is the hyper-parameter v , which has a fatal influence on the correct training of the predictor.

3.4.1 Toy problem

In the following experiment, we clarify this method, concretize the individual components, and simplify it for our purposes. The objective here is to compare the performance of a predictor of HM with that of a separately trained predictor in SM. In the first phase, we train the SM and compute its AUC-ROC. In the second phase, we will test whether for some values of the hyper-parameter v the predictor of HM achieves a higher AUC-ROC.

Training and testing data are synthetically generated as 2-moons, i.e., two interleaving half circles. Therefore, it is a type of the well-known N-Moons [40], often used to visualize clustering and classification algorithms. Note that this data set is completely supervised with 2 balanced classes, which means that the predictor is used to predict the class for new data, not to fill in missing labels. We already mentioned the meaningfulness of the term

$\log p_{\theta}(y)$, which occurs in both bounds (2.29) and (2.30). Both classes are equally represented in the data set, hence we do not take this term into account, and it is not included.

In addition, we can use and adjust the settings from the standard VAE experiment (2.28), since $-J(\theta, \phi; \mathbf{x}, y)$ actually corresponds to $-L(\theta, \phi; \mathbf{x})$. It only needs to be extended with class labels y so that the input layer accepts both \mathbf{x} and y and generator f_{θ} accepts both \mathbf{z} and y . Specifically, f_{θ} consists of 2 dense layers with SELU non-linearity and identity, μ and σ are simple dense layers with SELU non-linearity, and finally the predictor $q_{\phi}(y|\mathbf{x})$ is stacked with 2 dense layers with SELU and Softmax non-linearity as the output layer. Note that observations are first encoded using a single dense layer with stereotypical SELU non-linearity.

First, we train the SM model and compute the AUC-ROC_{SM} on the predictor. The next step is to find the values of ν where the AUC-ROC_{HM} for the predictor of HM is higher than the AUC-ROC_{SM} . Figure 3.7 illustrates the results. Note that both models were trained by Adam with default values. The HM predictor performs better than the SM predictor for approximately $\nu > 9000$. Concretely, $\text{AUC-ROC}_{HM}(\nu = 10500) = 96.68\%$ and for SM model we get $\text{AUC-ROC}_{SM} = 95.48\%$. This improvement is 1.2% compared to SM, and thus we can say that our hybrid model is fully functional. An interesting finding is also the fact that for very

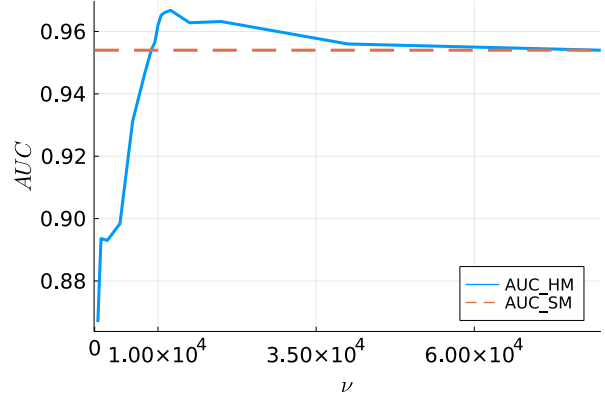


Figure 3.7: AUC-ROC for both SM and HM.

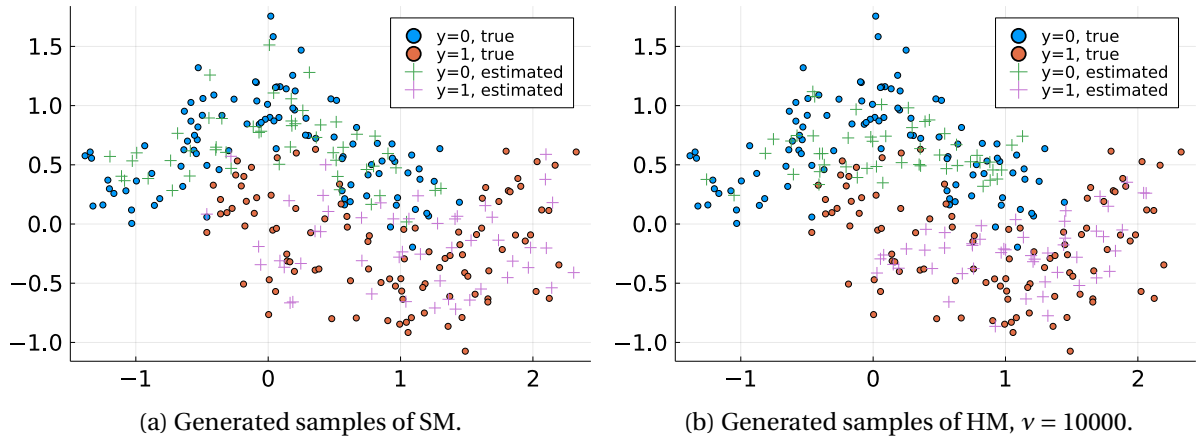


Figure 3.8: Comparison of the generated samples of the SM and HM models. The train data are identical in both cases.

small values of ν the AUC-ROC_{HM} drops rapidly. Therefore, care should be taken to ensure that the model is set up correctly. On the other hand, for higher values of ν , the AUC-ROC_{HM} asymptotically approaches the AUC-ROC_{SM} .

The question remains as to how well the generators are trained. If the predictors are trained well but the generators work badly, the whole method would be useless. We compare the newly generated data for both SM and HM with the training data in Figure 3.8. As can be seen, the samples generated for both classes correspond to the training samples for both models.

4

Multiple Instance Learning

4.1 Fundamentals

The term multi-instance learning (MIL) (or multiple instance learning) originates from the pioneer work [8], nevertheless the authors of [11] proposed the following nomenclature, which will be reviewed and used gladly in our work.

In standard machine learning (ML) problems, each sample is represented by a fixed vector \mathbf{x} , however, in MIL it is dealt with samples which are represented by a set of vectors. These vectors are called *instances* and come from an instance space \mathcal{X} , for example \mathbb{R}^D . The sets of these instances are called *bags* and come from the bag space $\mathcal{B} = \mathcal{P}_F(\mathcal{X})$, where $\mathcal{P}_F(\mathcal{X})$ denotes all finite subsets of \mathcal{X} . With this in mind, we can easily write any bag as $b = \{\mathbf{x} \in \mathcal{X}\}_{\mathbf{x} \in b}$. Instances in the bag meet the standard i.i.d. assumption. Each bag b can be arbitrarily large or empty, thus the size of the bag is defined in the form $|b| \in \mathbb{N}_0$. There may exist intrinsic labeling of instances, but we are at this point only interested in labeling at the bag levels. Bag labels come from a finite set \mathcal{C} . Unlike ML, where a predictor is learned in the form $f_{\theta} : \mathbb{R}^D \rightarrow \mathcal{C}$, what we want in MIL is to learn a predictor in the form $f_{\theta} : \mathcal{B}(\mathcal{X}) \rightarrow \mathcal{C}$. Note that a predictor can also be rewritten in the form $f_{\theta}(\{\mathbf{x}\}_{\mathbf{x} \in b})$. We consider the supervised setting in which each sample of the data set is assigned a label. We can denote the available data by the notation

$$\mathcal{D}^* = \left\{ (b_i, y_i) \in \mathcal{B} \times \mathcal{C} \mid i \in \{1, 2, \dots, |\mathcal{D}^*|\} \right\}, \quad (4.1)$$

where $|\mathcal{D}^*|$ apparently denotes the size of \mathcal{D}^* . The difference between standard ML and MIL is visualized graphically in Figure 4.1. Note that MIL problems fall into the category of binary classification, therefore $\mathcal{C} = \{-1, +1\}$.

