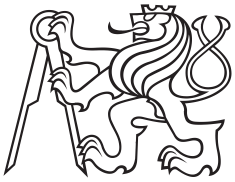
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Semi-Supervised Learning for Spatio-Temporal Segmentation of Satellite Images

Antonín Hruška

Supervisor: doc. Boris Flach, Dr. rer. nat. habil.
Study program: Cybernetics and Robotics
May 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Hruška Antonín** Personal ID number: **474562**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Semi-Supervised Learning for Spatio-Temporal Segmentation of Satellite Images

Master’s thesis title in Czech:

Částečné učení s učitelem pro časově-prostorovou segmentaci satelitních snímků

Guidelines:

The task considers Earth observation and in particular land-cover monitoring from time series of multispectral satellite images. The goal is to develop a semi-supervised learning approach for spatio-temporal land-cover segmentation of multispectral satellite images with focus on monititoring changes in national parks. It is essential that the novel learning approach can learn from partial annotations in presence of partially missing measurements (cloud cover, snow cover).
The particular tasks for the diploma thesis are:
1. Survey suitable state of the art deep learning approaches for semi-supervised learning and chose one as baseline.
2. Develop a semi-supervised learning approach for spatio-temporal segmentation that combines spatio-temporal U-Nets (3D CNN) with variational autoencoders.
3. Implement the baseline approach and the novel approach in PyTorch.
4. Validate the generative capabilities of the novel approach (e.g. inpainting of missing data).
5. Validate the segmentation accuracy of the novel approach and compare it with the chosen state of the art baseline on time series of multi-spectral satellite data (Sentinel2/Landsat) for the national park Bohemian Switzerland along with partial annotations from an expert.

Bibliography / sources:

[1] Kingma, Diederik P. and Welling, Max, An Introduction to Variational Autoencoders, Foundations and Trends in Machine Learning: Vol. 12 (2019), eprint arXiv:1906.02691
[2] Alexander Shekhovtsov, Dmitrij Schlesinger, Boris Flach, VAE Approximation Error: ELBO and Exponential Families, Intl. Conf. on Learning Representations (ICLR), 2022
[3] Olaf Ronneberger, Philipp Fischer, Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, Intl. Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2015
[4] Casper Kaae Sønderby et al., Ladder Variational Autoencoders, Intl. Conf. on Neural Information Processing Systems (NIPS), 2016
[5] Yassine Ouali, Céline Hudelot, Myriam Tami, An Overview of Deep Semi-Supervised Learning, Arxiv, 2020

Name and workplace of master’s thesis supervisor:

doc. Boris Flach, Dr. rer. nat. habil. Machine Learning FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **02.02.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

doc. Boris Flach, Dr. rer. nat. habil. Supervisor's signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my deepest gratitude to Doc. Boris Flach, Dr. rer. nat. habil., without whom this endeavor would not have been possible. I am immensely thankful to him for the professional guidance he provided me with, as well as his kind words and moral support during the challenging times. I would also like to extend my heartfelt thanks to all the teachers, lecturers, and individuals who generously dedicated their time, effort, and passion to teach me new things and propel me forward.

I am also grateful for the access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765.
The support provided by the "Research Center for Informatics" through this project is sincerely acknowledged and appreciated.

In addition, I would like to express my gratitude to my family, partner, and close friends who have been unwavering in their support throughout my studies and have played a significant role in helping me complete this thesis. Their encouragement and assistance have been invaluable to me, and I am truly grateful for their presence in my life.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In particular, I declare that I have not used natural language processing models to generate the ideas and themes used in the thesis. Nevertheless, I have used such tools (Grammarly, DeepL) to correct any grammatical or syntactical errors or to translate the original Czech version of the text into English and to improve the wording of such text. I have also used the Chat-GPT models for creating code snippets in both Python and Latex, as well as having a second reader of the thesis.

In Prague, 26th May, 2023

Abstract

This thesis investigates the applicability of semi-supervised machine learning algorithms for forest land cover segmentation in satellite images. Instead of directly evaluating satellite imagery, the CityScape dataset is utilized for result verification and reproducibility. We review the semi-supervised machine learning algorithms, introduce MixMatch as a reference method and a novel algorithm based on symmetric learning of variational autoencoders, along with a self-contained introduction to variational autoencoders. The research aims to compare the segmentation potential and capabilities of both the new and the reference algorithms using a U-net network-based model architecture. The results show that MixMatch effectively enhances segmentation performance to supervised baseline, particularly in scenarios with limited labeled data. Although the symmetrical learning does not exceed the supervised baseline, the experiments still serve as a proof of concept, highlighting areas for further investigation.

Keywords: Image segmentation, SSL, VAE, hiearchical VAE, ladder VAE,symmetric equilibrium learning, MixMatch, CityScape

Supervisor: doc. Boris Flach, Dr. rer. nat. habil.

Abstrakt

Tato práce zkoumá použitelnost algoritmů částečného strojového učení s učitelem pro segmentaci lesních ploch ze satelitních snímků. Namísto přímého vyhodnocování na satelitních snímcích budeme používat CityScape dataset pro ověření výsledků a reprodukovatelnost. Poskytujeme přehled algoritmů v oblasti částečného strojového učení a představujeme vyhodnocované algoritmy: referenční MixMatch a nový algoritmus založený na symetrickém učení variačních autoenkodérů. Uvádíme také náhled do variačních autoenkodérů, zahrnující jejich teoretické základy a rozšíření, aby čtenář porozuměl základním principům. Hlavním cílem je porovnat segmentační schopnosti obou algoritmů. V obou případech využíváme stejnou architekturu modelů založenou na U-net síti. MixMatch účinně zvyšuje přesnost segmentace, zejména ve scénářích s velmi malým množstvím dostupných dat. Přestože symetrické učení hierarchického autoenkodéru nepřekonává základní model s učitelem, experimenty prokazují použitelnost symetrického učení a zároveň identifikují oblasti pro zlepšení.

Klíčová slova: Segmentace obrazu, SSL, hiearchický VAE, ladder VAE, symmetric equilibrium learning, MixMatch, CityScape

Překlad názvu: Částečné učení s učitelem pro časově-prostorovou segmentaci satelitních snímků

Contents

Figures

Tables

Chapter 1

Introduction

Satellite imagery provides a wealth of information about our planet and has become a standard tool for monitoring, predicting and understanding the change in vegetation, agriculture and human environmental impact. With advances in satellite technology, it is now possible to collect considerably large amounts of high-resolution multispectral images over time, enabling researchers to perform complex spatiotemporal analyses of these datasets. However, the magnitude of the raw data makes it challenging to process and analyze it effectively. Moreover, remote sensing still faces further challenges, which are rare in other areas of computer vision. Those are, in particular, partial measurements (i.e. cloud cover, electromagnetic (EM) interference), geolocation, different quality of measurement (spatial resolution, different EM bands) and calibration (issue of atmospheric reflectance).

From a machine learning perspective, the main challenges are considerable amounts of unannotated data and partially missing measurements. To address the first, semi-supervised learning (SSL) [5] has shown promise in leveraging the abundance of unlabeled data to improve model performance. SSL algorithms are designed to learn from labeled and unlabeled data, using the labeled data to guide the learning process and the unlabeled data to regularize the model. Regarding the issue of partially missing measurements, generative models, such as variational autoencoders (VAEs) [18], have shown the potential to fill in the missing data gaps. By learning a generative model of the data distribution, VAEs can be used to impute missing values in a dataset, making it possible to fully utilize the available data.

The thesis topic is motivated by the real-world problem of segmenting the satellite imagery of a national park to monitor and predict its forest development. The forest development prediction could allow national park rangers to respond proactively to protect forest vegetation and thus improve national park preservation. The thesis aims to create an approach that could be used in data preprocessing to obtain segmentation for many images and generate inpainting for partially missing measurements. The project is

funded by the European Space Agency (ESA) [*].

The authors of [40] proposed a novel SSL algorithm that employs symmetrical learning with hierarchical VAEs, capable of handling exponential families of distribution, not just multivariate Gaussians for the latent variable. We adapt this algorithm to the land-cover segmentation task, where the learning process from only partially available inputs is necessary. This unavailability issue is due to missing measurements, which are mainly caused by clouds or snow cover. Our approach combines a reasonably sized model inspired by the U-net architecture and its skip connections[37] with hierarchical VAEs, Such an approach would enable us to obtain a latent space distribution that represents the desired segmentation. Additionally, this setup will allow us to generate both segmentation from images and images from segmentation.

We compare our approach with the MixMatch [2] algorithm, which unifies consistency regularization methods with proxy-labeling methods [33] and is the cornerstone for other algorithms such as ReMixMatch [3] and FixMatch [42]. To our knowledge, the MixMatch algorithm has only been used for classification tasks (CIFAR10, SVHN). We adapt it so that it applies to the segmentation task. To verify its suitability and compare it with our method, we use the CityScape dataset and standard metrics used in segmentation, such as plain accuracy, intersection over union (IoU), and others. We also validate the generative capabilities of the novel approach (e.g., inpainting of missing data).

We briefly introduce SSL and its methods in chapter 2. We discuss the main ideas and cornerstones of MixMatch and the novel symmetrical equilibrium learning, namely consistency regularization with proxy labelling methods and VAE framework, respectively. This chapter also provides the hierarchical models (HVAE) and ladder variational autoencoders (LVAE) as a foundation for the novel algorithm. The segmentation task, dataset, metrics and models architecture are described in chapter 3. The experiments and results are available in chapter 4 with a summary of the main findings and contributions in 5. We make all code publicly available [†].

[*] `https://eo4society.esa.int/projects/spatiotemporal-sen2vhr/`
[†] At `https://github.com/hruskan1/SSL-diploma-thesis`

Chapter 2

State of the art of SSL

In this chapter, we will present a brief introduction to semi-supervised learning in section 2.1. We will discuss the key concepts and principles underlying semi-supervised learning and its significance in machine learning. Furthermore, we will classify the algorithms used in semi-supervised learning into various groups based on the different ideas and paradigms they employ.

Next, we will delve deeper into two specific approaches: MixMatch and Variational Autoencoders (VAEs). Section 2.2 will provide an in-depth introduction to MixMatch, while section 2.3 will focus on the basics of Variational Autoencoders, introducing the topic, its achievements, and recognized shortcomings. Following that, in section 2.4, we will explore advanced variants of VAEs that partially overcome the aforementioned shortcomings and represent the state-of-the-art in the field of VAE.

These sections serve as a foundation for comprehending a novel algorithm based on symmetric equilibrium learning in VAEs, which will be introduced in section 2.6. Additionally, section 2.5 provides an introduction to the family of exponential distributions, which are extensively used in VAE and its advanced variants.

2.1 SSL introduction

Semi-Supervised Learning (SSL) is an essential subfield of Machine Learning (ML) that aims to improve model performance by leveraging both labeled and unlabeled data. In many real-world scenarios, obtaining labeled data is expensive and time-consuming, whereas unlabeled data is abundant and relatively easy to acquire. Therefore, SSL algorithms seek to learn from both labeled and unlabeled data to improve model generalization and achieve higher accuracy. Unlike supervised learning, where models rely

entirely on labeled data, SSL algorithms use a small amount of labeled data to guide the model's learning process while exploiting the vast amounts of unlabeled data to extract useful features and improve its predictions. In recent years, there has been a growing interest in developing novel SSL algorithms that can tackle complex problems and achieve state-of-the-art performance, making SSL a rapidly evolving field of research.

Typically the training dataset $\mathcal{D}$ can be divided into two subsets $\mathcal{D} = \mathcal{D}_l \cup \mathcal{D}_u$:

$$\mathcal{D}_l = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}, \quad \mathcal{D}_u = \{(\mathbf{x}_{l+1}), \ldots, (\mathbf{x}_u)\},$$

In this *standard* setting, SSL can be viewed as supervised learning, where the *unlabeled* data provide additional information on the underlying distribution of the examples $\mathbf{x}$. We will refer to this setting in this thesis. However, there are also different formulations of the SSL problem, such as *SSL with constraints* and others [5, p. 1].

> *"A natural question arises: is semi-supervised learning meaningful? More precisely: in comparison with a supervised algorithm that uses only labeled data, can one hope to have a more accurate prediction by taking into account the unlabeled points? ... Yes, however there is an important prerequisite: that the distribution of examples, which the unlabeled data will help elucidate, be relevant for the classification problem. ... One should thus not be too surprised that for semi-supervised learning to work, certain assumptions will have to hold."* – Chappele et *al* [5, p. 4]

2.1.1 Assumptions in SSL

As stated in the quote above, several assumptions are necessary for SSL algorithms to work [5, p. 5]. Some of them are well-known from unsupervised learning:

- **The Smoothness Assumption**: *If two points $\mathbf{x}_1$ and $\mathbf{x}_2$ lies nearby in high-density region, then the desired outputs $\mathbf{y}_1$ and $\mathbf{y}_2$ should be similar.* This assumption generalizes the supervised learning assumption, where the same holds if $\mathbf{x}_1$ and $\mathbf{x}_2$ *are close* (not necessarily in the high-density region). Due to transitivity, the assumption clusters the data into high-density clusters, and many clusters can share the same output value.
- **The Cluster Assumption**: *Points in one cluster are likely to be of the same class*, or in other words, *the decision boundary should be located in the low-density region.* This assumption is a special case of the previously mentioned assumption, as clusters are often considered regions with a high data density. However, it is independently presented as it is easier to understand and has motivated several unsupervised algorithms such as K-means and others.

- **The Manifold Assumption**: *The data lie along low-dimensional latent manifolds inside that high-dimensional space.* This assumption tries to overcome the *curse of dimensionality*. Simply put, as the dimension grows, the sparsity of data increases, which makes clustering impossible, as there are no clusters to be found. If the manifold assumption holds, we can search for a mapping into such a low-dimensional manifold in which clustering is possible. Several unsupervised algorithms utilize this assumption, such as PCA, MDS, ISOMAP, and t-SNE.

2.1.2 SSL methods

SSL algorithms can be categorized into the following groups based on their motivation, making it easier to navigate and understand them [33]:

- **Consistency Regularization**: According to the smoothness assumption, if the input $\mathbf{x}$ and its perturbed version $\tilde{\mathbf{x}}$, are close to each other, their corresponding outputs, $\mathbf{y}$ and $\tilde{\mathbf{y}}$, should also be similar. By minimizing the distance between the model outputs$f_\theta(\mathbf{x})$ and $f_\theta(\tilde{\mathbf{x}})$, where the distance can be measured using a variety of techniques, such as mean square error (MSE) or Kullback-Leibler (KL) divergence, we can train the model to make consistent predictions on both the original and perturbed inputs [23, 39]. We can also use other divergence techniques, such as Jeffreys divergence or Jensen-Shanon (JSD) divergence, which have the advantage of being symmetric with respect to the inputs. This requirement is transformed into an expanded loss objective with a new term for consistency regularization:

$$\mathcal{L} = \sum_{\mathbf{x},y \in \mathcal{D}_l} l(\mathbf{x}, y) + \sum_{\mathbf{x} \in \mathcal{D}_u} d(f_\theta(\mathbf{x}), f_\theta(\tilde{\mathbf{x}}))$$

  where $l(\mathbf{x}, y)$ corresponds to the standard supervised loss for given task and $d(\cdot, \cdot)$ corresponds to the one of the mentioned metrics.

- **Proxy-label Methods**: These methods are based on an (iterative) scheme, where the model generates the proxy label on unlabeled data (or parts thereof) using the prediction function itself or some variant of it [24]. These labels are then taken as targets for the next iteration. Although the proxy labels are often weak, the methods can provide additional information for training. We can divide these methods into two groups: Self-training, where the model produces the proxy label itself, and *multi-view learning*, where the proxy labels are produced by (multiple) models trained on different views of the training data. The idea of multi-view learning is exactly the same as bootstrapping.