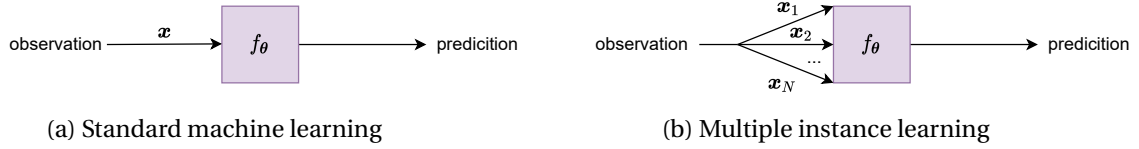


Figure 4.1: The difference between standard ML and MIL. Standard ML is special case of MIL with $|b| = 1$.

4.2 Embedded-space paradigm

Very elegant solution to deal with samples on the level of bags provides an embedded-space paradigm. This paradigm defines a vector space for the representation of bags and specifies a mapping from each bag $b \in \mathcal{B}$ to this space. It is a crucial component in which most MIL methods differ. Assume that the target vector space is \mathbb{R}^M . Let the overall embedding $\Lambda : \mathcal{B} \rightarrow \mathbb{R}^M$ be given by

$$\begin{aligned} \Lambda(b) &= (\lambda_1(b), \lambda_2(b), \dots, \lambda_M(b)) \\ &= (\lambda_1(\{\mathbf{x}\}_{\mathbf{x} \in b}), \lambda_2(\{\mathbf{x}\}_{\mathbf{x} \in b}), \dots, \lambda_M(\{\mathbf{x}\}_{\mathbf{x} \in b})) \in \mathbb{R}^M, \end{aligned} \quad (4.2)$$

where $\lambda_i : \mathcal{B} \rightarrow \mathbb{R}$, $i \in \{1, 2, \dots, M\}$ is a partial mapping or individual projection. Mappings λ_i are instrumental in obtaining and aggregating the appropriate information on the level of instances. They can be defined by some instance transformation $k : \mathcal{X} \rightarrow \mathbb{R}$ and an aggregation function (or pooling) $g : \mathcal{P}_F(\mathbb{R}^{|\mathcal{B}|}) \rightarrow \mathbb{R}$ of the form

$$\lambda_i(b) = g(k\{\mathbf{x}\}_{\mathbf{x} \in b}). \quad (4.3)$$

The most widely used aggregation functions are minimum, maximum, or mean value. On the resulting embedded representation of bag samples, any standard machine learning algorithm can be applied, that is, training a bag-level classifier $f_{\theta}^B : \mathbb{R}^M \rightarrow \mathcal{C}$ using an adjusted data set

$$\mathcal{D}_{\text{ES}}^* = \left\{ (\Lambda(b_i), y_i) \in \mathbb{R}^M \times \mathcal{C} \mid i \in \{1, 2, \dots, |\mathcal{D}^*|\} \right\}. \quad (4.4)$$

Compare (4.4) and (4.1). However, such simple definitions of embedding do not lead to good performance in practice. Consider two bags; these bags may have a similar mean value of all of their instances despite these instances having a completely different structure. It is necessary to proceed to a different embedding solution based on more complex rules. These methods first pre-process the instances and extract the relevant patterns from them in an unsupervised way (no labeling of instances is required). Embedding is then applied to these patterns. The collection of abstract patterns analyzed in the training set is written as $\Lambda_{\Omega}(b)$, where the parameters Ω are usually referred to as vocabulary. These parameters can be considered in both transformation k and aggregation g . For example, the function k is typically implemented as a distance measure between an instance and the pattern.

In summary, three components are required to solve MIL problems, that is, a function that operates at the level of instances k , an aggregation g , and a bag-level classifier f_{θ}^B . The first two components form the embedding $\Lambda_{\Omega}(b)$, where the parameters Ω are optimized together with all other parameters θ in the model. An example of such a pipeline is illustrated in Figure 4.2.

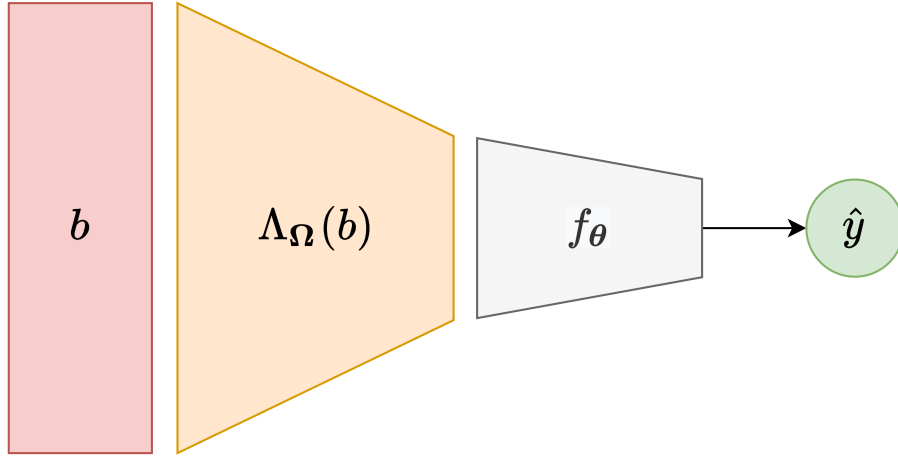


Figure 4.2: MIL diagram, where b is the input bag, f_{θ} is the classifier applied on the resulting embedded space representation $\Lambda_{\Omega}(b)$ of the bag.

4.3 Training

The authors of [11] proposed a versatile unified framework called HMill (Hierarchical multi-instance learning library) for the definition and training of the models and even implemented this functionality in the *Julia* programming language. Furthermore, the framework was published as an open-source project entitled *Mill.jl* under the MIT license [21].

Standard MIL classifier: For our purposes, we use a rather simple NN architecture. Instances are first passed through a single dense layer with $\xi \in \mathbb{N}$ neurons and tanh non-linearity, then mean and maximum aggregations are applied simultaneously (in some cases, maximum is better than mean, and thus it is better to use both). It is followed by another dense layer with ξ neurons (with a standard value of $\xi = 10$) and tanh non-linearity and the output linear dense layer with 2 neurons (since we classify into two classes and use one-hot encoding). Finally, the cross-entropy (2.3) is used as a loss function. Therefore, the training process is similar to a common classification problem. For actual optimization, we stick to the Adam optimizer with the default values analogously to all previous experiments.

4.3.1 Cross-validation

For the MIL testing, we have four data sets available, namely Musk1, Musk2, Tiger, and Fox. These data sets come from the UCI database and are specially modified for modeling with set data. All of them will be used to assess the performance of the MIL model.

4.3.1.1 Setup and results

In this experiment, the data sets are split 100 times *randomly* into two sets in advance, the train and the test sets, with 80% of the observations being in train set and 20% of the observations belonging to the test set. For future simplification, let a number of random

splits be denoted by r , and thus we have $r = 100$. Note that r is not a number of folds. We fit the model in the train set and then evaluate the prediction error in both the train and test set by cross-entropy. The objective here is to plot the dependence of the prediction error on the complexity of the model. A smaller number of random splits, such as $r \in \{25, 50\}$, was tested,

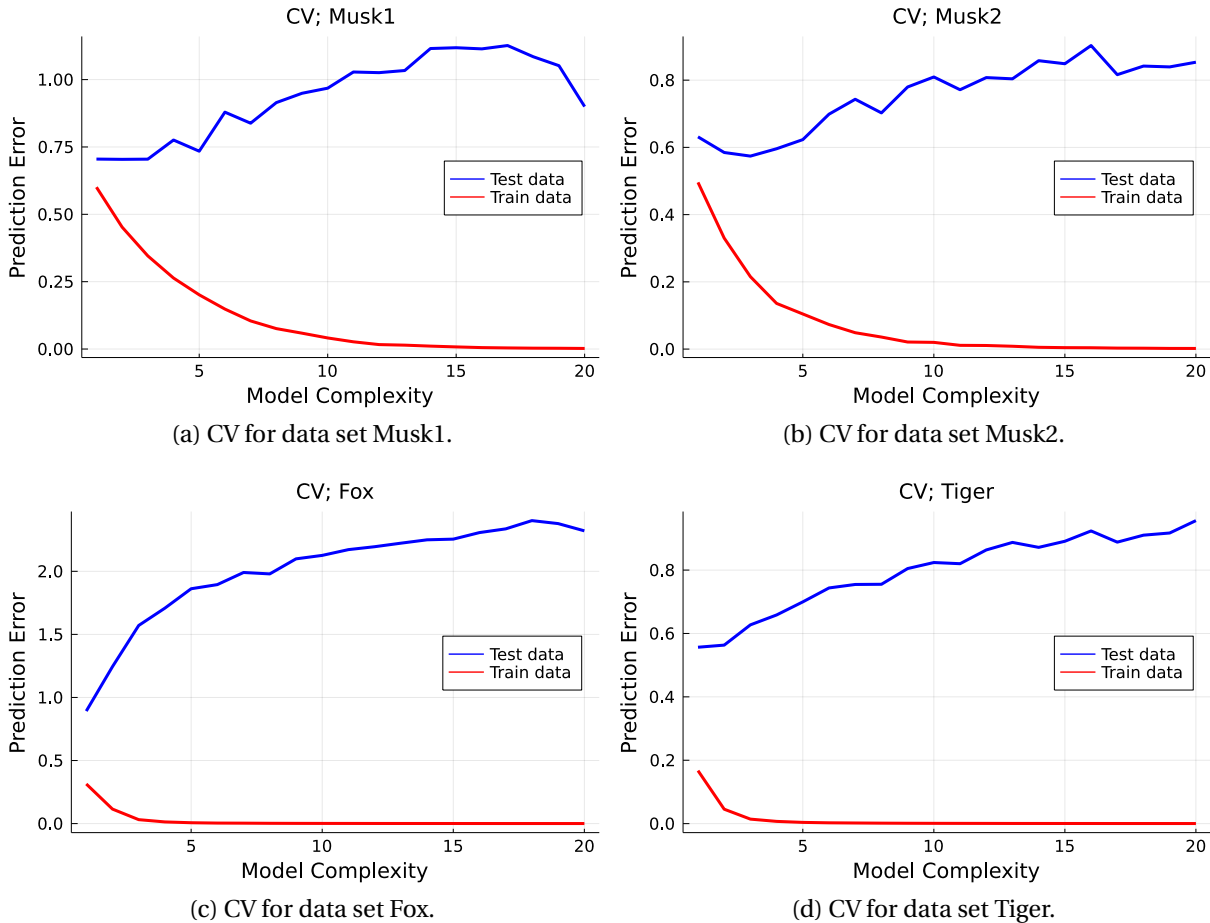


Figure 4.3: Evaluation of prediction error with the use of training data and testing data on MIL data sets Musk1, Musk2, Fox, and Tiger.

however, the results were too noisy. For this reason, it is necessary to select a higher value of r , although the experiment becomes noticeably expensive to compute.

Model Complexity: The suitable candidate for a model complexity seems to be number of neurons ξ in the first and the penultimate layer of the MIL model.

Prediction Error: For prediction error, we simply use the standard cross-entropy loss as outlined at the beginning of this section. There is no need for any trickier objective. To simplify the notation, let the total cross-entropy loss evaluated in the k^{th} random split for

fixed ξ be denoted by $\text{CE}_k(\hat{\theta}, \xi)$, then the estimated prediction error is given by

$$\text{err}(\xi) = \frac{1}{r} \sum_{k=1}^r \text{CE}_k(\hat{\theta}; \xi). \quad (4.5)$$

To summarize, we fit 100 models for selected ξ on train data and evaluate the prediction error for all fitted models on test data, then the mean value is taken over all splits. This process is repeated for each $\xi \in \{1, 2, 3, \dots, 20\}$, giving 2000 models to train in total.

	$\text{argmin err}(\xi)$	$\text{min err}(\xi)$	$\text{err}(\xi = 10)$
Musk1	2	0.70	0.97
Musk2	3	0.57	0.81
Fox	1	0.89	2.13
Tiger	1	0.56	0.82

Table 4.1: Results of CV evaluated on the testing data.

As can be seen in Figure 4.3, the results obtained are expected. The prediction error evaluated on the training data (red curve) for the higher complexity of the model approaches zero. However, testing data (blue curve) give oscillating curves with an

increasing trend (with a little exception of Musk1), therefore, model selection is necessary. This applies to each data set. Furthermore, Table 4.1 numerically summarizes the results evaluated in the test data. The first column of the table represents the best choice of model complexity ξ for the given MIL model.

4.4 Application of HDGM to the MIL problems

In the previous part, the CV experiment was performed with the expected results. However, the calculated prediction error on the test data is quite high. The question is whether there is any method to reduce the prediction error of the MIL model. The name of the section suggests that it will be attempted using the HDGM.

4.4.1 Setup and results

On the initiative of reducing the prediction error, it is proposed to train an MIL model that is obtained by minimizing the hybrid loss function. Since the discriminative part is already used in the HMill framework, the only task is to add the generative part. At this point, we do not operate with an enormous number of data points. The data sets we use have hundreds of bags at maximum, and thus we can normalize the generative term over all bags without approximation. Then the MIL model remains the same as in the previous experiment; the only difference is in the loss function. Once again, we split this experiment into two parts.

Dependence on hyper-parameter α

In the first part of this experiment, we train the models in relation to the hyper-parameter α . For this setup, we need to choose a fixed ξ . Since the authors of [11] usually set $\xi = 10$, we use

this value as well. For prediction error evaluation, standard cross-entropy is used as in the previous experiment. Once again, the number of splits is $r = 100$ and the weights are identical to the polynomial regression example, that is $\alpha \in \{0.0, 0.1, 0.2, \dots, 1.0\}$. Recall that for the value of $\alpha = 1$, Equation (3.13) is reduced to the standard cross-entropy. For further simplification, let the total hybrid loss calculated on the k^{th} split be written as $\text{HDMG}_k(\hat{\theta}; \xi = 10, \alpha)$. Therefore, the calculated prediction error can be written in the form

$$\text{err}(\alpha) = \frac{1}{r} \sum_{k=1}^r \text{HDMG}_k(\hat{\theta}; \xi = 10, \alpha). \quad (4.6)$$

We expect to see a curve in the shape of a bowl that has its global minimum at a point $\alpha = 0.5$ or at least somewhere near. The results of the first part are represented in Figure 4.4 and