- **Generative Models**: The *generative* models try to model the feature density $p(\mathbf{x})$ or even joint density $p(\mathbf{x}, y)$ by some unsupervised learning procedure (i.e. maximum likelihood estimation (MLE)). An inference can be then obtained by Bayes inference rule (for a given loss l):

$$f^\star(\mathbf{x}) = \operatorname*{argmin}_{y' \in \mathcal{Y}} \sum_{y \in \mathcal{Y}} (y|\mathbf{x}) l(y, y')$$

where conditional probability $p(y|x)$ can be obtained through Bayes theorem:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}, y)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|y)p(y)}{\int_{\mathcal{Y}} p(\mathbf{x}|y)p(y)\mathrm{d}y}$$

After training a model, we can use it to generate new samples from a *modelled* distribution $p_\theta(\mathbf{x})$ at any time. This procedure allows us to obtain features that were not present in the original training set, but the quality of these new features depends on how closely our model approximates the true underlying distribution $p^\star(\mathbf{x})$ represented by the training set distribution $p_{\mathcal{D}}(\mathbf{x})$, which is also known as the *evidence*. Therefore, the quality of the generated samples depends on the accuracy of the model's approximation to the true distribution.

Generative models are used in SSL because they can easily incorporate the unlabeled data points (compared to *discriminative* models, which only focus on estimating $p(y|\mathbf{x})$ and cannot directly exploit the infromation in $p(\mathbf{x})$). On the other hand, the *discriminative* models fulfil Vapnik's principle and can provide comparable results even without using the unlabeled data. In a broader context, SSL can be viewed in the field of generative models as either classification with supplementary information on the marginal density or unsupervised clustering with additional information, i.e., labels of a subset of points. A reasonable requirement on SSL would be that any valid SSL technique should surpass baseline methods by a significant margin across various practical and relevant scenarios.

- **Graph-Based Methods**: Semi-supervised methods that are based on graphs establish a graph structure where the labeled and unlabeled examples in the dataset constitute the nodes, and the similarity between examples is reflected by edges that may be weighted. Typically, these methods smooth the label values across the graph structure, obtaining the proxy label for unlabeled examples. Graph-based approaches are characterized as nonparametric, discriminative, and transductive in nature [53].

When talking about *consistency regularization*, one should also mention **Entropy minimization** [12] as it shares the same underlying concept of *smoothness assumption* and aims at the same result: Moving the decision boundary into the low-density region. The entropy minimization encourages the network to make confident (i.e., low-entropy) predictions on unlabled data regardless of the predicted class, thus moving the decision boundary away from any point in the dataset. We obtain this behaviour by adding an entropy minimization term:

$$H(p) = -\sum_{k=1}^{C} p_\theta(y|\mathbf{x})_k \log p_\theta(y|\mathbf{x})_k$$

Nevertheless, the neural networks (NN) can quickly overfit to low confident points early on in the learning process. Such overfitting is caused by their high capacity [32]. The Entropy minimization alone does not lead to strong results. However, it is often combined with different approaches to improve their performance [33].

2.2 MixMatch

We have selected the MixMatch algorithm as a reference algorithm for the comparison as it yielded state-of-the-art results. This *holistic* approach David Berthelot et *al.* proposed in 2019 [2] and combined several ideas and components from classical dominant paradigms of SSL. It is the cornerstone for new algorithms such as ReMixMatch [3] and FixMatch [42]. Namely, it combines *consistency regularization* and *proxy-labeling* with *entropy minimization*.It also utilizes other forms of regularizations, such as *data augmentation*, *exponentially weighted average of network weights* [45],*weight decay* [25] and *MixUp* procedure [52]. The consistency regularization is obtained through a loss term. The proxy-labeling occurs in the stage of the label guessing (2), and the entropy minimization is applied in the form of a sharpening procedure (3).

The algorithm comprises several steps and provides augmented inputs to the model with *guessed* labels. The batched augmented inputs are propagated through the network, and the standard semi-supervised loss containing the supervised and unsupervised term is computed from the outputs of the model and the (guessed) labels. The gradient is backpropagated to the network's weights, meaning the MixMatch is applicable in the Deep Learning (DL) setting. Assume we have batch of labeled inputs $\mathcal{X}$ (with labels encoded as one-hot vectors with L possible classes) and batch of unlabeled inputs $\mathcal{U}$ (without labels), both with the same number of examples n. The SSL loss is defined as:

$$\begin{aligned}
\mathcal{X}', \mathcal{U}' &= \text{MixMatch}(\mathcal{X}, \mathcal{U}, T, K, \alpha) \\
\mathcal{L}_{\mathcal{X}} &= \frac{1}{|\mathcal{X}'|} \sum_{x', p' \in \mathcal{X}'} H(p', f_\theta(x')) \\
\mathcal{L}_{\mathcal{U}} &= \frac{1}{L|\mathcal{U}'|} \sum_{u', q' \in \mathcal{U}'} ||q' - f_\theta(u')||_2^2 \\
\mathcal{L} &= \mathcal{L}_{\mathcal{X}} + \lambda_{\mathcal{U}} \mathcal{L}_{\mathcal{U}}
\end{aligned}$$

where $H(p, q)$ is cross-entropy loss between distributions p and q:

$$H(p, q) = -\sum_{k=1}^{C} p_k(x) \log q_k(x),$$

T, K, α and $\lambda_{\mathcal{U}}$ are hyperparameters, and $f_\theta(\cdot)$ represents the model output in the form of a probability distribution. T is the *temperature* in the probability sharpening procedure, K is the *number of augmentations* applied to unlabeled input u, and the α is the Beta distribution parameter for MixUp. The $\lambda_{\mathcal{U}}$ replaces the original normalizing factor and provides a tuning knob for weighting the loss terms.

2.2.1 MixMatch algorithm

The MixMatch algorithm consists of the following steps:

1. **Data Augmentation**: Given the (stochastic) augmentation A, we transform each labeled features $x_i \in \mathcal{X}$ into $\tilde{x}_i$ while keeping the original label p unchanged. For unlabeled feature $u_j \in \mathcal{U}$, we produce K augmented views $\tilde{u}_{j,k}$. Through this, we obtain n labeled features and nK unlabeled features.

2. **Label Guessing**: For each of K views of unlabeled feature $\tilde{u}_{j,k}$ we make the predictions with the current model $\hat{q}_{j,k} = f_\theta(\tilde{u}_{j,k})$. We then compute the average

$$\bar{q}_j = \frac{1}{K} \sum_{k=1}^{K} \hat{q}_{j,k}$$

for each unlabeled feature u_j.

3. **Sharpening**: We sharpen the averaged prediction $\bar{q}_j$ to reduce its entropy through the operation:

$$q_{j,c} = \text{Sharpen}(\bar{q}_j, T)_c = \bar{q}_{j,c}^{\frac{1}{T}} \Big/ \sum_{k=1}^{K} \bar{q}_{j,k}^{\frac{1}{T}}$$

where $q_{j,c}$ corresponds to c-th element of vector q_j, representing the probability of c-th class. The hyperparameter $T \in \mathbb{R}_{>0}$ is the *temperature*. As $T \to 0$, the $\text{Sharpen}(p, T)$ approaches Dirac (one-hot) distribution, therefore lowering the T minimizes the entropy of p. We obtain the sharpened q_j and replicate it to each of K views of feature u_j.

4. **MixUp**: Before further describing, we define the slightly alternated version of the vanilla MixUp [52]. For a pair of two features with their corresponding class probabilities (x_1, p_1) and (x_2, p_2), we define MixUp operation as follows:

$$\begin{aligned}
\lambda &\sim \text{Beta}(\alpha, \alpha) \\
\lambda' &= \max(\lambda, 1 - \lambda) \\
x' &= \lambda' x_1 + (1 - \lambda') x_2 \\
p' &= \lambda' p_1 + (1 - \lambda') p_2
\end{aligned}$$

where α is hyperparameter. Vannila MixUp omits the second equation (i.e. $\lambda' = \lambda$), but it is crucial in MixMatch as you will see later. We define MixUp operation for (equally sized) sets* as a MixUp per elements, i.e.

$$\text{MixUp}(\mathcal{D}_a, \mathcal{D}_b) = \{\text{MixUp}((x_{ai}, y_{ai}), (x_{bi}, y_{bi})) \,|\, i \in 1, \ldots, |\mathcal{D}_a|\}.$$

Going back to MixMatch, the previous steps resulted in two batches with different sizes:

$$\begin{aligned}
\mathcal{X}^\star &= \{(\tilde{x}_i, p_i) \,|\, i \in \{1, \ldots, n\}\}, \; |\mathcal{X}^\star| = n \\
\mathcal{U}^\star &= \{(\tilde{u}_{j,k}, q_j) \,|\, j \in \{1, \ldots, n\}, \; k \in \{1, \ldots, K\}\}, \; |\mathcal{U}^\star| = Kn
\end{aligned}$$

*We should rather speak about sequences, as the sets do not have ordering. Nevertheless, in the field of ML, we often neglect this difference. In reality, computer memory always has the implicit ordering, which is used.

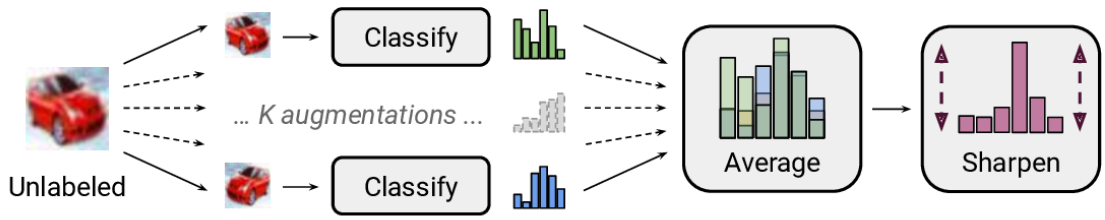


Figure 2.1: Data augmentation, label guessing and sharpening procedure visualized for unlabeled datapoint. The unlabeled image is at first K times augmented, each augmentation is then classified by the current model. The predictions are then averaged and sharpened. Source [2]

First, we concatenate those two batches and shuffle them :

$$\mathcal{W} = \text{Shuffle}(\text{Concat}(\mathcal{X}^\star, \mathcal{U}^\star))$$

we then slice the $\mathcal{W}$ into two: $\mathcal{W}_1$ of the same size as $\mathcal{X}^\star$ and $\mathcal{W}_2$ of the same size as $\mathcal{U}^\star$, and we compute MixUp for both labeled and unlabeled sets:

$$\mathcal{X}' = \text{MixUp}(\mathcal{X}^\star, \mathcal{W}_1)$$
$$\mathcal{U}' = \text{MixUp}(\mathcal{U}^\star, \mathcal{W}_2)$$

The definition of λ' in alternated MixUp ensures, that the (x', y') is always closer to the (x_1, y_1) then to (x_2, y_2), which is important as it may happen, that the $\mathcal{W}_1$ will contain features from $\mathcal{U}$ Furthermore, we need to compute individual loss components appropriately. In other words, the $\mathcal{X}'$ and $\mathcal{U}'$ are always closer to the $\mathcal{X}^\star$, resp. $\mathcal{U}^\star$ so the computed loss corresponds to the original inputs, i.e. batches $\mathcal{X}$, resp. $\mathcal{U}$.

Algorithm 1 MixMatch

Input: Batch of labeled examples and their one-hot labels $X = ((x_i, p_i); i \in (1, \dots, n))$, batch of unlabeled examples $U = (u_i; i \in (1, \dots, n))$, sharpening temperature T, number of augmentations K, Beta distribution parameter α for MixUp.
for $i = 1$ **to** n **do**
  $\bar{x}_i = \text{Augment}(x_i)$ ▷ Apply data augmentation to x_i
  for $k = 1$ **to** K **do**
    $\bar{u}_{i,k} = \text{Augment}(u_i)$ ▷ Apply kth round of data augmentation to u_b
  end for
  $\bar{q}_i = \frac{1}{K} \sum_{k=1}^{K} p_{\text{model}}(y|\bar{u}_{i,k}; \theta)$ ▷ Compute average predictions across all augmentations of u_i
  $q_i = \text{Sharpen}(\bar{q}_i, T)$ ▷ Apply temperature sharpening to the average prediction
end for
$X^\star = ((\bar{x}_i, p_i); i \in (1, \dots, n))$ ▷ Augmented labeled examples and their labels
$U^\star = ((\bar{u}_{i,k}, q_i); i \in (1, \dots, n), k \in (1, \dots, K))$ ▷ Augmented unlabeled examples, guessed labels
$W = \text{Shuffle}(\text{Concat}(X^\star, U^\star))$ ▷ Combine and shuffle labeled and unlabeled data
$X' = (\text{MixUp}(\bar{x}_i, w_i); i \in (1, \dots, |X^\star|))$ ▷ Apply MixUp to labeled data and entries from W
$U' = (\text{MixUp}(\bar{u}_i, w_{i+|X^\star|}); i \in (1, \dots, |U^\star|))$ ▷ Apply MixUp to unlabeled data and the rest of W
return X', U'

2.3 Variational Autoencoders (VAEs)

The *Variational Autoencoder* or VAE for short, is a generative model that falls into generative modelling (page 5). It is a neural network architecture capable of learning a low-dimensional representation of complex high-dimensional data such as images, text, or audio. The VAE is a probabilistic model that learns to approximate the true data distribution by using an encoder network to map input data into a latent space and a decoder network to map the latent space back to the original data space. Nevertheless, before we delve into the details of VAE, let us explain the term "Variational Autoencoders" itself and what it represents. The explanation comes in two parts: first, we explain autoencoders in subsection 2.3.1 and then variational inference in subsection 2.3.2. The experienced reader can skip those introductory parts and go right to subsection 2.3.3.

2.3.1 Autoencoders

An autoencoder was first introduced in the 1980s by Hinton [38]. However, it was only with the advent of deep learning and the availability of large amounts of data and

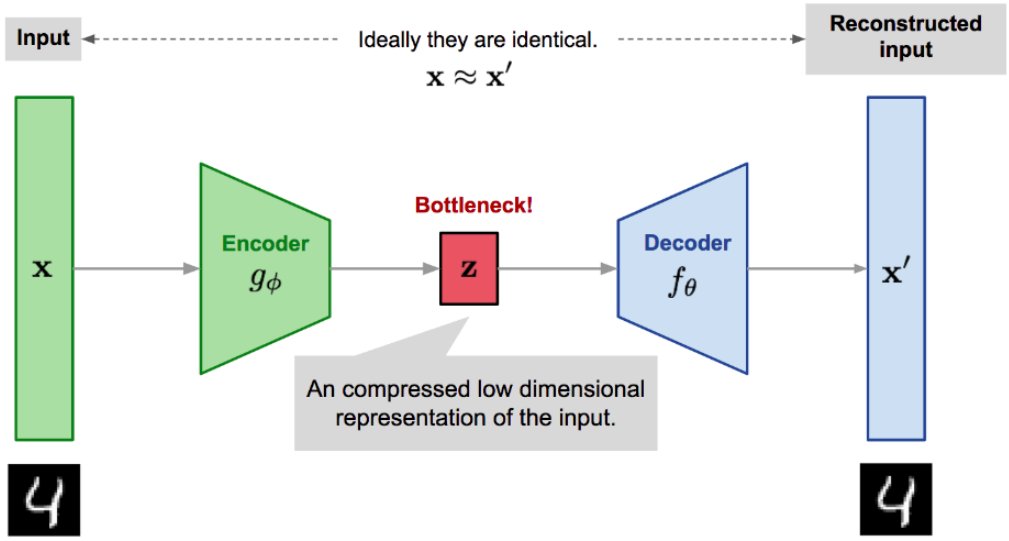


Figure 2.2: Illustration of autoencoder network with two networks: *Encoder* and *Decoder*, each parametrized by learnable parameters. Source [47]

computational resources in the 2000s and 2010s that autoencoders became widely used and achieved state-of-the-art results in various tasks [14]. An *autoencoder* is a neural network designed to learn identity mapping in an unsupervised manner to reconstruct the original input while compressing the information in the "bottleneck" layer to obtain a compressed representation (see 2.2). Through this, we obtain an efficient dimensionality reduction: The low-dimensional latent representation can be used as an embedding vector in various applications, such as search or data compression [47].

Plato alegory

The concept of latent variables in generative modelling can be explained using Plato's Allegory of the Cave [48]. In the allegory, people are confined to a cave and can only see two-dimensional shadows of three-dimensional objects projected onto a wall. Similarly, the objects we observe in the world may be generated by higher-level abstract concepts that we can never directly observe. These abstract concepts may represent properties such as colour, size, and shape. Even though we never see and can not fully comprehend these higher-level concepts in all details, we can still reason and draw inferences about them through their manifestation in our lives. Similarly, we can approximate the latent representations, which encode the observed data [26].