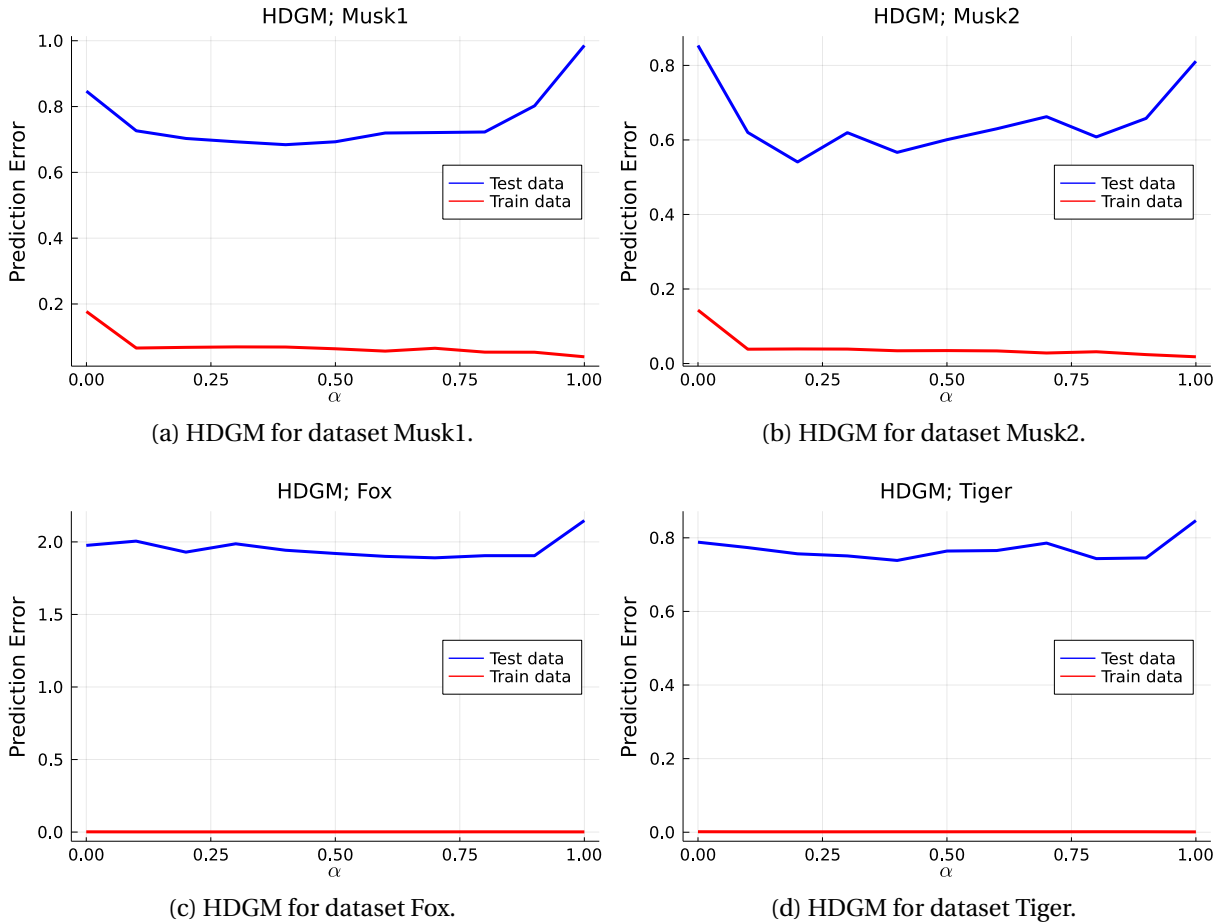


Figure 4.4: Dependence of the prediction error $\text{err}(\alpha)$ on the hyper-parameter α .

Table 4.2, where it can be seen that the addition of the generative term to the MIL loss function decreased the prediction error evaluated in all data sets. This improvement is considerable in Musk1 and Musk2 data sets, where a nice bowl can be observed.

Dataset	argminerr(α)	minerr(α)
Musk1	0.4	0.68
Musk2	0.2	0.54
Fox	0.7	1.89
Tiger	0.4	0.74

Table 4.2: Prediction error summary for the HDGM loss in case of $\xi = 10$.

In the Fox and Tiger data sets, the analysis does not reveal any significant differences between hybrid and discriminative training. Overall, these results suggest that the HDGM approach works moderately and that regularization in the form of a very simple generative term may improve the predictions. Unfortunately, the choice of $\alpha = 0.5$ was not confirmed as the best in our experiment; see the first column of Table 4.2.

Dependence on model complexity ξ

In the second part, the evaluation of the prediction error is approached using the complexity of the model. We fix $\alpha = 0.5$ and for model complexity, we select $\xi \in \{1, 2, 3, \dots, 20\}$. Similarly to the above, let the total hybrid loss calculated on the k^{th} split of the data be denoted by $\text{HDMG}_k(\hat{\theta}; \xi, \alpha = 0.5)$. Finally, the estimated prediction error in this case is defined by

$$\text{err}(\xi) = \frac{1}{r} \sum_{k=1}^r \text{HDMG}_k(\hat{\theta}; \xi, \alpha = 0.5). \quad (4.7)$$

In other words, the setup is the same as in 4.3.1, therefore, the results obtained will be added to Figure 4.3 and Table 4.1 for convenient comparison. Since we are interested in predictions for test data, the curves for train data will be omitted in this setting. The results are shown in Figure 4.5 and Table 4.3.

	Discriminative part only			HDGM; $\alpha = 0.5$		
	argminerr(ξ)	minerr(ξ)	err($\xi = 10$)	argminerr(ξ)	err(ξ)	err($\xi = 10$)
Musk1	2	0.70	0.97	6	0.64	0.69
Musk2	3	0.57	0.81	6	0.55	0.66
Fox	1	0.89	2.13	1	0.95	1.96
Tiger	1	0.56	0.82	2	0.58	0.80

Table 4.3: Comparison of prediction errors for HDGM $\alpha = 0.5$ and discriminative part only. Pay special attention to the last column $\text{err}(\xi = 10)$ in each approach.

Here is a situation very similar to the previous part of this experiment. The improvement in the prediction error for HDGM (green curve) is quite noticeable in the first two data sets, Musk1 and Musk2, while Fox and Tiger do not look so convincing. Furthermore, the number of random splits $r = 100$ is still needed to remove the noise from the prediction error. In general, it can be said that the HDGM approach leads to a small decrease in prediction error.

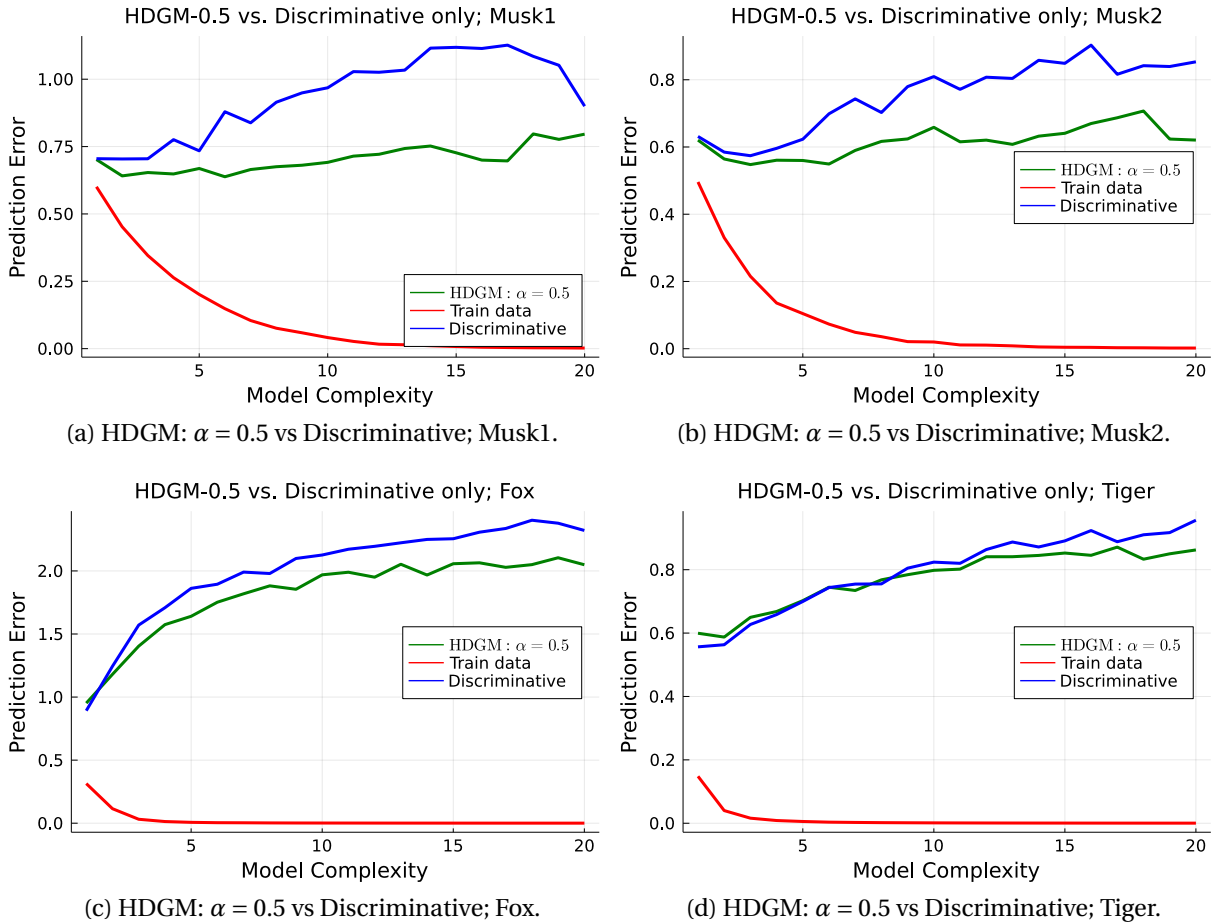


Figure 4.5: Comparison of prediction errors for HDGM and standard discriminative training.

4.4.2 Discussion

In ongoing experiments, we tested the possibility of improving the training of the MIL model using HDGM. The prediction error of these models was slightly reduced. However, prediction error is not a commonly used metric to measure the performance of a prediction model. Common tools are the ROC or PR curve and the corresponding AUC, as outlined in Section 1.3.1. However, the previous analysis revealed a large amount of noise in the results, and therefore it was necessary to average the prediction error. There is no point in plotting individual ROC or PR curves, but to average the AUC values of the model over r runs. Note that all data sets are fairly balanced in classes, and thus we employ AUC–ROC.

Due to the CV results 4.3, we select such MIL model complexity, where the model gives the best performance (for both approaches, D and HDGM with $\alpha = 0.5$) and calculate the mean of AUC–ROC over $r = 50$ training runs. Table 4.4 collects the results achieved with great correspondance to the previous results. Slight improvement can be observed in Musk1, Musk2, and Fox. However, the increase in AUC for Tiger has not been achieved. Note that for Musk1

	D	HDGM
Musk1	81.40±12.11 %	83.89 ±11.13%
Musk2	79.77±8.96%	82.78 ±7.38%
Tiger	91.22 ±2.34%	90.25±2.80%
Fox	54.43 ±3.77%	56.22±6.18%

Table 4.4: Average AUCs and st. deviations of individual data sets for both approaches with $r = 50$.

and Musk2 there is a rather high standard deviation. This can be explained by the small number of bags in these data sets.

4.5 Hybrid VAE for MIL

We will use a common assumption that instances \mathbf{x} in the bag b are independent, identically drawn samples from a distribution $p_f(\mathbf{x})$. Then, the likelihood of the bag is governed by the point process distribution [47], that is

$$p(b) = p_c(b) \prod_{j=1}^{|b|} p_f(\mathbf{x}_j)$$

where $p_c(b)$ is probability of cardinality of the bag. Due to the independence assumption, the maximum likelihood estimate of unknown parameter of $p_f(\mathbf{x})$ is identical to its estimation from a collection of instances from all bags [47]. Analogous result holds for the ELBO and thus the VAE of the instances can be trained from all instances. In the case of supervised VAE, the label associated with the instance is that of the bag. Implementation of the hybrid VAE is thus composed of two parts:

1. the generative model (VAE) is trained on the instances from all bags,
2. the discriminative model is trained using the standard MIL classifier.

Both of these objects have already been derived. The only thing left to do is to put them together using (3.25), replacing the classifier $q_\phi(y|\mathbf{x})$ with the standard MIL classifier. Note that such an arrangement means that the two losses combined in (3.25) may have very different scales, and the challenge is to tune the proportionality constant ν .

4.5.1 Setup and results

We now proceed to a more specific description of the individual parts of the hybrid VAE model for MIL.

Standard MIL classifier: Standard MIL classifier completely corresponds to the classifier described in Section 4.3, with $\xi = 10$.

Hybrid VAE: The hybrid VAE model is similar to that of 3.4.1. It consists of an encoder containing single dense layer with tanh non-linearity, then $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are also single dense layers with tanh non-linearity and f_θ is stacked with 2 dense layers with tanh non-linearity and identity as the output. This setting is subtly shallower than the previous hybrid VAE setting. Additionally, we need to switch SELU for tanh to suppress the outcoming values, as we expect an enormous scale of this loss.

In this experiment, we want to plot the dependence of AUC–ROC on the hyper–parameter ν similar to Figure 3.7, in order to find the best option for ν . Unfortunately, during the training of this model, the influence of stochasticity reappears again, and we get different results for different initializations. We reduce the effect of stochasticity by averaging over $r = 10$ runs. To balance both losses in (3.25), the value of $\nu = 10^{10}$ is approximately needed. Therefore, we start with $\nu = 1.25 \cdot 10^9$ and proceed by doubling this value. The following Table 4.5 summarizes the results.