However, in generative modelling, we typically seek to learn lower-dimensional latent representations rather than higher-dimensional ones. This is because attempting to learn a representation of a higher dimension than the observation is often difficult without strong priors. Learning lower-dimensional latent can also be viewed as a form of compression, which can uncover semantically meaningful structures describing observations.

The autoencoder's architecture is composed of two networks:

- *Encoder* network, which takes (high-dimensional) input $\boldsymbol{x}$ and maps it into low-dimensional latent code $\boldsymbol{z}$. We denote it as a function $\boldsymbol{g}(\cdot)$ parametrized by $\boldsymbol{\phi}$. Its goal is to reduce dimensionality, like any other approaches such as principle component analysis (PCA) or t-SNE.
- *Decoder* network, which takes the code $\boldsymbol{z}$ and recovers the data $\boldsymbol{x}$. We donte it as an function $\boldsymbol{f}(\cdot)$ parametrized by $\boldsymbol{\theta}$.

The parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are learned simultaneously using stochastic gradient descent (SGD). We encode the input, decode it, and compute the mean squared error (MSE) loss for each feature in the batch to train the model. By minimizing this loss, we encourage the model to produce output as close as possible to the input.

$$\boldsymbol{x}' = f_{\boldsymbol{\theta}}(\boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x})) = f_{\boldsymbol{\theta}}(\boldsymbol{z})$$
$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}'_i - \boldsymbol{x}_i)^2$$

Vanilla autoencoders can be prone to overfitting if their capacity is too high relative to the dataset size. To improve their robustness, Vincent et *al.* proposed a *denoising* autoencoder in 2008 [46]. This approach involves adding random noise to the input and then training the model to reconstruct the original signal. Denoising autoencoders can achieve better generalization performance by forcing the model to learn the underlying structure of the data rather than just memorizing the training examples. The idea of adding noise is today known as the dropout technique [44].

Since then, other architectures have been proposed to improve robustness and prevent overfitting. These include sparse autoencoders [31], k-sparse autoencoders [28], and contractive autoencoders [36]. The novel approach was defined in 2013 by Kingma and Welling [17], and VAEs were introduced. The key idea was to assume that latent space is not deterministic but stochastic with some distribution $p(\boldsymbol{z})$ over it. The goal of the VAE is to model this distribution by variational bayes.

2.3.2 Variational Bayes

The idea of autoencoder was described above, but what does the "*variational*" in "variational autoencoder" stands for? Calculus of variations is a branch of mathematics that deals with finding the optimal solution of a functional, which is a mathematical function that maps a set of functions to real numbers. The optimal solution is the function that minimizes or maximizes the functional. The calculus of variations has many applications in physics [13], engineering [21], economics, and other fields. Its development

dates back to the 17th century. The Euler-Lagrange equations [9] play a crucial role in variational calculus by providing the stationary points of a given functional that needs to be minimized or maximized.

Now we know what *variations* are, but how do we apply them in the context of Bayesian inference? Let us assume we have a probabilistic graphical model (PGM) or Bayesian network [49] with some hidden (or unobserved) nodes H and some observed nodes (evidence) E. The goal of Bayesian inference is to compute posterior probability $p(H \mid E)$:

$$p(H \mid E) = \frac{p(H, E)}{p(E)} = \frac{p(E \mid H)p(H)}{p(E)},$$

where $p(E)$ is the marginal density of the evidence:

$$p(E) = \int_H p(H, E)\, \mathrm{d}H \tag{2.1}$$

For most of the models, computing the evidence is intractable due to the integral in eq. 2.1. The interactability might be an issue even for a case with the sum of a large number of discrete random variables (even if they were from simple categorical distribution), as it would take too much time to compute. The intractability of $p(E)$ is related to the intractability of the posterior distribution $p(H \mid E)$. Note that the joint distribution $p(H, E)$ is efficient to compute as it is specified through the model (PGM). The Bayes formula relates the densities:

$$p(E) = \frac{p(H, E)}{p(H|E)} \quad \text{or} \quad p(H|E) = \frac{p(H, E)}{p(E)}$$

Since $p(H, E)$ is tractable to compute, a tractable marginal likelihood $p(E)$ leads to a tractable posterior $p(H \mid E)$, and vice versa [18, 16].

Variational Bayes (VB) is a technique for approximating complex probability distributions by simpler ones and was introduced by Jordan et al. in 1999 [16]. VB provides a way to approximate the posterior distribution by a simpler one that belongs to a tractable family of distributions and to do so by minimizing the Kullback-Leibler (KL) divergence between the true posterior and the approximate one. The KL divergence is a functional with respect to approximate tractable distribution $q(H)$ since the true posterior $p(H \mid E)$ is given, hence the connection with *variational calculus*. The KL divergence is a measure of the dissimilarity between two probability distributions:

$$q^\star(H) = \operatorname*{argmin}_{q \in \mathcal{Q}} \mathrm{KL}(q(H)||p(H \mid E))$$
$$\mathrm{KL}(q(H)||p(H \mid E)) = \int q(H) \log \left[\frac{q(H)}{p(H \mid E)}\right] \mathrm{d}H,$$

where $\mathcal{Q}$ is the family of tractable distributions, and KL is the KL divergence. It should be noted that there are other non-optimization-based methods to make such approximate inferences, such as MCMC [50].

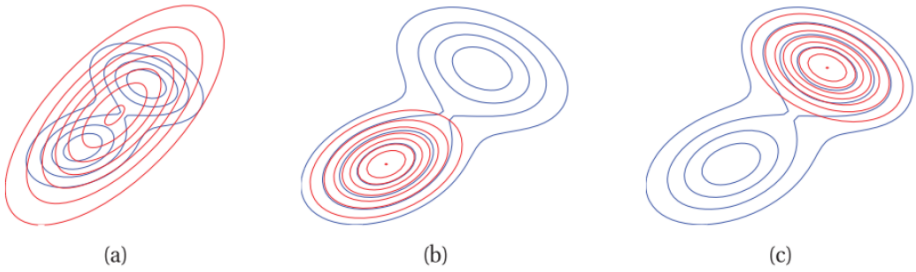


Figure 2.3: Illustration of forward vs reverse KL-divergence on a bimodal distribution. The blue and the red contours represent the target p and the unimodal approximation q, respectively.In (a), the forward KL-divergence minimization is visualized with q trying to cover up p. The (b) and (c) show the reverse KL-divergence where q locks on to one of the two modes. Source [10]

As the KL divergence is not symmetrical, one could ask why we have defined the optimization task as the reverse KL-divergence and not the other way around, i.e. forward KL divergence $\mathrm{KL}(p(H \mid E) \mid q(H))$. Both cases are illustrated in figure 2.3. The reverse KL divergence minimization results in q *under-estimating* p, which can be perceived as a safe choice. This choice ensures that sampling from found q provides plausible values under original p; in other words, we do not want to obtain samples from q that the original p does not support. For a thorough explanation, we refer to the literature [10]. We will now relabel our variables to follow the standard notation used in deep learning literature, where hidden variables H are known as latent $\mathbf{z}$ and observed E as features $\mathbf{x}$. In this setting, we want to optimize:

$$q^\star(\mathbf{z}) = \underset{q \in \mathcal{Q}}{\operatorname{argmin}}\, \mathrm{KL}(q(\mathbf{z}) || p(\mathbf{z} \mid \mathbf{x})) \tag{2.2}$$

$$\mathrm{KL}(q(\mathbf{z}) || p(\mathbf{z} \mid \mathbf{x})) = \int_{\mathbf{z}} q(\mathbf{z}) \log \left[\frac{q(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})}\right] \mathrm{d}\mathbf{z} \tag{2.3}$$

$$= \int_{\mathbf{z}} \left[q(\mathbf{z}) \log q(\mathbf{z})\right] \mathrm{d}\mathbf{z} - \int_{\mathbf{z}} \left[q(\mathbf{z}) \log p(\mathbf{z} \mid \mathbf{x})\right] \mathrm{d}\mathbf{z} \tag{2.4}$$

$$= \mathbb{E}_q \left[\log q(\mathbf{z})\right] - \mathbb{E}_q \left[\log p(\mathbf{z} \mid \mathbf{x})\right] \tag{2.5}$$

$$= \mathbb{E}_q \left[\log q(\mathbf{z})\right] - \mathbb{E}_q \left[\log \left(\frac{p(\mathbf{x}, \mathbf{z})}{\mathbf{x}}\right)\right] \tag{2.6}$$

$$= \mathbb{E}_q \left[\log q(\mathbf{z})\right] - \mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z})\right] + \mathbb{E}_q \left[\log p(\mathbf{x})\right] \tag{2.7}$$

$$= \mathbb{E}_q \left[\log q(\mathbf{z})\right] - \mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z})\right] + \log p(\mathbf{x}) \tag{2.8}$$

where the $\mathbb{E}_q \left[\log p(\mathbf{x})\right] = \log p(\mathbf{x})$ because the $p(\mathbf{x})$ does not depend on $q(\mathbf{x})$. We can not optimize the KL divergence directly since the evidence $p(\mathbf{x})$ is intractable. However, it is constant (for the given dataset). If we rearrange the last equation, we obtain

$$\log p(\mathbf{x}) - \mathrm{KL}(q(\mathbf{z}) || p(\mathbf{z} \mid \mathbf{x})) = \mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z})\right] - \mathbb{E}_q \left[\log q(\mathbf{z})\right] \tag{2.9}$$

$$= \mathrm{ELBO}(q) \tag{2.10}$$

where the left-hand side (LHS) of the equation is called evidence lower bound (ELBO) since it is truly a lower bound on the logarithm of evidence $p(\mathbf{x})$. This is clear to see, as

the KL divergence is always positive. As the $\log p(\mathbf{x})$ is constant, maximizing the RHS is equal to minimizing the KL-divergence:

$$\begin{aligned} q^\star(\mathbf{z}) &= \underset{q\in\mathcal{Q}}{\operatorname{argmin}}\, \mathrm{KL}(q(\mathbf{z})||p(\mathbf{z}\mid\mathbf{x})) \\ &= \underset{q\in\mathcal{Q}}{\operatorname{argmax}}\, \mathrm{ELBO}(q) \\ &= \underset{q\in\mathcal{Q}}{\operatorname{argmax}} \left[\mathbb{E}_q\left[\log p(\mathbf{x},\mathbf{z})\right] - \mathbb{E}_q\left[\log q(\mathbf{z})\right]\right] \end{aligned}$$

A commonly made assumption regarding the variational distribution $q(\mathbf{z})$ is that it factorizes over a partition of the latent variables, such that the latent variables can be partitioned into subsets $\mathbf{z}_1, \ldots, \mathbf{z}_m$. Specifically, we can write:

$$q(\mathbf{z}) = \prod_{i=1}^{m} q_i(\mathbf{z}_i \mid \mathbf{x})$$

This assumption is known as a *mean field approximation*. It can be shown (see [29]) that the following the equation holds for optimal $q_j^\star$:

$$\log q_j^\star(\mathbf{z}_j \mid \mathbf{x}) = \mathbb{E}_{i\neq j} \log p(\mathbf{x},\mathbf{z}) + \text{constant}$$

where $\mathbb{E}_{i\neq j}[\log p(\mathbf{z},\mathbf{x})]$ represents the expected value of the logarithm of the joint probability of the observed data and latent variables, taken over all variables that are not included in the current partition. The constant is only a normalizing constant, so $q_j^\star$ is truly a probability distribution. This leads to an iterative algorithm of block-coordinate ascent on ELBO with an initial random guess (see algorithm 2). Furthermore, Neal and Hinton have shown in [30] that the Expectation Maximization (EM) alogrithm (proposed by Dempster et *al.* [8]) can be seen as such ascent on ELBO.

Algorithm 2 Coordinate Ascent Variational Inference (CAVI) Source: [29]

Input: A model $p(\mathbf{x},\mathbf{z})$, a dataset $\mathbf{x}$
Initialize: Variational factors $q_j(\mathbf{z}_j)$
while the ELBO has not converged **do**
  for $j \in \{1,\ldots,m\}$ **do**
    Set $q_j(z_j) \leftarrow \frac{\exp\{\mathbb{E}_{i\neq j}[\log p(\mathbf{z},\mathbf{x})]\}}{\int_{\mathbf{z}_j}\exp\{\mathbb{E}_{i\neq j}[\log p(\mathbf{z},\mathbf{x})]\}\mathrm{d}\mathbf{z}_j}$
  end for
  Compute $\mathrm{ELBO}(q) \leftarrow \mathbb{E}[\log(p(\mathbf{x},\mathbf{z}))] - \mathbb{E}[\log q(\mathbf{z})]$
end while
return $\prod_{j=1}^{m} q_j(\mathbf{z}_j)$

2.3.3 VAE

Kingma and Welling first introduced the variational autoencoders in 2013 [17]. Since its introduction, the VAE has been widely used in research and industry and has inspired

many other generative models. It continues to be an active area of research, with ongoing efforts to improve its performance and applicability to different domains. This subsection was mainly inspired by [18].

Deep Latent Variable Models

A deep latent variable model (DLVM) is a probabilistic graphical model or Bayesian network where some variables are hidden or latent [18]. These models use neural networks to parameterize the distributions of their variables, enabling very complex marginal distributions (evidence) $p_{\boldsymbol{\theta}}(\mathbf{x})$ even if each factor in the directed model (prior $p_{\boldsymbol{\theta}}(\mathbf{z})$ or conditional $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$) is relatively simple. This expressivity makes them attractive for approximating complicated true distributions $p^\star(\mathbf{x})$. The simplest DLVM is one that is factored as with the following structure:

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

where $p_{\boldsymbol{\theta}}(\mathbf{z})$ or $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$ is specified, i.e. their family of distribution is fixed and its distribution parameters $\boldsymbol{\eta}$ are parametrized by neural network:

$$\begin{aligned} \boldsymbol{\eta} &= \mathrm{NeuralNet}_{\boldsymbol{\theta}}(\mathrm{Pa}(\mathbf{x})) \\ p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathrm{Pa}(\mathbf{x})) = p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{\eta}) &= p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \mathrm{NeuralNet}_{\boldsymbol{\theta}}(\mathrm{Pa}(\mathbf{x}))) \end{aligned}$$

or even both their family and distribution parameters are fixed, e.g.

$$p_{\boldsymbol{\theta}}(\mathbf{z}) = p(\mathbf{z}) = \mathcal{N}(\mathbf{z}, 0, \mathbf{I})$$

The $\mathrm{Pa}(\mathbf{x})$ corresponds to all parents of node $\mathbf{x}$ in Bayesian Network. Such DLVM is visualized in picture 2.4 as a Bayesian network. We will later see that it is precisely this model that is under consideration when talking about VAE.

Example 2.1 (DLVM for multivariate Bernoulli data)**.** A simple example of DLVM for binary model $\mathbf{x} \in \{0, 1\}^D$ used in [19] has following structure: The latent space $\mathbf{z}$ is fixed as a spherical Gaussian distribution and conditional probability is modelled as a factorized Bernoulli:

$$\begin{aligned} p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}, 0, \mathbf{I}) \\ \mathbf{p} &= \mathrm{DecoderNet}_{\boldsymbol{\theta}}(\mathbf{z}) \\ \log p(\mathbf{x} \mid \mathbf{z}) &= \sum_{i=1}^{D} \log(x_i \mid \mathbf{z}) = \sum_{i=1}^{D} \log \mathrm{Bern}(x_i, p_i) \\ &= \sum_{i=1}^{D} x_i \log p_i + (1 - x_i) \log(1 - p_i) \end{aligned}$$

where $\mathbf{p} \in \{0, 1\}^D$ and $\mathrm{Bern}(\cdot, p)$ is the probability mass function (PMF) of the Bernoulli distribution.

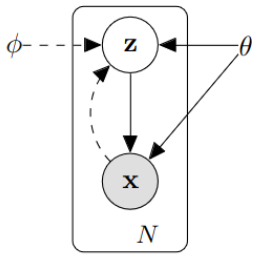


Figure 2.4: Illustration of VAE as Bayesian Network. Solid lines denote generative capabilities $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})$, dashed lines denote the variational approximation $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$ to the intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$. The variational parameters $\boldsymbol{\phi}$ are learned jointly with the generative model pa- rameters $\boldsymbol{\theta}$. Source [17]

Until now, we have only discussed DLVMs, which involve using neural networks in Bayesian networks. However, DLVMs face the same issues as other Bayesian models discussed in the Variational Bayes section, such as the inability to compute the evidence $p(\mathbf{x})$ due to intractabilities. To overcome this, approximate inference techniques such as Expectation Maximization (EM) (see 2) or Monte Carlo Markov Model (MCMC) can be used to approximate the posterior $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$ and the marginal likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$ [18, appx. A2]. However, these traditional inference methods are computationally expensive, often requiring per-datapoint optimization loops and are not well-suited for datasets with a large number of examples N. Moreover, they tend to yield poor posterior approximations. Thus, there is a need for more efficient procedures.