	Musk1	Musk2	Fox	Tiger
ν	AUC[%]	AUC[%]	AUC[%]	AUC[%]
$1.25 \cdot 10^9$	92.46±9.51	89.01±7.47	65.98±2.10	82.14±1.89
$2.50 \cdot 10^9$	83.24±12.54	92.08±8.22	63.86±4.25	80.96±1.86
$5.00 \cdot 10^9$	88.18±9.37	94.89±4.21	66.11±5.36	81.51±2.46
$1.00 \cdot 10^{10}$	90.78±9.45	92.97±5.23	63.19±4.24	80.60±1.74
$2.00 \cdot 10^{10}$	91.16±5.34	89.01±8.14	63.98±3.06	82.99±2.14
$4.00 \cdot 10^{10}$	87.99±9.39	87.08±10.93	64.54±3.77	82.70±2.34
$8.00 \cdot 10^{10}$	90.91±8.23	93.80±4.46	64.56±4.73	81.44±2.15
$1.60 \cdot 10^{11}$	91.04±8.71	95.47±5.84	63.70±5.11	81.73±2.23
$3.20 \cdot 10^{11}$	85.65±10.33	92.19±5.88	65.61±3.81	81.99±1.71
$6.40 \cdot 10^{11}$	90.71±4.90	83.59±13.59	66.14±4.16	81.64±1.58

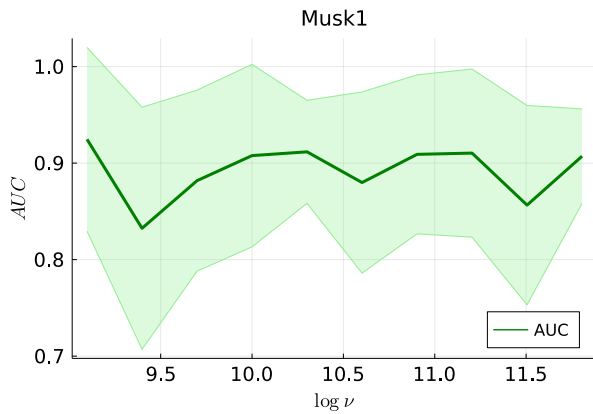
Table 4.5: Average AUCs and st. deviations of individual data sets for hybrid VAE over $r = 10$ runs.

Apparently, finding the optimal value for ν is very difficult, as the results behave very unpredictably 4.6. Hybrid VAE for MIL problems does not follow the trend of 3.7 as we had hoped. Compared to a purely discriminative approach 4.4, noticeably higher AUC values were obtained for Musk1 and Musk2. This indicates the great potential of this method and confirms its functionality. A slight improvement can also be seen on the Fox, but on the Tiger there is a significant deterioration. Unfortunately, standard deviations are still very high, especially for Musk1 and Musk2.

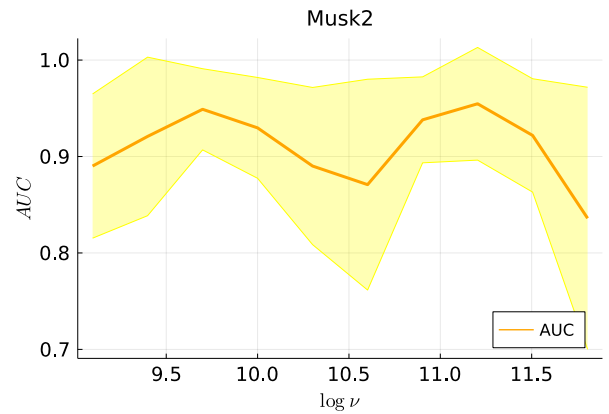
In general, we can say that, compared to standard discriminative training, there has been a considerable improvement.

4.5.2 Discussion

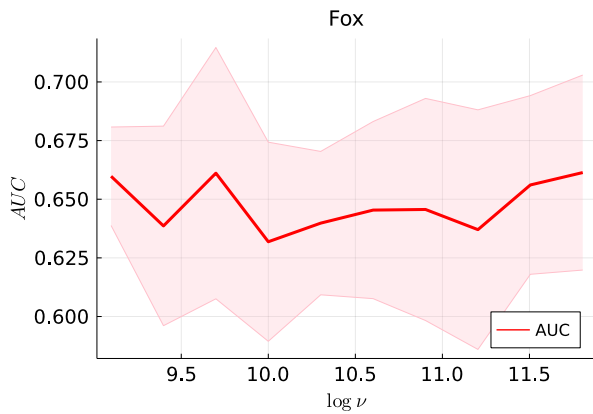
Let us try in this discussion to dissect the possible causes of high standard deviations. The most probable variant seems to be the excessive complexity of our hybrid VAE model. There are only a few layers, but these layers are, in fact, dense layers. This means that there



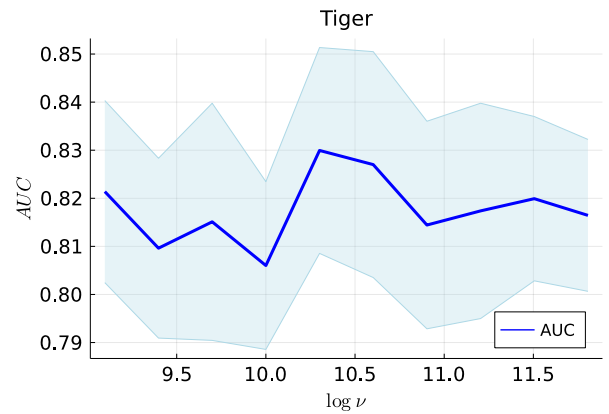
(a) Average AUC of Hybrid VAE for Musk1.



(b) Average AUC of Hybrid VAE for Musk2.



(c) Average AUC of Hybrid VAE for Fox.



(d) Average AUC of Hybrid VAE for Tiger.

Figure 4.6: Comparison of the dependencies of the average AUC on $\log \nu$ with st. deviation for hybrid VAE.

is a large number of parameters to train, and thus any type of pruning could improve the performance of the model.

Another possible cause is the small amount of data. We have 4 data sets, but they are not very large. Each of them has hundreds of bags and a few thousand instances at maximum. In addition, these data are also divided into training and testing sets, making the data more scarce.

There are, of course, other methods for dealing with high standard deviation, which together with the two already mentioned, could be the basis for further research.

Conclusion

The aim of this thesis was to provide an insight into the hybrid training of set data and compare the properties with the discriminative training of set data. In total, two hybrid training options were presented and experimentally tested on real data.

Before the actual application of these methods, we were concerned with discriminative and generative modeling separately. We summarized the principles of discriminative learning in classification problems and revealed its connection with KL distance. Generative modeling was primarily dealing with VAE, where individual theoretical aspects were explained thoroughly. The important component for hybrid modeling is not the vanilla VAE, but its extended version, called SSVAE, which turns out to be the key for our purposes. In addition, part of the generative modeling was noise-contrastive estimation, which was presented in a short overview. Once the technicalities of these approaches have been resolved, we performed simple experiments to obtain a full clarification of these methods. The next research objective was hybrid modeling. The first method was derived on the basis of energy-based models and contrastive learning, where the standard cross-entropy loss is extended by the contrastive loss to produce an advanced, hybrid loss. Thus, models are consequently trained using this hybrid loss. The second method is the hybrid VAE, which we have modified on the basis of the SSVAE. Since SSVAE is a semi-supervised method, the modification mainly concerned the conversion to a fully supervised form, which was experimentally verified on simple synthetic 2-Moons data. We call this modification hybrid VAE. The continuation of this thesis is about the novel machine learning paradigm called multiple instance learning, which copes with set data. We introduced basic concepts and principles such as embedded-space and its way of training.

Using the Mill.jl library, we tested the standard discriminative learning of the MIL model by cross-validation and AUC. There was a large amount of noise in the results, and it was necessary to compare the averages. Then, we applied a hybrid approach based on constrative learning. According to the averages of prediction errors and AUCs, there is a slight improvement, but unfortunately standard deviations are too high, and therefore we cannot claim this improvement to be that significant. Even in the second method, the hybrid VAE, we encountered considerable noise in the results. On this impulse, we had to average the resulting AUCs over more training runs. Furthermore, we trained the model with different values of proportionality hyper-parameter in order to find the best setting. Due to this, an initially very simple task became very expensive to compute. The best setting for the hyper-parameter could not be found, however, three of the four data sets showed a significant increase in AUC

compared to the standard MIL model. This achievement indicates the great potential of this approach, which can be further researched.

In general, we have been able to improve the standard discriminative training of set data using the two hybrid methods presented.

Bibliography

- [1] D. Yu, L. Deng, *Automatic Speech Recognition*. Berlin: Springer, 2016.
- [2] Y. Zhang, *Progress and challenges in protein structure prediction*. *Current opinion in structural biology*, 2008, 18.3: 342-348.
- [3] B. Kuhlman, P. Bradley, *Advances in protein structure prediction and design*. *Nature Reviews Molecular Cell Biology*, 2019, 20.11: 681-697.
- [4] T. Pevny, P. Somol, *Discriminative models for multi-instance problems with tree structure*. In 'Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security', 2016, 83-91.
- [5] T. Pevny, M. Grill, M. Rehak, *Reducing false positives of network anomaly detection by local adaptive multivariate smoothing*. *Journal of Computer and System Sciences* 83.1, 2017, 43-57.
- [6] J. Stiborek, T. Pevny, M. Rehak, *Multiple instance learning for malware classification*. *Expert Systems with Applications* 93, 2018, 346-357.
- [7] G. Pang, et al., *Deep learning for anomaly detection: A review*. *ACM Computing Surveys (CSUR)*, 2021, 54.2: 1-38.
- [8] T.G. Dietterich, R.H. Lathrop, T. Lozano-Pérez, *Solving the multiple instance problem with axis-parallel rectangles*. *Artificial intelligence*, 1997, 89.1-2: 31-71.
- [9] J. Wu, S. Pan, X. Zhu, C. Zhang, X. Wu, *Multi-instance learning with discriminative bag mapping*. *IEEE Transactions on Knowledge and Data Engineering*, 30(6), 2018, 1065-1080.
- [10] S. J. Yang, Y. Jiang, Z. H. Zhou, *Multi-instance multi-label learning with weak label*. In 'Proceedings of the Twenty-Third international joint conference on Artificial Intelligence', 2013. p. 1862-1868.
- [11] S. Mandlik, *Mapping the Internet: Modelling Entity Interactions in Complex Heterogeneous Networks*. arXiv preprint arXiv:2104.09650.
- [12] P. Khosla, et al., *Supervised contrastive learning*. *Advances in Neural Information Processing Systems*, 2020, 33: 18661-18673.

- [13] A. v. d. Oord, Y. Li, O. Vinyals, *Representation learning with contrastive predictive coding*. arXiv preprint arXiv:1807.03748, 2018.
- [14] H. Liu, P. Abbeel, *Hybrid discriminative-generative training via contrastive learning*. arXiv preprint arXiv:2007.09070, 2020.
- [15] B. Paige, et al., *Learning disentangled representations with semi-supervised deep generative models*. Advances in neural information processing systems, 2017, 30.
- [16] Y. Li, et al., *Disentangled variational auto-encoder for semi-supervised learning*. Information Sciences, 2019, 482: 73-85.
- [17] W. Qian, F. Lauri, F. Gechter, *Supervised and semi-supervised deep probabilistic models for indoor positioning problems*. Neurocomputing, 2021, 435: 228-238.
- [18] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*. arXiv preprint arXiv:1312.6114, 2013.
- [19] C. P. Burges, I. Higgins, et al., *Understanding disentangling in β -VAE* arXiv preprint arXiv:1804.03599, 2018.
- [20] J. Bezanson, et al., *A fast dynamic language for technical computing*. ArXiv12095145, 2012.
- [21] T. Pevny, S. Mandlik, *Mill.jl framework: a flexible library for (hierarchical) multi-instance learning* [on-line] Available from: <https://github.com/CTUAvastLab/Mill.jl>. Accessed 30 April 2020.
- [22] S. Mandlik, M. Racinsky, V. Lisy, T. Pevny, *Mill.jl and JsonGrinder.jl: automated differentiable feature extraction for learning from raw JSON data*. arXiv preprint arXiv:2105.09107, 2021.
- [23] M. Jordan, A. Ng, *On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes*. Advances in neural information processing systems, 2001, 14.
- [24] M. Talabis, R. McPherson, I. Miyamoto, J. Martin and D. Kaye, *Information Security Analytics*. Syngress, 2015.
- [25] S.-S. Shai, B.-D. Shai, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [26] W. Gratwohl, K. C. Wang, J. H. Jacobsen, *Your classifier is secretly an energy based model and you should treat it like one*. arXiv preprint arXiv:1912.03263, 2019.
- [27] Y. LeCun, S. Chopra, R. Hadsell, et al., *A tutorial on energy-based learning*. Predicting structured data, 2006, 1.0.
- [28] Y. Song, D. P. Kingma, *How to train your energy-based models*. arXiv preprint arXiv:2101.03288, 2021.

- [29] J. Kelly, R. Zemel, W. Grathwohl, *Directly Training Joint Energy-Based Models for Conditional Synthesis and Calibrated Prediction of Multi-Attribute Data*. arXiv preprint arXiv:2108.04227, 2021.
- [30] C. M. Bishop, N. M. Nasrabadi, *Pattern recognition and machine learning*. New York: Springer, 2006.
- [31] V. Smidl, *The Variational Bayes Approach in Signal processing*. PhD Thesis. Trinity College Dublin. 2004.
- [32] S. K. Ng, T. Krishnan, G. J. McLachlan, *Handbook of computational statistics*. The EM algorithm. Springer, Berlin, Heidelberg. 2012, 139-172.
- [33] H. Jeffrey, *An invariant form for the prior probability in estimation problems*. In 'Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences', 1946, 1–9.
- [34] J. Friedman, T. Hastie, R. Tibshirani, *The elements of statistical learning*. New York: Springer, 2009.
- [35] D. Commenges, *Information Theory and Statistics: an overview*. ArXiv preprint arXiv:1511.00860, 2015, 1–22.
- [36] L. Mao, *Cross Entropy, KL Divergence, and Maximum Likelihood Estimation*. [on–line] Available from: <https://leimao.github.io/blog/Cross-Entropy-KL-Divergence-MLE/>. Accessed 30 April 2020.
- [37] P. D. Kingma, J. Ba *A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.
- [38] G. W. Gundersen: *The Reparameterization Trick*. [on–line] Available from: <https://gregorygundersen.com/blog/2018/04/29/reparameterization/>. Accessed 30 April 2020.
- [39] J. Brownlee, *How to Handle Big-p, Little-n ($p \gg n$) in Machine Learning*. (2020). [on–line]. Available from: <https://machinelearningmastery.com/how-to-handle-big-p-little-n-p-n-in-machine-learning/>. Accessed 30 April 2020.
- [40] F. Pedregosa, G. Varoquaux, et al. *Scikit-learn: Machine Learning in Python*. [on–line] Available from: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html. Accessed 30 April 2020.
- [41] A. Krizhevsky, et al., *Learning multiple layers of features from tiny images*. 2009
- [42] J. Bures, *Generativní modely dat popsaných stromovou strukturou..* Bachelor's Thesis. Czech Technical University in Prague. 2019.