Approximate Posterior

The VAEs framework introduces two models, encoder and decoder (see figure 2.4) and also proposes a computationally efficient algorithm for optimizing the parameters jointly using SGD.

To address the intractabilities of DLVM, VAEs utilize the *recognition model* $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$, also known as an *inference model* or an *encoder*, to approximate the intractable $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$. While the encoder, similar to DLVM, can be any Bayesian network, in vanilla VAE, it is typically a simple network with $\mathbf{x}$ as input, which produces the parameter for the distribution for the latent $\mathbf{z}$. We denote the parameters of this network with $\boldsymbol{\phi}$ to differentiate them from $\boldsymbol{\theta}$, and refer to them as *variational*, as the goal of the encoder is to approximate the conditional distribution $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$:

$$q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \approx p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x}).$$

It is important to note that, unlike the approximate posterior in mean-field variational inference, the encoder is not necessarily factorial, and its parameters $\boldsymbol{\phi}$ are not computed from a closed-form expectation [17]. Instead, VAEs' framework introduces a method for jointly learning the recognition model parameters $\boldsymbol{\phi}$ and the generative model parameters

$\boldsymbol{\theta}$. This approach is known as *amortized variational inference* [11], which avoids a per-datapoint optimization loop and leverages the efficiency of SGD [18].

ELBO objective

The objective to minimize is an ELBO (defined in 2.9), like in any other Bayes variational method. For any selection of recognition model $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$, we can write:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right] - \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})\right] \quad + \mathrm{KL}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) || p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})) \tag{2.11}$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathrm{ELBO}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})) \quad + \mathrm{KL}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) || p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})) \tag{2.12}$$

This equation provides important insight into the divergence $\mathrm{KL}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) || p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x}))$. Firstly, it measures how well the approximating distribution $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$ fits the intractable distribution of decoder model $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$. Secondly, it quantifies the gap between the ELBO and the log-likelihood of the marginal likelihood $\log p_{\boldsymbol{\theta}}(\mathbf{x})$, also knwon as *tightness* of the bound. A tighter bound indicates a better fit between the approximate and model posterior distributions. Therefore maximizing the ELBO w.r.t to $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ can be seen as a concurrent task of improving the log evidence $p_{\boldsymbol{\theta}}(x)$ under the model and also improving the fit of variational $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$ to $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$:

$$\underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\operatorname{argmax}}\, \mathrm{ELBO}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})) = \underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\operatorname{argmax}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}) - \mathrm{KL}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) || p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x}))\right] \tag{2.13}$$

In order to use SGD, we need to compute the derivatives of the ELBO objective with respect to the model parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. The ELBO objective of a (mini)-batch is defined as the sum of terms for each data point in the (mini)-batch $\mathcal{B}$:

$$\mathrm{ELBO}(\mathcal{B}) = \sum_{\mathbf{x} \in \mathcal{B}} \mathrm{ELBO}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}))$$

where the data point term is defined as

$$\mathrm{ELBO}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right] - \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})\right].$$

This term depends on the data point $\mathbf{x}$ and the network parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ that correspond to the distributions $p_{\boldsymbol{\theta}}$ and $q_{\boldsymbol{\phi}}$, respectively. The ELBO is sometimes rearranged into:

$$\begin{aligned}
\mathrm{ELBO}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) + \log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})\right] - \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z})}\right] \\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})\right]}_{\text{recognition term}} - \underbrace{\mathrm{KL}(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) || p_{\boldsymbol{\theta}}(\mathbf{z}))}_{\text{variational regularization term}}
\end{aligned} \tag{2.14}$$

Although the individual data point ELBO and its gradient $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \operatorname{ELBO}(\mathrm{x}, \boldsymbol{\theta}, \boldsymbol{\phi})$ are generally not tractable, VAEs provide means and assumptions under which we can perform minibatch SGD. Namely we will show that good unbiased estimators $\hat{\nabla}_{\boldsymbol{\theta},\boldsymbol{\phi}} \operatorname{ELBO}(\mathrm{x}, \boldsymbol{\theta}, \boldsymbol{\phi})$ exist under the assumption that we know how to reparametrize the $q_{\boldsymbol{\phi}}$ distribution to remove the $\boldsymbol{\phi}$ influence on sampling. This is also known as *reparametrization trick* and is a key achievement of VAEs frameworks.

Say we want to compute gradient w.r.t. to decoder parameters $\boldsymbol{\theta}$, i.e.

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \operatorname{ELBO}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \nabla_{\boldsymbol{\theta}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \right] && (2.15) \\
&\simeq \nabla_{\boldsymbol{\theta}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \tilde{\mathbf{z}}) - \log q_{\boldsymbol{\phi}}(\tilde{\boldsymbol{z}} \mid \mathbf{x}) \right] \\
&\simeq \nabla_{\boldsymbol{\theta}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \tilde{\mathbf{z}}) \right] && (2.16)
\end{aligned}$$

The $\simeq$ symbol means that one of the two sides is an unbiased estimator of the other side. The RHS is a random variable with some source of the noise and two sides are equal when averaged over the noise distribuion. In this case, the noise distribution is $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$ and $\tilde{\mathbf{z}}$ is sampled from it. In other words, the last line (eq. 2.16) is a Monte Carlo estimator of the second line (eq. 2.15) [18].

The unbiased gradient w.r.t. to the encoder parameters $\boldsymbol{\phi}$ is slightly difficult to obtain as the expectation of the ELBO is taken w.r.t. the variational distribution $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$, which is a function of $\boldsymbol{\phi}$, i.e.

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} \operatorname{ELBO}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) &= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \right] && (2.17) \\
&\neq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \nabla_{\boldsymbol{\phi}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \right] && (2.18)
\end{aligned}$$

Nevertheless, in the case of continuous latent variables, we can use *reparametrization trick* [35] as we will show now [18].

Reparametrization trick

By parametrizing the random variable $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$ as a differentiable and invertible mapping of another random variable $\boldsymbol{\epsilon}$ that is independent of both the data $\mathbf{x}$ and the variational parameters $\boldsymbol{\phi}$, we can obtain an unbiased gradient and differentiate the evidence lower bound (ELBO) objective with respect to $\boldsymbol{\phi}$. Specifically, we have

$$\mathbf{z} = \mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}),$$

where $\boldsymbol{\epsilon}$ is randomly sampled from some distribution $p(\boldsymbol{\epsilon})$. Using this mapping, we can separate the noise source from the variational parameters and obtain the Monte Carlo estimator of the gradient. i.e.

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})} \right] &= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})) - \log q_{\boldsymbol{\phi}}(\mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \mid \mathbf{x}) \right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})} \nabla_{\boldsymbol{\phi}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})) - \log q_{\boldsymbol{\phi}}(\mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \mid \mathbf{x}) \right] \\
&\simeq \nabla_{\boldsymbol{\phi}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \tilde{\mathbf{z}}) - \log q_{\boldsymbol{\phi}}(\tilde{\mathbf{z}} \mid \mathbf{x}) \right],
\end{aligned}$$

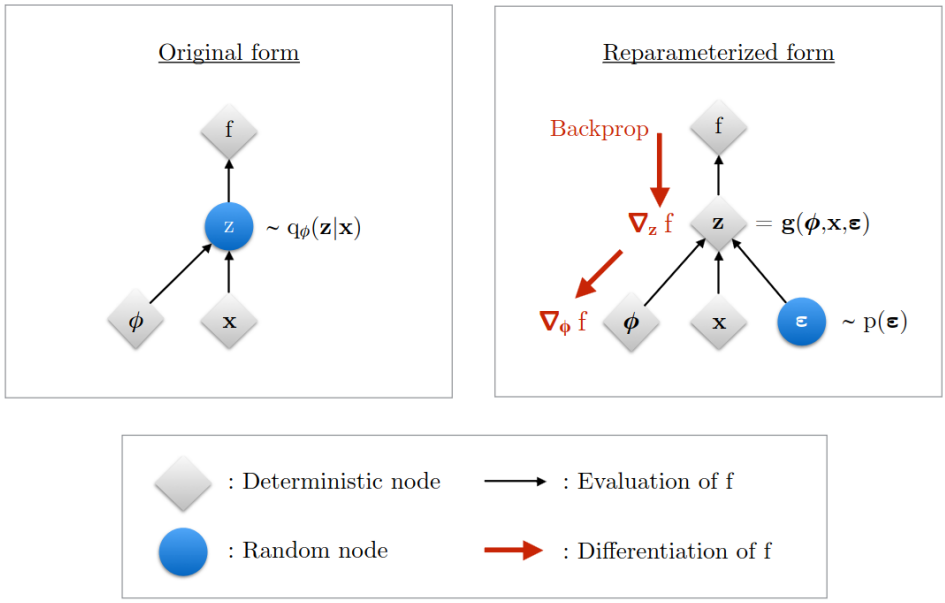


Figure 2.5: Illustration of reparametrization trick. The encoder parameters ϕ affect the objective f indirectly through the random variable $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x})$. We can not compute the gradient $\nabla_\phi f$ in the original setting (left), as we can not propagate the dependency of ϕ through the sampling process. Thanks to the reparameterization trick, we can first sample the ϵ and then compute $\mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})$ which enables us to compute the gradients $\nabla_\phi f$. Source [18]

where $\tilde{\mathbf{z}} = \mathbf{g}(\tilde{\boldsymbol{\epsilon}}, \boldsymbol{\phi}, \mathbf{x})$ for randomly sampled $\tilde{\boldsymbol{\epsilon}} \sim p(\boldsymbol{\epsilon})$. With this procedure, we can now differentiate the ELBO objective with respect to $\boldsymbol{\phi}$, which was not possible before. The situation is visualized in figure 2.5. The limitations of this procedure are that we have to know the mapping $\mathbf{g}$. For the Gaussian distribution, the particular mapping is the following:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and standard deviation of the Gaussian distribution, respectively, $\boldsymbol{\epsilon}$ is a vector of independent samples from a standard normal distribution, and $\odot$ denotes element-wise multiplication. The final algorithm for computing SGD for the ELBO loss is provided in 3.

Algorithm 3 Stochastic gradient descent for VAE. (a.k.a. Auto-Encoding Variational Bayes (AEVB) Algorithm)

Input: Dataset $\mathcal{D}$, Inference model $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$, Generative model $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$
Return: Learned parameters $\boldsymbol{\theta}, \boldsymbol{\phi}$
$(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow$ Initialize parameters
while SGD not converged **do**
    $M \sim \mathcal{D}$ ▷ Random minibatch of data
    $\tilde{L}_{\theta,\phi} = 0$
    for $\boldsymbol{m}$ **in** M **do**
        Sample $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$
        Apply encoder $\boldsymbol{\eta} = \mathrm{NeuralNet}_{\boldsymbol{\phi}}(\boldsymbol{m})$
        Reparametrize to obtain sample $\tilde{\boldsymbol{z}} = \boldsymbol{g}(\boldsymbol{\eta}, \boldsymbol{m}, \boldsymbol{\epsilon})$ ▷ Via reparametrization trick
        Apply decoder on $\tilde{\boldsymbol{z}}$ and compute $\log(p_{\boldsymbol{\theta}}(\mathbf{x} = \boldsymbol{m} \mid \mathbf{z} = \tilde{\boldsymbol{z}})) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z} = \tilde{\boldsymbol{z}})$
        Compute negative log: $-\log q_{\boldsymbol{\phi}}(\boldsymbol{z} = \tilde{\boldsymbol{z}} \mid \boldsymbol{x} = \boldsymbol{m})$ ▷ See subsection 2.3.3
        $\tilde{L}_{\theta,\phi}$ += $\log(p_{\boldsymbol{\theta}}(\mathbf{x} = \boldsymbol{m}, \mathbf{z} = \tilde{\boldsymbol{z}})) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z} = \tilde{\boldsymbol{z}} \mid \boldsymbol{x} = \boldsymbol{m})$ ▷ Accumulate loss
    end for
    Compute gradients $\nabla_{\theta,\phi} \tilde{L}_{\theta,\phi}(M, \boldsymbol{\epsilon})$
    Update θ and ϕ using SGD optimizer
end while

Evaluation of $\log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$

The evaluation of the estimator of the ELBO requires computation of the density log: (see algorithm 3)

$$\log q_{\boldsymbol{\phi}}(\boldsymbol{z} = \tilde{\boldsymbol{z}} \mid \boldsymbol{x} = \boldsymbol{m}).$$

The $\tilde{\boldsymbol{z}}$ is reparametrized through the *reparametrization trick*

$$\tilde{\boldsymbol{z}} = \mathbf{g}(\tilde{\boldsymbol{\epsilon}}, \boldsymbol{\phi}, \mathbf{x}), \quad \tilde{\boldsymbol{\epsilon}} \sim p(\boldsymbol{\epsilon})$$

and as long as the $\mathbf{g}(\cdot)$ is invertible, we can express the density of $\boldsymbol{z}$ through the density of $\boldsymbol{\epsilon}$:

$$q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) = \log p(\boldsymbol{\epsilon}) - \log d_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon})$$

where $d_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon})$ is the log of the absolute value of the determinant of the Jacobian matrix [18]:

$$d_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = |\det J(\boldsymbol{z}, \boldsymbol{\epsilon})|$$

$$J(\boldsymbol{z}, \boldsymbol{\epsilon}) = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = \begin{bmatrix} \frac{\partial z_1}{\partial \epsilon_1} & \frac{\partial z_1}{\partial \epsilon_2} & \cdots & \frac{\partial z_1}{\partial \epsilon_k} \\ \frac{\partial z_2}{\partial \epsilon_1} & \frac{\partial z_2}{\partial \epsilon_2} & \cdots & \frac{\partial z_2}{\partial \epsilon_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_k}{\partial \epsilon_1} & \frac{\partial z_k}{\partial \epsilon_2} & \cdots & \frac{\partial z_k}{\partial \epsilon_k} \end{bmatrix}$$

This transformation $\boldsymbol{g}(\cdot)$ can also help obtain a more expressive inference model. Namely, even though the decoder and encoder networks would be very complex and could

learn the true data distribution, the ELBO optimization prohibits them from doing so, as it enforces the consistency between the encoder and decoder. If the decoder and encoder are from an exponential family, the simple affine mapping is enough to ensure the encoder and decoder consistency for any exponential distribution [41] and ELBO maximization forces the model to be simple. It has been shown that flexible and invertible transformation $\boldsymbol{g}(\cdot)$ can overcome this issue [41, 7]. Between such transformations belong Normalizing flows (NF) [34] or Inverse autoregressive transformations and Inverse autoregressive flows (IAF) [20]. Another possibility how to overcome this consistency issue is to extend the latent space with auxiliary latent variables [27] which leads to hierarchical variational autoencoders (HVAE) first introduced in [43].

2.3.4 Issues with VAEs

As with any method, the vanilla VAEs also have weak points, which have been identified and pose an open question with active research:

- **Bluriness of the images**: If the VAE model does not have enough capacity to model the underlying true distribution properly, the variance of decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ will end up larger then the variance of the encoder

  $$q_{\phi,\mathcal{D}}(\boldsymbol{x}, \boldsymbol{z}) = q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) p_{\mathcal{D}}(\mathbf{x})$$

  where $p_{\mathcal{D}}(\mathbf{x})$ represents the true underlying distribution of data (however, only represented by the samples). This is due to the direction of the KL divergence: The generative model is only slightly penalized when putting probability mass on values of $(\boldsymbol{x}, \boldsymbol{z})$ with no support under $q_{\phi,\mathcal{D}}$ [18]. This corresponds to forward KL divergence visualized in (a) in 2.3.

- **Posterior collapse**: It has been observed that part of the latent variable can collapse during the training and become inactive [43, 20, 4], i.e. the optimization gets stuck in an undesirable stable equilibrium. This phenomenon is because the objective ELBO comprises the reconstruction error and the variational regularization term (see eq. 2.14). The reconstruction error is initially relatively weak, and the variational term forces the approximate posterior towards its prior. This can result in many latent units collapsing before learning a proper representation. The [43] alleviate the problem by scheduling the optimization objective such that the variational regularization term is forced to 0 at the beginning of the optimization and then slowly linear ramped up to its value through the optimization. The [20] proposes the method of *free bits*, which ensures that, on average, a certain minimum information is encoded per (group of) latent variable.

- **Disentangling latent factors**: Vannila VAEs often learn a tangled representation of the latent space, making it difficult to interpret or control individual factors in the data. This can limit their usefulness in applications such as image editing or style transfer.

2.4 Hiearchical VAE

A hierarchical variational autoencoder (HVAE) is a generalization of the VAE that introduces multiple layers of latent variables. The first proposal to address the limitations of the vanilla VAE was the use of auxiliary latent variables, as introduced in [27], which were followed up by truly hierarchical structure in [43] and [20]. Standard vanilla VAE contains only one layer of latent variables (fig. 2.6a) in contrast to HVAE, where each latent variable depends only on the previous one (fig. 2.6b). The HVAE can be represented as (almost) any directed acyclic graph and can be represented as DLVM or Bayesian Network with the following joint and posterior distributions:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_{1:T}) = p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}_{1:T}) \prod_{t=1}^{T-1} [p_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{>t})] p_{\boldsymbol{\theta}}(\boldsymbol{z}_T) \tag{2.19}$$

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x}) = q_{\boldsymbol{\phi}}(\boldsymbol{z}_1 \mid \boldsymbol{x}) \prod_{t=2}^{T} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t \mid \boldsymbol{x}, \boldsymbol{z}_{<t}) \tag{2.20}$$

where $\boldsymbol{z}_{<t} = \boldsymbol{z}_{1:t-1}$ represents the sequence of all latent variables $\boldsymbol{z}_1$ up to $\boldsymbol{z}_t$. We can derive the ELBO objective for the HVAE:

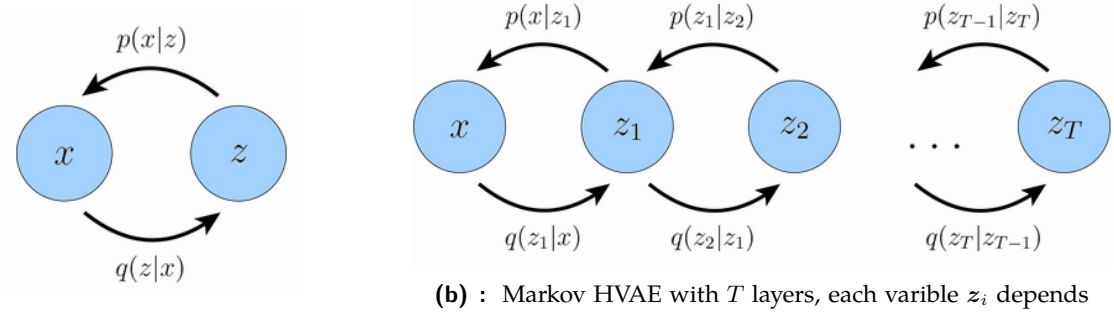
$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \log \int_{\boldsymbol{z}_{1:T}} p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_{1:T}) \mathrm{d}\boldsymbol{z}_{1:T} \\
&= \log \int_{\boldsymbol{z}_{1:T}} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_{1:T}) q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x})} d\boldsymbol{z}_{1:T} \\
&= \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})} \left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_{1:T})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x})} \right] \\
&\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_{1:T})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x})} \right] = \mathrm{ELBO}(q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x}))
\end{aligned} \tag{2.21}$$

We can plug the joint and posterior distribuions from eq. 2.19 and eq. 2.20 into the ELBO term (eq. 2.21) and obtain:

$$\mathrm{ELBO}(q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x})) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}_{1:T}) \prod_{t=1}^{T-1} p_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{>t})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_1 \mid \boldsymbol{x}) \prod_{t=2}^{T} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t \mid \boldsymbol{x}, \boldsymbol{z}_{<t})} \right] \tag{2.22}$$

2.4.1 Markov HVAE

The special case of the HVAE is Markov HVAE, where each variable $\boldsymbol{z}_i$ depends only on the previous variable $\boldsymbol{z}_{i+1}$. This can be seen as multiple VAEs stacked on each other as



(a) : Vanilla VAE

(b) : Markov HVAE with T layers, each varible z_i depends only on previous z_{i+1}, creating a Markov chain.

Figure 2.6: Visualization of vanilla VAE and Markov hiearchical VAE (MHVAE) architecture. Source [26]

represented in fig 2.6b. The joint and posterior distribution of HVAE is then

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_{1:T}) = p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}_1) \prod_{t=1}^{T-1} [p_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1})] p_{\boldsymbol{\theta}}(\boldsymbol{z}_T)$$

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x}) = q_{\boldsymbol{\phi}}(\boldsymbol{z}_1 \mid \boldsymbol{x}) \prod_{t=2}^{T} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1})$$

with objective $\text{ELBO}(q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x}))$:

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}_1) \prod_{t=1}^{T-1} p_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_1 \mid \boldsymbol{x}) \prod_{t=2}^{T} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1})} \right]$$

While MHVAEs can express complex distributions and overcome the issue of blurriness, optimizing them for deep hierarchies can be challenging because of their multiple conditional stochastic layers. This was partially overcome with ladder VAE, which proposes different ordering of the variational encoder to improve the learning.

2.4.2 Ladder VAE

The standard HVAEs (used in [17, 20, 35]), which use a bottom-up encoder and top-down decoder, are difficult to train due to the lack of interaction between the encoder and decoder during inference, as shown in (a) of fig. 2.7. However, the recently proposed ladder VAE (LVAE) by Sønderby et al. [43] addresses this issue by introducing a new inference model that leverages top-down dependencies, similar to the generative model, as depicted in (b) of fig. 2.7. This approximate posterior distribution merges information from a bottom-up approximate likelihood and top-down prior information. By sharing information and parameters with the generative model, the inference model gains access to the current state of the generative model at each layer. The top-down pass recursively corrects the generative distribution with a data-dependent approximate log-likelihood. This inference model can be formally expressed as follows:

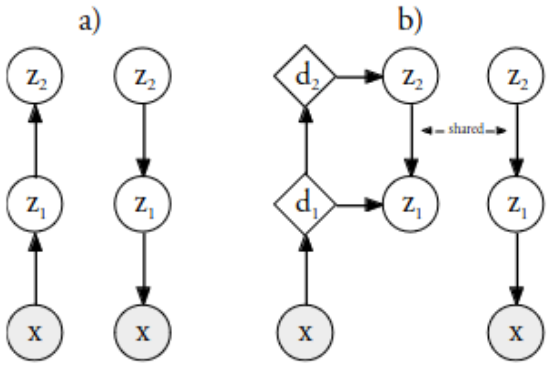


Figure 2.7: Illustration of bottom-up and top-down approaches of the encoder (inference model) and decoder (generative model). In (a), we can see the standard HVAE and in (b), the newly proposed LVAE. Source [43]

$$q_{\phi,\theta}(\boldsymbol{z}_t \mid \boldsymbol{z}_{>t}, \boldsymbol{x}) \sim p_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{>t}) \tilde{q}_{\boldsymbol{\phi}}(\boldsymbol{z}_i \mid \boldsymbol{x}) \tag{2.23}$$

where $\tilde{q}_{\boldsymbol{\phi}}(\boldsymbol{z}_i \mid \boldsymbol{x})$ represents the probability distribution from given family $\mathcal{Q}$ parametrized by (deterministic) encoder:

$$\begin{aligned} \boldsymbol{d}_i &= \text{NeuralNet}_{\boldsymbol{\phi}}(\boldsymbol{d}_{i-1}),\ \boldsymbol{d}_0 = \boldsymbol{x} \\ \tilde{q}_{\boldsymbol{\phi}}(\boldsymbol{z}_i \mid \boldsymbol{x}) &= \tilde{q}_{\boldsymbol{\phi}}(\boldsymbol{z}_i \mid \boldsymbol{d}_i) \end{aligned}$$

The multiplication in eq. 2.23 is a choice of authors but is favourable in case the distributions are members of the exponential family (see 2.5)

2.5 Exponential family

An exponential family is a parametric set of probability distribuions, whose probability densities or masses can be expressed in form:

$$p(\boldsymbol{x} \mid \boldsymbol{\eta}) = h(\boldsymbol{x}) \exp(\boldsymbol{T}(\boldsymbol{x}) \cdot \boldsymbol{\eta} - A(\boldsymbol{\eta})) \tag{2.24}$$

where $h(\boldsymbol{x})$ is a base measure, $\boldsymbol{\eta}$ is vector of *natural parameters*, $\boldsymbol{T}(\boldsymbol{x})$ are *suffient statistics* and $A(\boldsymbol{\eta})$ is *cumulant function* also known as *log normalizer* (see eq. 2.25 for explanation).

Many common distributions, such as normal distribution, categorical distribution, Bernoulli distribution, gamma distribution, Dirichlet distribution, etc., are the exponential family. We show the reparametrization of some distributions so they correspond to eq. 2.24 in table 2.1.

Distribution	θ	η	$h(x)$	$T(x)$	$A(\eta)$	$A(\theta)$
Bernoulli	p	$\log \frac{p}{1-p}$	1	x	$\log(1+e^{\eta})$	$-\log(1-p)$
Categorical	$\begin{bmatrix} p_1 \\ \vdots \\ p_k \end{bmatrix}$	$\begin{bmatrix} \log p_1 \\ \vdots \\ \log p_k \end{bmatrix}$	1	$\begin{bmatrix} [x=1] \\ \vdots \\ [x=k] \end{bmatrix}$	0	0
Gaussian	$\begin{matrix} \mu \\ \sigma^2 \end{matrix}$	$\begin{bmatrix} \frac{\mu}{\sigma^2} \\ -\frac{1}{2\sigma^2} \end{bmatrix}$	$\frac{1}{2\pi}$	$\begin{bmatrix} x \\ x^2 \end{bmatrix}$	$-\frac{\eta_1^2}{4\eta_2} - \frac{\log(-2\eta_2)}{2}$	$\frac{\mu^2}{2\sigma^2} + \log \sigma$

Table 2.1: Some members of the Exponential family. θ represents the standard parameter. Other symbols are described in eq. 2.24.

Cumulant function

Because the $p(\boldsymbol{x} \mid \boldsymbol{\eta})$ is a probability density, the integral of it equals one:

$$\begin{aligned} \int_{\boldsymbol{x}} p(\boldsymbol{x} \mid \boldsymbol{\eta}) \mathrm{d}\boldsymbol{x} &= \int_{\boldsymbol{x}} h(\boldsymbol{x}) \exp(\boldsymbol{T}(\boldsymbol{x}) \cdot \boldsymbol{\eta} - A(\boldsymbol{\eta})) \mathrm{d}\boldsymbol{x} \\ &= \frac{\int_{\boldsymbol{x}} h(\boldsymbol{x}) \exp(\boldsymbol{T}(\boldsymbol{x}) \cdot \boldsymbol{\eta})}{\exp(A(\boldsymbol{\eta}))} \mathrm{d}\boldsymbol{x} = 1 \\ A(\boldsymbol{\eta}) &= \log\Big[\int_{\boldsymbol{x}} h(\boldsymbol{x}) \exp(\boldsymbol{T}(\boldsymbol{x}) \cdot \boldsymbol{\eta})\Big] \end{aligned} \tag{2.25}$$

and therefore the name *log normalizer*. Another interesting property is that the derivative of cumulant function w.r.t. natural parameters is:

$$\frac{d}{d\boldsymbol{\eta}} A(\boldsymbol{\eta}) = \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{\eta})}[\boldsymbol{T}(\boldsymbol{x})]$$

this is easy to see since:

$$\begin{aligned} \frac{d}{d\boldsymbol{\eta}} \int_{\boldsymbol{x}} p(\boldsymbol{x} \mid \boldsymbol{\eta}) \mathrm{d}\boldsymbol{x} &= \frac{d}{d\boldsymbol{\eta}} \int_{\boldsymbol{x}} h(\boldsymbol{x}) \exp(\boldsymbol{T}(\boldsymbol{x}) \cdot \boldsymbol{\eta} - A(\boldsymbol{\eta})) \mathrm{d}\boldsymbol{x} \\ &= \int_{\boldsymbol{x}} \frac{\partial}{\partial \boldsymbol{\eta}} [h(\boldsymbol{x}) \exp(\boldsymbol{T}(\boldsymbol{x}) \cdot \boldsymbol{\eta} - A(\boldsymbol{\eta})) \mathrm{d}\boldsymbol{x}] \\ &= \int_{\boldsymbol{x}} [h(\boldsymbol{x}) \exp(\boldsymbol{T}(\boldsymbol{x}) \cdot \boldsymbol{\eta} - A(\boldsymbol{\eta}))] [\boldsymbol{T}(\boldsymbol{x}) - \frac{d}{d\boldsymbol{\eta}} A(\boldsymbol{\eta})] \mathrm{d}\boldsymbol{x} \\ &= \int_{\boldsymbol{x}} [\boldsymbol{T}(\boldsymbol{x}) - \frac{d}{d\boldsymbol{\eta}} A(\boldsymbol{\eta})] p(\boldsymbol{x} \mid \boldsymbol{\eta}) \mathrm{d}\boldsymbol{x} \\ &= \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{\eta})} [\boldsymbol{T}(\boldsymbol{x}) - \frac{d}{d\boldsymbol{\eta}} A(\boldsymbol{\eta})] = 0 \\ \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{\eta})} [\boldsymbol{T}(\boldsymbol{x})] &= \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{\eta})} [\frac{d}{d\boldsymbol{\eta}} A(\boldsymbol{\eta})] = \frac{d}{d\boldsymbol{\eta}} A(\boldsymbol{\eta}). \end{aligned}$$

The second derivative of the cumulant function with respect to natural parameters is the variance of sufficient statistic [15]:

$$\frac{d^2}{d\boldsymbol{\eta}^2} A(\boldsymbol{\eta}) = \mathbb{V}_{\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{\eta})}[\boldsymbol{T}(\boldsymbol{x})]$$

Another important theorem about the convexity of the exponential family [15] states:

Theorem 2.2. *The natural parameter space $\mathcal{N}$ is convex (as a set), and the cumulant function $A(\boldsymbol{\eta})$ is convex (as a function). If the family is minimal, then $A(\boldsymbol{\eta})$ is strictly convex.*

Kullback Leibler divergence

The KL divergence for two distribuions p and q is defined as:

$$\mathrm{KL}(p(\boldsymbol{x})||q(\boldsymbol{x})) = \int_x p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \mathrm{d}\boldsymbol{x} = \mathbb{E}_{p(\boldsymbol{x})} \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}$$

however for the distributions of the family, one can obtain a closed formula:

$$\begin{aligned} \mathrm{KL}(p(\boldsymbol{x})||q(\boldsymbol{x})) &= \mathbb{E}_{p(\boldsymbol{x})}(\boldsymbol{\eta}_p - \boldsymbol{\eta}_q) \cdot \boldsymbol{T}(\boldsymbol{x}) - A(\boldsymbol{\eta}_p) + A(\boldsymbol{\eta}_q) \\ &= (\boldsymbol{\eta}_p - \boldsymbol{\eta}_q) \cdot \boldsymbol{\mu}_p - A(\boldsymbol{\eta}_p) + A(\boldsymbol{\eta}_q) \end{aligned}$$

where $\mu_p = \mathbb{E}_{p(\boldsymbol{x})}[\boldsymbol{T}(\boldsymbol{x})]$ is the mean parameter and can be obtained through differentiating the cumulant function.