- [43] M. Zhuang, M. Collins, *Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency*. arXiv preprint arXiv:1809.01812, 2018.
- [44] M. Gutmann, A. Hyvärinen, *Noise-contrastive estimation: A new estimation principle for unnormalized statistical models*. In 'Proceedings of the thirteenth international conference on artificial intelligence and statistics', JMLR Workshop and Conference Proceedings, 2010, 297-304.
- [45] D. A. Levin, Y. Peres, *Markov chains and mixing times*. American Mathematical Soc., 2017
- [46] J. Davis, M. Goadrich, *The relationship between Precision-Recall and ROC curves*. In proceedings of the 23rd international conference on Machine learning. 2006. 233-240.
- [47] B. N. Vo et. al, *Model-based learning for point pattern data*. Pattern Recognition, 2018, 84: 136-151.

A

Computational formulas

A.1 Solution of $D_{\text{KL}}(\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \sigma^2 \mathbb{I}_P) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbb{I}_P))$

In the VAE section, the ELBO (2.16) is derived and subsequently optimized. One of the ELBO expressions is the KL distance mentioned above. For two multivariate Gaussian distributions, we have a KL distance analytical solution. The complete calculation is given here. First, recall that the PDF for a multivariate Gaussian distribution in \mathbb{R}^P with mean $\boldsymbol{\mu} \in \mathbb{R}^P$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{P \times P}$ is defined as

$$\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^P \det \boldsymbol{\Sigma}}} \exp\left(-\frac{1}{2}(\mathbf{z} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{z} - \boldsymbol{\mu})\right).$$

Then, the KL distance for two different multivariate Gaussian distributions can be computed as

$$\begin{aligned}
D_{\text{KL}}(\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \parallel \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)) &= \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)} [\log \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) - \log \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)] \\
&= \frac{1}{2} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)} \left[-\log \det \boldsymbol{\Sigma}_1 - (\mathbf{z} - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}_1^{-1} (\mathbf{z} - \boldsymbol{\mu}_1) + \log \det \boldsymbol{\Sigma}_2 + (\mathbf{z} - \boldsymbol{\mu}_2)^\top \boldsymbol{\Sigma}_2^{-1} (\mathbf{z} - \boldsymbol{\mu}_2) \right] \\
&= \frac{1}{2} \log \frac{\det \boldsymbol{\Sigma}_2}{\det \boldsymbol{\Sigma}_1} + \frac{1}{2} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)} \left[-(\mathbf{z} - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}_1^{-1} (\mathbf{z} - \boldsymbol{\mu}_1) + (\mathbf{z} - \boldsymbol{\mu}_2)^\top \boldsymbol{\Sigma}_2^{-1} (\mathbf{z} - \boldsymbol{\mu}_2) \right] \\
&= \frac{1}{2} \log \frac{\det \boldsymbol{\Sigma}_2}{\det \boldsymbol{\Sigma}_1} + \frac{1}{2} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)} \left[-\text{Tr} \left(\boldsymbol{\Sigma}_1^{-1} (\mathbf{z} - \boldsymbol{\mu}_1) (\mathbf{z} - \boldsymbol{\mu}_1)^\top \right) + \text{Tr} \left(\boldsymbol{\Sigma}_2^{-1} (\mathbf{z} - \boldsymbol{\mu}_2) (\mathbf{z} - \boldsymbol{\mu}_2)^\top \right) \right] \\
&= \frac{1}{2} \log \frac{\det \boldsymbol{\Sigma}_2}{\det \boldsymbol{\Sigma}_1} + \frac{1}{2} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)} \left[-\text{Tr} \left(\boldsymbol{\Sigma}_1^{-1} \boldsymbol{\Sigma}_1 \right) + \text{Tr} \left(\boldsymbol{\Sigma}_2^{-1} \left(\mathbf{z} \mathbf{z}^\top - 2 \mathbf{z} \boldsymbol{\mu}_2^\top + \boldsymbol{\mu}_2 \boldsymbol{\mu}_2^\top \right) \right) \right] \\
&= \frac{1}{2} \log \frac{\det \boldsymbol{\Sigma}_2}{\det \boldsymbol{\Sigma}_1} - \frac{1}{2} P + \frac{1}{2} \text{Tr} \left(\boldsymbol{\Sigma}_2^{-1} \left(\boldsymbol{\Sigma}_1 + \boldsymbol{\mu}_1 \boldsymbol{\mu}_1^\top - 2 \boldsymbol{\mu}_2 \boldsymbol{\mu}_1^\top + \boldsymbol{\mu}_2 \boldsymbol{\mu}_2^\top \right) \right) \\
&= \frac{1}{2} \left(\log \frac{\det \boldsymbol{\Sigma}_2}{\det \boldsymbol{\Sigma}_1} - P + \text{Tr} \left(\boldsymbol{\Sigma}_2^{-1} \boldsymbol{\Sigma}_1 \right) + \text{Tr} \left(\boldsymbol{\mu}_1^\top \boldsymbol{\Sigma}_2^{-1} \boldsymbol{\mu}_1 - 2 \boldsymbol{\mu}_1^\top \boldsymbol{\Sigma}_2^{-1} \boldsymbol{\mu}_2 + \boldsymbol{\mu}_2^\top \boldsymbol{\Sigma}_2^{-1} \boldsymbol{\mu}_2 \right) \right) \\
&= \frac{1}{2} \left(\log \frac{\det \boldsymbol{\Sigma}_2}{\det \boldsymbol{\Sigma}_1} - P + \text{Tr} \left(\boldsymbol{\Sigma}_2^{-1} \boldsymbol{\Sigma}_1 \right) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}_2^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \right),
\end{aligned}$$

where $\text{Tr}(\cdot)$ denotes the trace of a matrix. Finally, substituting $\boldsymbol{\mu}_1 = \boldsymbol{\mu}$, $\boldsymbol{\Sigma}_1 = \sigma^2 \mathbb{I}_P$, $\boldsymbol{\mu}_2 = \mathbf{0}$ and $\boldsymbol{\Sigma}_2 = \mathbb{I}_P$ gives

$$\begin{aligned}
D_{\text{KL}}(\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \sigma^2 \mathbb{I}_P) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbb{I}_P)) &= \frac{1}{2} \left(\log \frac{\det \mathbb{I}_P}{\det \sigma^2 \mathbb{I}_P} - P + \text{Tr}(\sigma^2 \mathbb{I}_P \cdot \mathbb{I}_P) + \boldsymbol{\mu}^\top \boldsymbol{\mu} \right) \\
&= \frac{1}{2} \left(-P \log \sigma^2 - P + \sum_{j=1}^P \sigma^2 + \sum_{j=1}^P \mu_j^2 \right) \\
&= \frac{1}{2} \sum_{j=1}^P \left(-1 - \log \sigma^2 + \sigma^2 + \mu_j^2 \right).
\end{aligned}$$

A.2 Derivation of $\mathbb{E}_{p_{\theta}(\mathbf{x})}[-\nabla_{\theta} E_{\theta}(\mathbf{x})]$

In the third chapter, we introduced EBM and its way of training. The necessary step is to compute the following gradient

$$\begin{aligned}\nabla_{\theta} \log Z(\theta) &= \nabla_{\theta} \log \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \\ &= \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \nabla_{\theta} \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \\ &= \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \nabla_{\theta} \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \\ &= \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \exp(-E_{\theta}(\mathbf{x})) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \\ &= \int \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \exp(-E_{\theta}(\mathbf{x})) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \\ &= \int \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z(\theta)} (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \\ &= \int p_{\theta}(\mathbf{x}) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \\ &= \mathbb{E}_{p_{\theta}(\mathbf{x})}[-\nabla_{\theta} E_{\theta}(\mathbf{x})].\end{aligned}$$