We define the *empirical data distribution*

$$p_{\mathcal{D}} = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}' \in \mathcal{D}} \delta(\boldsymbol{x}, \boldsymbol{x}')$$

where $\delta(\boldsymbol{x}, \boldsymbol{x}')$is Kronecker delta. This distribution places a point mass at each datapoint in dataset $\mathcal{D}$. We can utilize it for writing the log-likelihood (in discrete case):

$$\begin{aligned} \sum_{\boldsymbol{x}} p_{\mathcal{D}} \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) &= \sum_{\boldsymbol{x}} \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}' \in \mathcal{D}} \delta(\boldsymbol{x}, \boldsymbol{x}') \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) \\ &= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}' \in \mathcal{D}} \sum_{\boldsymbol{x}} \delta(\boldsymbol{x}, \boldsymbol{x}') \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) \\ &= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}' \in \mathcal{D}} \log p(\boldsymbol{x}' \mid \boldsymbol{\theta}) \\ &= \frac{1}{|\mathcal{D}|} l(\boldsymbol{\theta} \mid \mathcal{D}) \end{aligned}$$

where $l(\boldsymbol{\theta} \mid \mathcal{D}) = \log p(\mathcal{D} \mid \boldsymbol{\theta})$ is the log likelihood. So computing the cross entropy between empirical data distribution and the model provides us with log-likelihood. If we compute the KL divergence of the empirical data distribution and model $p(\boldsymbol{x} \mid \boldsymbol{\theta})$, we obtain

$$\mathrm{KL}(p_{\mathcal{D}}||p(\boldsymbol{x} \mid \boldsymbol{\theta})) = \sum_{\boldsymbol{x}} p_{\mathcal{D}} \frac{p_{\mathcal{D}}}{p(\boldsymbol{x} \mid \boldsymbol{\theta})} = \mathbb{E}_{p_{\mathcal{D}}} \log p_{\mathcal{D}} - \frac{1}{N} l(\boldsymbol{\theta} \mid \mathcal{D})$$

the empirical data distribution is not depending on the model parameters $\boldsymbol{\theta}$ and thus *by minimizing the KL divergence to the empirical distribution, we maximize the (log) likelihood (of data under the model)*.

2.6 Symmetric leaning in VAE

The authors of [40] present an alternative approach to maximizing the evidence lower bound (ELBO) in Variational Autoencoders (VAEs). The traditional ELBO optimization imposes restrictions on the architectures of VAEs, as it requires the latent distributions to be in closed form while only providing data samples. This asymmetry in the ELBO formulation contributes to the issue of blurriness in generated images (discussed in 2.3.4), which has been partially addressed by methods like normalizing flows [34] and LVAE [43].

The proposed symmetric learning approach relaxes these restrictions and enables VAE learning when both the data and latent distributions are accessible only through sampling. This approach also applies to more complex models, such as Hierarchical VAEs (HVAEs), and leads to simpler algorithms for training. The experiments provided in the paper show that models obtained from this training approach are comparable to those achieved through ELBO learning.

In the standard VAE framework, we train the encoder and decoder through maximizing the ELBO objective, i.e. given the true underlying distribution of data $\pi(\boldsymbol{x}), \boldsymbol{x} \in \mathcal{X}$ and underlying the distribution in latent variable $\pi(\boldsymbol{z}), \boldsymbol{z} \in \mathcal{Z}$, we maximize ELBO:

$$\mathcal{L}_B = \text{ELBO} = \mathbb{E}_{\pi(\boldsymbol{x})}[\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) - \text{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})||\pi(\boldsymbol{z}))]$$

in order to obtain the pair of encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ and decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$. It is necessary to define the model distribution $p(\boldsymbol{z})$ in closed form to keep the computation of KL divergence tractable. This is an issue in case the $\pi(\boldsymbol{z})$ is complex, and we cannot model it by a simple distribution family. Another necessity is that the $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ allow the *reparametrization trick*.

The authors propose a new algorithm for learning the encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ and decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ in case of *semi-supervised* and *unsupervised* learning:

- *Semi-supervised learning*: We can draw i.i.d samples from underlying unknown distributions $(\boldsymbol{x}, \boldsymbol{z}) \sim \pi(\boldsymbol{x}, \boldsymbol{z})$ and its marginals: $\boldsymbol{x} \sim \pi(\boldsymbol{x})$, $\boldsymbol{z} \sim \pi(\boldsymbol{z})$.
- *Unsupervised learning*: We can draw only $\boldsymbol{x} \sim \pi(\boldsymbol{x})$. The latent space is modelled through the choice of model $p(\boldsymbol{z})$.

The encoder and decoder belong to the exponential family and allow for tractable computation of log density and its derivatives.

$$p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) \propto \exp[\boldsymbol{\Theta}(\boldsymbol{x}) \cdot \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z})]$$
$$p_{\boldsymbol{\phi}}(\boldsymbol{x} \mid \boldsymbol{z}) \propto \exp[\boldsymbol{\Phi}(\boldsymbol{z}) \cdot \boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x})]$$

where $\boldsymbol{\Theta}(\boldsymbol{x}) \in \mathbb{R}^n$ and $\boldsymbol{\Phi}(\boldsymbol{z}) \in \mathbb{R}^m$ are sufficient statistics. The variables $\boldsymbol{x}$ and $\boldsymbol{z}$ can be either discrete or continuous depending on the choice of an exponential family (e.g. Bernoulli or Gaussian).

The authors provide a new optimization function, which is motivated by finding a *Nash equilibrium* for a two-player game where players' strategies are represented through the encoder and decoder distributions, respectively and the utility function is a sum of the player expectation w.r.t his strategy [40]. The objectives are

$$\mathcal{L}_p(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\pi(\boldsymbol{x},\boldsymbol{z})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right] + \mathbb{E}_{\pi(\boldsymbol{z})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{z})\right] + \mathbb{E}_{\pi(\boldsymbol{x})}\,\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right]$$
$$\mathcal{L}_q(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\pi(\boldsymbol{x},\boldsymbol{z})}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right] + \mathbb{E}_{\pi(\boldsymbol{z})}\,\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right]$$

for semi-supervised training and

$$\mathcal{L}_p(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\pi(\boldsymbol{x})}\,\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right] \tag{2.26}$$
$$\mathcal{L}_q(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right] \tag{2.27}$$

for unsupervised training with the following interpretation: We maximize the decoder and encoder likelihood of the training data simultaneously. The mixed terms reinforce the encoder-decoder consistency. This corresponds to the maximization of the ELBO objective since we can rewrite the ELBO into:

$$\mathbb{E}_{\pi(\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - \mathrm{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) || p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}))\right]$$

After inspecting the terms, we see that the ELBO goal is the same as above: To maximize the data likelihood and reinforce the consistency of the decoder-encoder pair simultaneously.

2.6.1 Hiearchical VAEs

The algorithm can also be adopted for hierarchical VAE. Let us assume that we have HVAE with $M+1$ layers, i.e. $\boldsymbol{z}$ consists of $\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_m$, where the encoder models correspond to the LVAE, and we can sample $\boldsymbol{x} \sim \pi(\boldsymbol{x})$. The encoder and decoder factorize (the ordering is in reverse to the one in eq. 2.19 and eq. 2.20):

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}_0) \prod_{t=1}^{M} [p_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{<t})] p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$$
$$q_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z}) = \pi(\boldsymbol{x}) q_{\boldsymbol{\phi}}(\boldsymbol{z}_0 \mid \boldsymbol{x}) \prod_{t=1}^{M} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{<t}, \boldsymbol{x})$$

where the encoder shares the parameter as described in 2.23, the objectives remain as in unsupervised case (eq. 2.26 and eq. 2.27). The terms can be decomposed due to the factorization of decoder and encoder and are thus tractable. If there is access to the samples $(\boldsymbol{x}, \boldsymbol{z}_0) \sim \pi(\boldsymbol{x}, \boldsymbol{z}_0)$, e.g. segmentation task with target masks, we can utilize them by adding terms

$$\mathbb{E}_{\pi(\boldsymbol{x},\boldsymbol{z}_0)}\,\mathbb{E}_{q(\boldsymbol{z}_{>0}|\boldsymbol{z}_0,\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) \quad \text{and} \quad \mathbb{E}_{\pi(\boldsymbol{x},\boldsymbol{z}_0)} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_0 \mid \boldsymbol{x})$$

to the decoder and encoder, respectively.

Chapter 3

Methods

3.1 Problem definition

Our main objective is to develop an effective approach for segmenting satellite imagery using a limited number of accessible images and a smaller number of expert-provided annotations. The segmentation task involves assigning a specific label to each pixel in the image, effectively partitioning the image into distinct regions based on their semantic or visual characteristics.

To achieve this objective, we will employ the U-Net architecture, which has been proven to be effective for image segmentation [37]. The U-Net architecture will serve as the core model for evaluating the proposed algorithms. We will use the plain accuracy and average "intersection over union" (IoU) metrics to evaluate the quality of the segmentation. The IoU measures the overlap between the predicted segmentation and the ground truth segmentation, i.e.:

$$\mathrm{IoU} = \frac{\text{area of intersection}}{\text{area of union}},$$

where the area of intersection corresponds to the area where the model predicts the given class and the class is in the ground truth segmentation. The union area is then composed of an area where either model predicts the class or the class is in the ground truth segmentation (or both). This can be rephrased to binary classification terminology as:

$$\mathrm{IoU} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}},$$

where the TP is *true positive*, FP is *false positive* and FN is *false negative*. The average IoU is then the simple average of the class IoU over all classes.

To evaluate our methods and ensure reproducible experiments, we will utilize the CityScape dataset [6] as a benchmark. The CityScape dataset is almost publicly available,

with a fee-less registration required to access. Although the dataset primarily consists of urban street scenes, we believe that the complexity of segmentation in this dataset is comparable to, or even higher than, the challenges encountered in land-coverage segmentation of forests.

By leveraging the CityScape dataset and employing the U-Net architecture, we can thoroughly evaluate our methods and provide meaningful results that others can replicate and validate.

The U-Net is a feed-forward convolutional neural network (FF CNN), and its architecture consists of an encoder and a decoder part, which work together for image segmentation tasks. (We would like to point out to the reader that in this context, the "encoder" and "decoder" are not referring to the VAE networks but to the structure of the U-Net architecture):

- *Encoder*: The encoder part of U-Net is responsible for downscaling the spatial resolution of the input image while increasing its channel capacity. This is achieved through a series of convolutional blocks and downsampling. Each block typically consists of one or more convolutional layers, followed by non-linear activation functions (such as ReLU) and batch normalization. The encoder acts as a feature extractor for image classification tasks.
- *Decoder*: The decoder part of U-Net is connected to the encoder and performs the opposite operation of the encoder. It upscales the feature maps while reducing the number of channels. This is done through a series of upsampling and convolutional blocks. Like the encoder, each block consists of convolutional layers, activation functions, and batch normalization.
- *Skip Connections*: U-Net incorporates skip connections between the encoder and decoder. These connections allow information from the encoder to be directly propagated to the corresponding decoder block at the same spatial resolution. By sharing this information, U-Net helps to preserve spatial details and enables more precise segmentation.
- *Convolutional Blocks*: The convolutional blocks in U-Net typically consist of convolutional layers, activation functions, and batch normalization. The number of convolutional layers in each block can vary but is usually up to 2 or 3. These blocks are responsible for learning and extracting features from the input data. Their parameters are typically chosen such that they retain the spatial resolution
- *Spatial Resolution*: U-Net can maintain the spatial resolution of the input or perform downsampling and upsampling operations at specific layers, depending on the chosen parameters. This flexibility allows U-Net to capture both local and global information.

Overall, the U-Net architecture is designed to effectively capture contextual information

and spatial details for image segmentation tasks. It has been widely used and has shown promising results in various applications.

3.2 Mixmatch adaptation

The current implementation of the MixMatch algorithm is limited as it only allows for augmentations that do not alter the labels of given features. However, this limitation poses a problem for image segmentation tasks because most commonly used augmentations apply spatial transformations that affect image segmentations. To address this issue, we propose an extension of the MixMatch algorithm specifically designed for image segmentation tasks.

Our approach involves adapting the data augmentation and proxy-labeling procedure to allow for augmentations that modify the corresponding segmentation masks. Specifically, we suggest using affine transformations for the data augmentations, as they are easily invertible and widely applicable.

To combine predictions across multiple views of an image, we align the predictions with the original image by applying the inverse augmentation and then apply the original augmentation to assign the averaged predictions to each view. However, it is important to note that care must be taken while averaging because a part of the original image may be cropped while keeping the spatial dimensions. Thus, the segmentation prediction is only valid on the uncropped region of the image.

Finally, the MixUp procedure requires adaptation to ensure that cropped images are correctly mixed without propagating the empty parts further down the stream while keeping the output of MixMatch $(\mathbf{x}', \mathbf{p}')$ closer to the first argument $(\mathbf{x}^1, \mathbf{p}^1)$.

The proposed changes have been incorporated into algorithm 4 and are commented on there. In MixUp, we ensure that the cropped-out parts of the images are suppressed so that they do not influence the MixUp process. Additionally, we aim to keep the output image closer to the first argument. The modified MixUp procedure is as follows:

$$
\begin{aligned}
&\lambda \sim \text{Beta}(\alpha, \alpha) \\
&\lambda' = \max(\lambda, 1-\lambda) \\
&\boldsymbol{\lambda}'_{i,j} = \begin{cases} \lambda' & \text{if } \mathbf{x}^1_{i,j} \text{ and } \mathbf{x}^2_{i,j} \text{ are valid} \\ 1 & \text{if } \mathbf{x}^1_{i,j} \text{ is not valid or } \mathbf{x}^2_{i,j} \text{ is not valid} \end{cases} \\
&\mathbf{x}' = \boldsymbol{\lambda}' \odot \mathbf{x}^1 + (\mathbf{1} - \boldsymbol{\lambda}') \odot \mathbf{x}^2 \\
&p' = \boldsymbol{\lambda}' \odot \mathbf{p}^1 + (\mathbf{1} - \boldsymbol{\lambda}') \odot \mathbf{p}^2.
\end{aligned}
\tag{3.1}
$$

Here, $\odot$ represents element-wise multiplication, and the upper index is used for indexing. The alternation of $\boldsymbol{\lambda}'$ ignores the invalid parts of the second input and retains the original

Algorithm 4 MixMatch adapted for segmentation

Input: Batch of labeled examples and their segmentation masks $X = ((\mathbf{x}_i, \mathbf{p}_i); i \in (1, \dots, n))$, batch of unlabeled examples $U = (\mathbf{u}_i; i \in (1, \dots, n))$, sharpening temperature T, number of augmentations K, Beta distribution parameter α for MixUp.
for $i = 1$ **to** n **do**
  $\bar{\mathbf{x}}_i, \bar{\mathbf{p}}_i = \text{Augment}(\mathbf{x}_i, \mathbf{p}_i)$ ▷ We apply augmentation both to $\mathbf{x}_i$ and $\mathbf{p}_i$
  for $k = 1$ **to** K **do**
    $\bar{\mathbf{u}}_{i,k} = \text{Augment}_k(\mathbf{u}_i)$
    $\bar{\mathbf{q}}_{i,k} = p_{\text{model}}(y|\bar{u}_{i,k}; \theta)$
  end for
  $\tilde{\mathbf{q}}_{i,k} = \text{Inverse Augment}_k(\bar{\mathbf{q}}_{i,k})$ ▷ Align prediction with original image
  $\bar{\mathbf{q}}_i = \frac{1}{K} \sum_{k=1}^{K} \tilde{\mathbf{q}}_{i,k}$
  $\mathbf{q}_i = \text{Sharpen}(\bar{\mathbf{q}}_i, T)$
  for $k = 1$ **to** K **do**
    $\mathbf{q}_{i,k} = \text{Augment}_k(\mathbf{q}_i)$ ▷ Rematch the average prediction to augmented image
  end for
end for
$X^\star = ((\bar{\mathbf{x}}_i, \mathbf{p}_i); i \in (1, \dots, n))$
$U^\star = ((\bar{\mathbf{u}}_{i,k}, \mathbf{q}_{i,k}); i \in (1, \dots, n), k \in (1, \dots, K))$
$W = \text{Shuffle}(\text{Concat}(X^\star, U^\star))$
$X' = (\text{MixUp}(\bar{\mathbf{x}}_i, \mathbf{w}_i); i \in (1, \dots, |X^\star|))$ ▷ Apply MixUp described in 3.1
$U' = (\text{MixUp}(\bar{\mathbf{u}}_i, \mathbf{w}_{i+|X^\star|}); i \in (1, \dots, |U^\star|))$
return X', U'

picture with its invalid parts. This modification aims to keep the output of the MixMatch procedure close to the input, as we have inputs of uneven quality, i.e., labeled and unlabeled data.

3.3 Symetric learning for HVAE

We implement a hierarchical variational autoencoder (HVAE) that aims to provide a segmentation $\mathbf{s}$ for a given image $\mathbf{x}$ and reconstruct the image if a ground truth segmentation is provided. In our scenario, we only have access to samples from the unknown underlying distribution of images and segmentations, denoted as $\mathbf{x}, \mathbf{s} \sim \pi(\mathbf{x}, \mathbf{s})$.

The hierarchical VAE consists of $M + 1$ layers of latent space:

$$\mathbf{z} = (\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_m), \quad \mathbf{z}_0 = (\mathbf{s}, \mathbf{l}), \; \mathbf{z}_m = \mathbf{x}$$

Here, $\mathbf{z}_m$ corresponds to the image, and $\mathbf{z}_0$ is the composition of the segmentation $\mathbf{s}$ and the latent code $\mathbf{l}$, which follows a uniform prior distribution. The latent code is expected to encode global information in the image (e.g., weather, texture), while the segmentation

provides local information (e.g., road shape, surrounding types, pedestrians, cars, etc.). The hierarchical model consists of an encoder and a decoder with a common factorization:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}_0) \prod_{t=1}^{M} [p_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{<t})] p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$$

$$q_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z}) = \pi(\boldsymbol{x}) q_{\boldsymbol{\phi}}(\boldsymbol{z}_0 \mid \boldsymbol{x}) \prod_{t=1}^{M} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{<t}, \boldsymbol{x})$$

Both the encoder $q_{\boldsymbol{\phi}}$ and decoder $p_{\boldsymbol{\theta}}$ have a U-Net-like architecture. Since the U-Net is a feedforward network, sampling from the decoder is tractable, and we can construct the HVAE. The conditional models in the HVAE belong to different distributions within the exponential family, depending on the specific case. The hidden layers $\mathbf{z}_1, \ldots, \mathbf{z}_{m-1}$, as well as the latent code $\mathbf{l}$, are modeled using a Bernoulli distribution. For the image and segmentation, we utilize Gaussian and categorical distributions, respectively.

We utilize the U-net architecture's standard building blocks, consisting of convolutional layers and rescaling layers. These blocks are applied up to the last (first) layer of the encoder (decoder), due to the $\mathbf{z}_0$ choice. We wish the segmentations $\mathbf{s}$ to contain the local information corresponding to its spatial position and the latent code $\mathbf{l}$ to contain global information at the same time. To achieve this, we adapt the architecture of the last block of both the encoder and decoder:

- The last block of the *encoder* in our architecture comprises a shared core network, which is common to both the segmentation and latent code branches. Following the core network, there are two independent heads (networks) with separate parameters, each tailored to produce the desired outputs.

  The network architectures for heads and core networks follow the standard convolutional block structure. In the segmentation head, the final layer is a convolutional layer with output channels corresponding to the number of classes in the segmentation task. This layer generates the segmentation predictions.

  On the other hand, the latent code head concludes with an adaptive average pooling layer. This layer reduces the spatial dimensions of the output to 1, resulting in a compact latent code representation that captures global information.

- The first block of the *decoder* is composed of a single convolutional block. Initially, we replicate the latent code activations in the spatial dimension to match the segmentation's spatial dimensions. Next, we concatenate the replicated latent code and the segmentations along the channel dimension, creating a compact block, which serves as the input to the aforementioned convolutional block in the decoder.

  By replicating the latent code in the spatial dimension and combining it with the segmentations, we ensure that the latent code $\mathbf{l}$ possesses complete information about the image and has a global influence.

The overall architecture of both the encoder and decoder is inspired by the U-net architecture, incorporating skip connections. The architecture exhibits a "symmetry"

between the encoder and decoder, defined as follows: a skip connection exists from $\mathbf{z}_i$ to $\mathbf{z}_j$ in the decoder if and only if there is a skip connection from $\mathbf{z}_j$ to $\mathbf{z}_i$ in the encoder.

This architecture is our choice. In fact, HVAE permits any set of skip connections between the $\mathbf{z}$ layers for the decoder, as long as a topological ordering exists for such a set. In other words, the corresponding directed graph formed by the m nodes representing layers $\mathbf{z}_1, \ldots, \mathbf{z}_m$ should be acyclic. Moreover, it is worth noting that the encoder and decoder architectures are independent of each other up to the fact that the encoder should supply the appropriate activations to the decoder layers, following the LVAE paradigm.

In the case of semi-supervised learning, the HVAE is trained using block-wise maximization of the following objective functions:

$$\mathcal{L}_p(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\pi(\mathbf{x},\mathbf{s})}\, \mathbb{E}_{q_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{z}_{>0},\mathbf{l}|\mathbf{x},\mathbf{s})} \big[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\big] + \mathbb{E}_{\pi(\mathbf{x})}\, \mathbb{E}_{q_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \big[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\big] \tag{3.2}$$

$$\mathcal{L}_q(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\pi(\mathbf{x},\mathbf{s})} \big[\log q_{\boldsymbol{\phi}}(\mathbf{s} \mid \mathbf{x})\big] + \mathbb{E}_{\pi(\mathbf{s})}\, \mathbb{E}_{p(\mathbf{l})}\, \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}_{>0}|\mathbf{z}_0)} \big[\log q_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{z}_{>0}|\mathbf{x})\big] \tag{3.3}$$

Here, $\pi(\mathbf{x}, \mathbf{s})$ represents the underlying distribution with marginals $\pi(\mathbf{x})$ and $\pi(\mathbf{s})$. The distribution $p(\mathbf{l})$ is a uniform prior distribution for the latent code (model choice). The first terms in both objectives correspond to supervised learning, while the second terms correspond to unsupervised learning.

During training, we sample from the corresponding distributions and compute the stochastic gradient as in the standard VAE framework. The samples from $\pi(\mathbf{x}, \mathbf{s})$ and $\pi(\mathbf{x})$ are provided in the labeled dataset $\mathcal{D}_l$ and unlabeled dataset $\mathcal{D}_u$, respectively. If there is available segmentation information, we can incorporate it in the unsupervised learning term of the encoder. In our case, we have access to segmentation information. However, it is often not available in practice, requiring the introduction of the model distribution $p(\mathbf{s})$ from which we can sample.

In the case of unsupervised learning, we drop the terms corresponding to supervised learning from eq. 3.2 and eq. 3.3.

Chapter 4

Experiments & Results

4.1 Mixmatch experiments

As the original implementation of MixMatch is written in `TensorFlow` we have re-implemented it in `PyTorch`. To verify the correctness of our implementation, we have run tests on CIFAR10 [22] and compared our results with those reported in [2] and [51]. The comparison is shown in table 4.1. For details of reported results, see respective papers. The specific hyperparameters of all experiments are provided with the code and are available. We will not report any of them here.

Labels [#]	250	500	1000	2000	4000	All
Our code	88.45	89.58	91.83	93.03	93.50	93.54
Reported	88.92 ± 0.87	90.35 ± 0.94	92.25 ± 0.32	92.97 ± 0.15	93.76 ± 0.06	94.27
Baseline	38.42	45.77	50.42	60.21	79.57	

Table 4.1: Mixmatch accuracy rate (%) on CIFAR10 dataset. Labels row corresponds to a number of labeled points available during training. The last column ("All") corresponds to fully-supervised mode performance on the whole dataset (50k images). We additionally provide the baseline row, which contains the result of supervised training on a given number of images (Our code). Our results are based only on one run

We have conducted another experiment on CityScape dataset. The classes and labelling policy of the CityScape dataset are well described in [6] and on their website. In our setting, we want to predict only seven valid classes and one void class corresponding to the CityScape "categories". The prediction on the void class is ignored during the training and evaluation. In tab. 4.2, we report the plain accuracy for the MixMatch and supervised baseline and in tab. 4.3, we provide the average IoU metric. The MixMatch hyperparameters were found with the help of `Optuna` framework [1]. The applied

augmentation is a simple combination of padding, crop and horizontal flip as in original paper [2], even though the framework can use any affine transformations. The original images are rescaled to the 256×512 size. We provide a few images for visualization in fig. 4.1 and fig. 4.2. MixMatch tends to "regularize and smooth" the predictions. However, this can be a disadvantage for unbalanced datasets like CityScapes, as the network may ignore classes with a small representation ratio.

Labels [#]	10	100	500	1000	All
Mixmatch	84.53	90.76	93.42	94.32	94.84
Supervised	76.85	87.59	93.50	94.71	95.58

Table 4.2: Mixmatch accuracy rate (%) on CityScape dataset compared to the supervised baseline. The "Labels" row indicates the number of labeled points available for training across eight class categories. The last column ("All") represents the performance of the fully-supervised mode on the entire dataset consisting of 2975 images. In this case, the MixMatch algorithm utilizes the entire training dataset for labeled and unlabeled data. Our results are based only on one run.

We can see that the MixMatch improve the results for the small amount of data and can produce comparable results to the supervised baseline for more images available. However, it has been observed that the best results achieved were obtained during the early stage of training for low λ_u parameter (which is linearly ramped up). This means that the unsupervised loss term $\mathcal{L}_{\mathcal{U}}$ worsens the learning. As its primary role is to regularize the model, we believe that the worsening effect is mainly caused by the fact that the model is not yet fully trained on labeled dataset $\mathcal{X}$. This could also fit well with better accuracy for the small size of $\mathcal{X}$ as the model has enough capacity to obtain 100% training accuracy. As the MixMatch is a complex model, it is hard to determine which component was most significant in this task. The paper does provide an ablation study [2]. However, I was not able to conduct a similar study for this experiment due to the limited time available. We would also like to note that the supervised experiment utilized the same augmentation as the MixMatch (on the labeled dataset only) and that even a small number of images contain much information, which can be distilled by the network (contrary to the classification task) since we use the convolutional neural network (CNN).

Labels [#]	10	100	500	1000	All
Mixmatch	48.10	61.98	69.24	71.97	73.08
Supervised	41.78	54.38	68.25	71.93	73.84

Table 4.3: Mixmatch average IoU (%) on CityScape dataset compared to the supervised baseline. The "Labels" row indicates the number of labeled points available for training across 8 class categories. The last column ("All") represents the performance of the fully-supervised mode on the entire dataset consisting of 2975 images. The MixMatch algorithm utilizes the entire training dataset for both labeled and unlabeled data in this case. Our results are based only on one run.

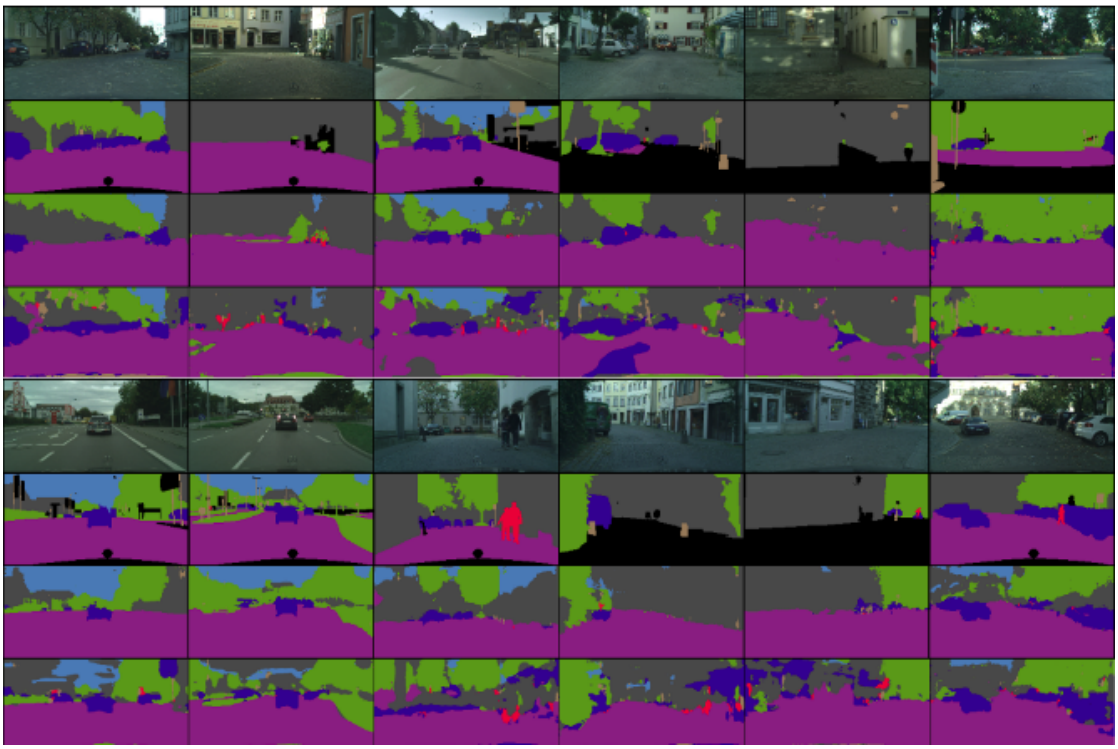


Figure 4.1: Evaluation of MixMatch and supervised baseline on CityScape for eight classes. Models were trained on ten images. The figure contains 12 examples stacked into two blocks, each containing four rows. The first two rows contain the image and its ground truth segmentation. The MixMatch predictions are shown in the third row, while the supervised baseline predictions are in the fourth.
The colors used in the segmentation are as follows: flat (purple), human (red), vehicle (dark blue), construction (dark grey), object (light gray), nature (green), sky (light blue), and void (black). The predictions on the "void" class are not penalized nor evaluated.

4.2 Symmetric learning for HVAE

In our experiment on the CityScape dataset, we adopt the following approach. We divide the available dataset, which consists of images and their corresponding segmentations into three distinct datasets:

- Labeled dataset $\mathcal{D}_l$: This dataset is a subset of the available dataset and contains both the images and their corresponding segmentations.
- Unlabeled dataset $\mathcal{D}_u^1$: This dataset includes the remaining images from the available dataset that were not included in the labeled dataset.
- Unlabeled dataset $\mathcal{D}_u^2$: This dataset comprises the remaining segmentations from the available dataset that were not included in the labeled dataset.

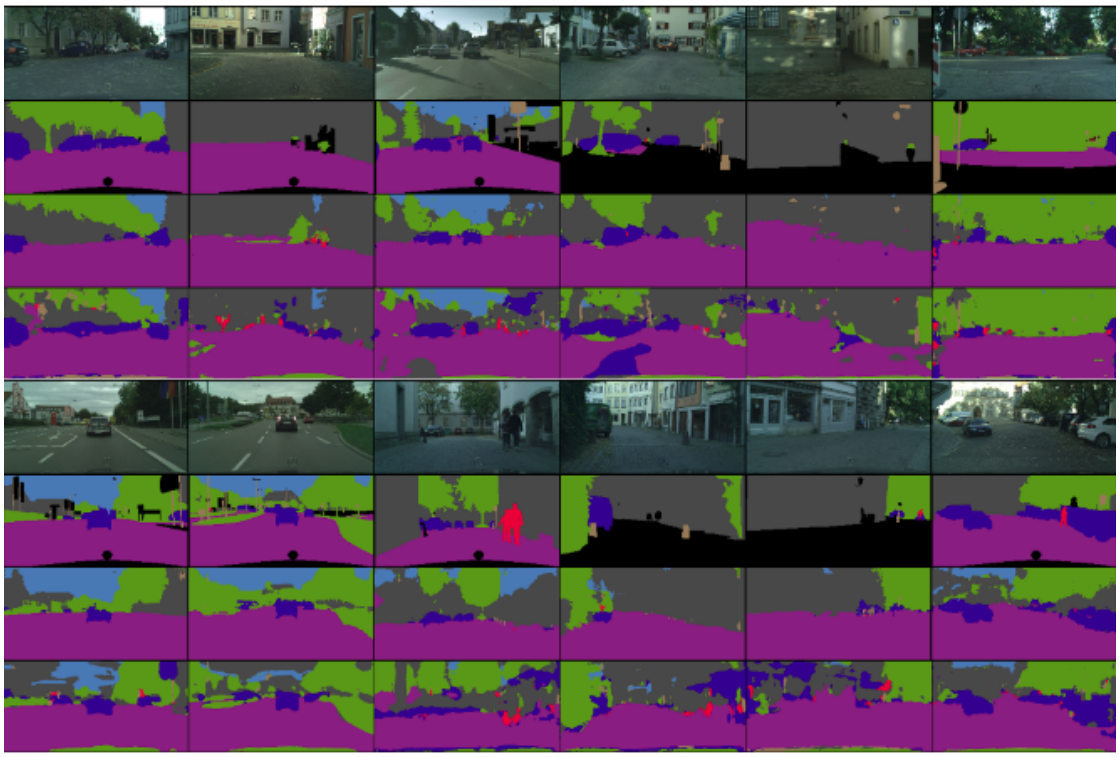


Figure 4.2: Evaluation of MixMatch and supervised baseline on CityScape for eight classes. Models were trained on all available images. See the caption of fig. 4.1 for the legend.

We do not discard the segmentations in our scenario, unlike the MixMatch experiments. Instead, we utilize the segmentations as described in Section 3.3. The achieved accuracy for the symmetric learning of HVAE is provided in tab. 4.4. When comparing the HVAE's accuracies with the supervised baseline from tab. 4.2, it is important to note that the HVAE's encoder architecture is not the same as the supervised baseline's. Specifically, they differ in the number of channels in their respective layers. Also, contrary to the baseline, the HVAE does not include batch normalization (BN) layers. This BN absence arises from the experimental observation that batch normalization is unsuitable for the HVAE. Figure 4.3 presents the cherrypicked visualizations of HVAE segmentations and reconstructed images. The decoder demonstrates the ability to distinguish the brightness and shapes of objects. However, it lacks the capability to reconstruct the colors of the original images.

Labels [#]	10	100	500	1000	All
HVAE	72.33	85.42	89.77	89.77	93.01

Table 4.4: HVAE accuracy rate (%) on CityScape dataset for epoch 180. The "Labels" row indicates the number of labeled points available for training across eight class categories, e.i. size of the dataset $\mathcal{D}_l$. The last column ("All") represents the performance of the fully-supervised mode on the entire dataset consisting of 2975 images

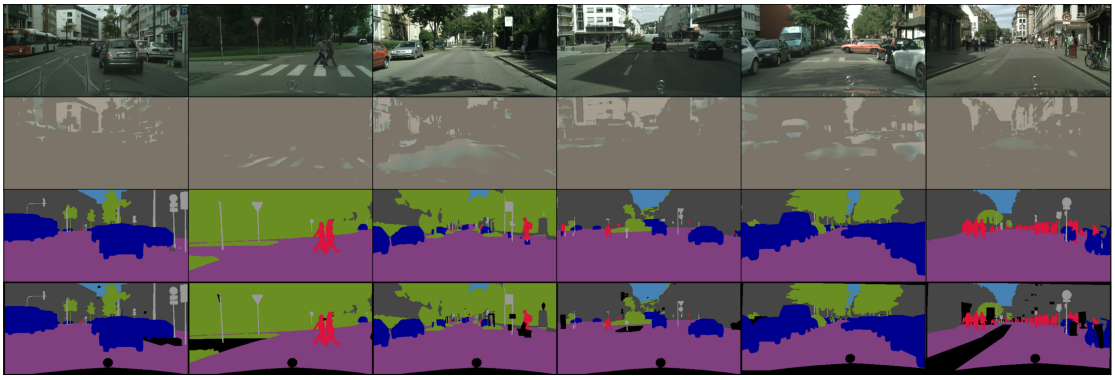


Figure 4.3: The cherrypicked reconstructions and segmentations of HVAE on CityScape for eight classes. The first row contains the original image. The second row is filled with the reconstructed images from encoding the original image into $\mathbf{z}_0$ and decoding. The third and fourth rows contain model and ground truth segmentation, respectively. See the caption of fig. 4.1 for the segmentation legend.

Known issues

During our experiments with symmetric learning on HVAE, we encountered a number of issues. Specifically, we observed that when the latent variables $\mathbf{z}$ had nontrivial spatial dimensions, the generated images often lacked global spatial coherence. Instead, they consisted of multiple locally coherent patches that failed to accurately represent the target images. This issue is visualized in Figure 4.4. However, we found that the problem of global coherence could be mitigated by introducing global skip connections, similar to those used in the U-Net architecture. This observation is supported by the absence of such patches in the reconstructed images shown in Figure 4.3.

Additionally, we have encountered issues related to the decoder blocks situated between the layers, i.e. neural networks predicting the natural parameters:

$$\begin{aligned} \boldsymbol{\eta} &= \mathrm{NeuralNet}_{\boldsymbol{\theta}}(\mathrm{Pa}(\mathbf{z}_{<t})) \\ p_{\boldsymbol{\theta}}(\mathbf{z}_t \mid \mathbf{z}_{<t}) = p_{\boldsymbol{\theta}}(\mathbf{z}_t \mid \boldsymbol{\eta}) &= p_{\boldsymbol{\theta}}(\mathbf{z}_t \mid \mathrm{NeuralNet}_{\boldsymbol{\theta}}(\mathrm{Pa}(\mathbf{z}_t))) \end{aligned}$$

We have observed that the decoder blocks need to be deep enough and simultaneously incorporate simple local skip connections (similar to those used in ResNet architectures) to be able to learn meaningful representations. Specifically, if we introduce too shallow networks into first blocks of decoder, the decoder can not provide any reasonable reconstruction and outputs only the constant noise.

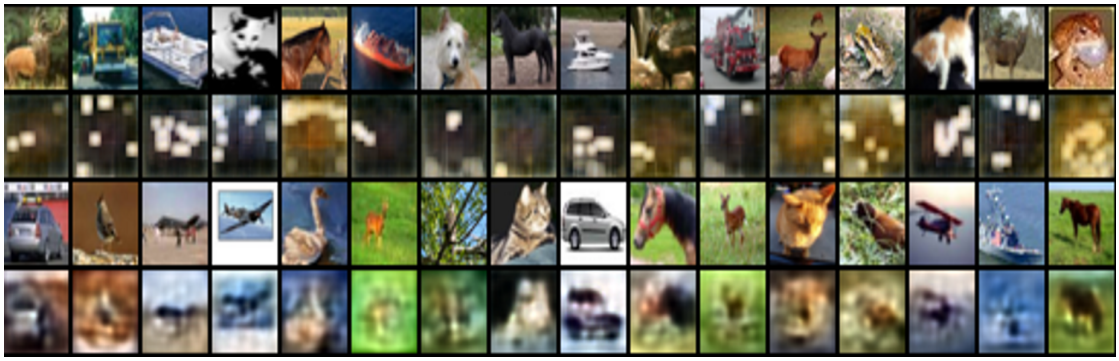


Figure 4.4: CIFAR10 experiments for HVAE. The first and third rows contain the original images. The second and fourth rows contain the reconstructed images. The model used for the second row has latent variables $\mathbf{z}$ with a nontrivial spatial dimension. We can see that the reconstructed images for this model contain blobs. The spatial resolution in latent variables prohibits the decoder from learning the overall image., This results in a significant drop in encoder accuracy and failure of the process. The model in the fourth row has (almost) identical architecture but without latent variables with a spatial dimension. We can see that the decoder is able to learn the overall image, which enables the encoder to classify correctly. This positive feedback allows for the algorithm to be successful.

Chapter 5

Conclusion

Throughout this thesis, we have conducted a thorough investigation and comparison of two fundamental approaches: MixMatch and the novel symmetrical equilibrium learning algorithm within the context of segmentation tasks.

We acknowledge that our deviation from the original assignment, specifically the absence of experiments conducted on the actual time series of multi-spectral satellite data for the national park Bohemian Switzerland, may raise questions regarding the validity and applicability of our findings to this specific task. However, we would like to provide a comprehensive explanation for this decision and offer reasoning as to why our experiments and obtained results remain valuable and relevant.

Instead of utilizing satellite imagery, we chose to work with the (almost) publicly available CityScape dataset, which is widely used and established. This decision was motivated by two factors: ensuring reproducibility and saving time. Although the CityScape dataset primarily consists of urban street scenes, we believe that the complexity of segmentation in this dataset is comparable to, or even higher than, the challenges encountered in land-coverage segmentation of forests. Additionally, we consider the convolution process, whether in 2D or 3D, to be fundamentally similar from the perspective of the network and training process. Therefore, we can extrapolate the results obtained on the CityScape dataset to the task of land cover classification.

In light of these considerations, we focused on exploring and evaluating the effectiveness of MixMatch and Symmetric Equilibrium Learning for VAE for the segmentation task in general. We aimed to gain insights into their applicability and performance by examining their advantages and shortcomings.

Our experiments with MixMatch have demonstrated its efficacy, especially in scenarios where labeled data are scarce. The integration of consistency regularization and proxy-

labeling methods in MixMatch have proven effective in leveraging unlabeled data to enhance segmentation performance and promote visually coherent model predictions.

The experiments for symmetrical learning of HVAE have shown that a U-net-like architecture, with slight adaptations, can be employed for the segmentation task. Specifically, including global skip connections is crucial in enabling the HVAE decoder to generate consistent images. The current results do not outperform the best obtainable supervised baseline, and although there is room for further improvement in the HVAE architecture, the current results nevertheless serve as proof of concept.

The experiments have also raised questions that warrant further investigation to improve the HVAE architecture. One such question pertains to the observed disregard of color information in the decoded images. Understanding the underlying reasons for this phenomenon and its relationship to the performance of the encoder and decoder is essential to improve the overall performance of the HVAE model.

References

[1] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[2] David Berthelot et al. "MixMatch: A Holistic Approach to Semi-Supervised Learning". In: *CoRR* abs/1905.02249 (2019). arXiv: 1905.02249. URL: http://arxiv.org/abs/1905.02249.

[3] David Berthelot et al. *ReMixMatch: Semi-Supervised Learning with Distribution Alignment and Augmentation Anchoring*. 2020. arXiv: 1911.09785 [cs.LG].

[4] Samuel R. Bowman et al. *Generating Sentences from a Continuous Space*. 2016. arXiv: 1511.06349 [cs.LG].

[5] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, eds. *Semi-Supervised Learning*. The MIT Press, 2006. ISBN: 9780262033589. URL: http://dblp.uni-trier.de/db/books/collections/CSZ2006.html.

[6] Marius Cordts et al. *The Cityscapes Dataset for Semantic Urban Scene Understanding*. 2016. arXiv: 1604.01685 [cs.CV].

[7] Bin Dai and David Wipf. *Diagnosing and Enhancing VAE Models*. 2019. arXiv: 1903.05789 [cs.LG].

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38. ISSN: 00359246. URL: http://www.jstor.org/stable/2984875 (visited on 05/07/2023).

[9] Charles Fox. *An introduction to the calculus of variations*. Dover Books on Mathematics. Mineola, NY: Dover Publications, Mar. 1987.

[10] Ankush Ganguly and Samuel W. F. Earp. *An Introduction to Variational Inference*. 2021. arXiv: 2108.13083 [cs.LG].

[11] Samuel Gershman and Noah Goodman. "Amortized inference in probabilistic reasoning". In: *Proceedings of the annual meeting of the cognitive science society*. Vol. 36. 36. 2014.

[12] Yves Grandvalet and Y. Bengio. "Semi-supervised Learning by Entropy Minimization". In: vol. 17. Jan. 2004.

[13] Louis N Hand and Janet D Finch. "LAGRANGIAN MECHANICS". In: *Analytical Mechanics*. Cambridge: Cambridge University Press, Nov. 1998, pp. 1–43.

[14] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647. eprint: https://www.science.org/doi/pdf/10.1126/science.1127647. URL: https://www.science.org/doi/abs/10.1126/science.1127647.

[15] Michael Jordan. *The Exponential Family: Basics*. 2009. URL: https://people.eecs.berkeley.edu/~jordan/courses/260-spring10/other-readings/readings.html.

[16] Michael I Jordan et al. "An introduction to variational methods for graphical models". In: *Machine learning* 37 (1999), pp. 183–233.

[17] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114 [stat.ML].

[18] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392. DOI: 10.1561/2200000056. URL: https://doi.org/10.1561%2F2200000056.

[19] Diederik P. Kingma et al. *Semi-Supervised Learning with Deep Generative Models*. 2014. arXiv: 1406.5298 [cs.LG].

[20] Durk P Kingma et al. "Improved Variational Inference with Inverse Autoregressive Flow". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/ddeebdeefdb7e7e7a697e1c3e3d8ef54-Paper.pdf.

[21] Donald E Kirk. *Optimal Control Theory*. Dover Books on Electrical Engineering. Mineola, NY: Dover Publications, Apr. 2004.

[22] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: 2009.

[23] Samuli Laine and Timo Aila. *Temporal Ensembling for Semi-Supervised Learning*. 2017. arXiv: 1610.02242 [cs.NE].

[24] Dong-Hyun Lee. "Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks". In: *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)* (July 2013).

[25] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[26] Calvin Luo. "Understanding Diffusion Models: A Unified Perspective". In: *calvinyluo.com* (2022). URL: https://calvinyluo.com/2022/08/26/diffusion-tutorial.html.

[27] Lars Maaløe et al. "Auxiliary Deep Generative Models". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1445–1453. URL: https://proceedings.mlr.press/v48/maaloe16.html.

[28] Alireza Makhzani and Brendan Frey. *k-Sparse Autoencoders*. 2014. arXiv: 1312.5663 [cs.LG].

[29] Lei Mao. "Introduction to Variational Inference". In: *leimao.github.io* (2019). URL: https://leimao.github.io/article/Introduction-to-Variational-Inference/.

[30] Radford M Neal and Geoffrey E Hinton. "A view of the EM algorithm that justifies incremental, sparse, and other variants". In: *Learning in graphical models* (1998), pp. 355–368.

[31] Andrew Ng. *CS294A Deep Learning and Unsupervised Feature Learning lecture notes*. Jan. 2011.

[32] Avital Oliver et al. "Realistic Evaluation of Deep Semi-Supervised Learning Algorithms". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/c1fea270c48e8079d8ddf7d06d26ab52-Paper.pdf.

[33] Yassine Ouali, Céline Hudelot, and Myriam Tami. "An Overview of Deep Semi-Supervised Learning". In: *CoRR* abs/2006.05278 (2020). arXiv: 2006.05278. URL: https://arxiv.org/abs/2006.05278.

[34] Danilo Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1530–1538. URL: https://proceedings.mlr.press/v37/rezende15.html.

[35] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, 22–24 Jun 2014, pp. 1278–1286. URL: https://proceedings.mlr.press/v32/rezende14.html.

[36] Salah Rifai et al. "Contractive Auto-Encoders: Explicit Invariance During Feature Extraction." In: *ICML*. Ed. by Lise Getoor and Tobias Scheffer. Omnipress, 2011, pp. 833–840. URL: http://dblp.uni-trier.de/db/conf/icml/icml2011.html#RifaiVMGB11.

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[38] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[39] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. *Regularization With Stochastic Transformations and Perturbations for Deep Semi-Supervised Learning*. 2016. arXiv: 1606.04586 [cs.CV].

[40] Dmitrij Schlesinger, Oleksandr Shekhovtsov, and Flach Boris. "Symmetric Equilibrium Learning of VAE". In: *NeurIPS* (2023).

[41] Alexander Shekhovtsov, Dmitrij Schlesinger, and Boris Flach. "VAE Approximation Error: ELBO and Exponential Families". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=0Is3SxU5Ynl.

[42] Kihyuk Sohn et al. *FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence*. 2020. arXiv: 2001.07685 [cs.LG].

[43] Casper Kaae Sønderby et al. *Ladder Variational Autoencoders*. 2016. arXiv: 1602.02282 [stat.ML].

[44] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958.

[45] Antti Tarvainen and Harri Valpola. *Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results*. 2018. arXiv: 1703.01780 [cs.NE].

[46] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: Jan. 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294.

[47] Lilian Weng. "From Autoencoder to Beta-VAE". In: *lilianweng.github.io* (2018). URL: https://lilianweng.github.io/posts/2018-08-12-vae/.

[48] Wikipedia. *Allegory of the cave — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Allegory%20of%20the%20cave&oldid=1151553170. [Online; accessed 06-May-2023]. 2023.

[49] Wikipedia. *Graphical model — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Graphical%20model&oldid=1131124218. [Online; accessed 07-May-2023]. 2023.

[50] Wikipedia. *Markov chain Monte Carlo — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Markov%20chain%20Monte%20Carlo&oldid=1148174830. [Online; accessed 07-May-2023]. 2023.

[51] Sergey Zagoruyko and Nikos Komodakis. *Wide Residual Networks*. 2017. arXiv: 1605.07146 [cs.CV].

[52] Hongyi Zhang et al. *mixup: Beyond Empirical Risk Minimization*. 2018. arXiv: 1710.09412 [cs.LG].

[53] Xiaojin Zhu. *Semi-Supervised Learning Literature Survey*. Tech. rep. 1530. Computer Sciences, University of Wisconsin-Madison, 2005. URL: http://pages.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf.