Wrocław University
of Science and Technology

Faculty of Information and Communication Technology
Field of study: **ARTIFICIAL INTELLIGENCE**

Master's Thesis

REPRESENTATION LEARNING WITH BAYESIAN METHODS

Michał Karbownik

keywords:
Bayesian representation learning, Bayesian, representation, deep learning, uncertainty

short summary:
This work tries to bridge a gap between recent representation learning advancements and their lack of proper Bayesian perspective by introducing, analyzing, and testing a cross-section of Bayesian representation learning models by an appropriate theoretical introduction, along with an observational and quantitative study.

Supervisor	dr hab. inż.Tomasz Kajdanowicz
	Title/degree/name and surname

The final evaluation of the thesis

Chairman of the Diploma Examination Committee	.....	.....	.....
	Title/degree/name and surname	grade	signature

For the purposes of archival thesis qualified to:*

a) category A (perpetual files)

b) category BE 50 (subject to expertise after 50 years)

* Delete as appropriate

stamp of the faculty

Wrocław 2022

Abstract

Learning representations of the data that make it easier to extract useful information when building classifiers or other predictors is referred to as representation learning. Recent years have brought a tremendous advancement in this area of Artificial Intelligence. It seems, however, as if this development neglected the benefits Bayesian approach could provide, namely uncertainty quantification, interpretability, or better latent space properties resulting in more universal and quality representations. This work tries to bridge this gap by introducing, analyzing, and testing a cross-section of Bayesian representation learning models by an appropriate theoretical introduction, along with an observational and quantitative study.

Streszczenie

Uczenie się reprezentacji danych, które ułatwiają wydobywanie przydatnych informacji podczas budowania klasyfikatorów lub innych predyktorów, jest nazywane uczeniem reprezentacji. Ostatnie lata przyniosły ogromny postęp w tej dziedzinie sztucznej inteligencji. Wydaje się jednak, że rozwój ten pomija korzyści, jakie może zapewnić podejście bayesowskie, a mianowicie kwantyfikację niepewności, interpretowalność lub lepsze właściwości przestrzeni ukrytej, co skutkuje bardziej uniwersalnymi i jakościowymi reprezentacjami. Niniejsza praca próbuje wypełnić tę lukę, przedstawiając, analizując i testując przekrój modeli uczenia reprezentacji bayesowskich poprzez odpowiednie wprowadzenie teoretyczne, wraz z badaniami obserwacyjnymi oraz ilościowymi.

Contents

1 Introduction **2**
- 1.1 Motivation . . . 2
- 1.2 Aim and scope . . . 2
- 1.3 Research problems . . . 3
- 1.4 Contribution . . . 3
- 1.5 Work structure . . . 3

2 Background and literature review **4**
- 2.1 Representation learning . . . 4
  - 2.1.1 What is a representation? . . . 4
  - 2.1.2 What makes a good representation? . . . 6
  - 2.1.3 Representation learning taxonomy . . . 8
- 2.2 Bayesian approach . . . 9
  - 2.2.1 Bayesian vs frequentist . . . 10
  - 2.2.2 Bayesian inference in representation learning . . . 11
- 2.3 Bayesian representation learning . . . 12

3 Methods introduction **14**
- 3.1 PCA and Bayesian PCA . . . 14
- 3.2 NN and Bayesian NN . . . 17
- 3.3 AE and Variational AE . . . 20

4 Experimental setting **23**
- 4.1 Datasets . . . 23
- 4.2 Observational study . . . 24
- 4.3 Quantitative study . . . 25

5 Observational study **26**
- 5.1 Bayesian PCA . . . 26
  - 5.1.1 Number of principal components . . . 26
  - 5.1.2 Inference . . . 27
  - 5.1.3 Representation reconstruction . . . 27
  - 5.1.4 Further observations . . . 28
- 5.2 Bayesian Neural Network . . . 32
  - 5.2.1 Learning process . . . 32
  - 5.2.2 Uncertainty . . . 32

5.2.3 Prior distributions . . . 32
5.3 Variational Autoencoder . . . 40
5.3.1 Latent separability . . . 40
5.3.2 Latent space quality . . . 40
5.3.3 Latent size analysis . . . 42

6 Quantitative study results **45**
6.1 Raw data . . . 45
6.2 PCA vs BPCA . . . 46
6.3 NN vs BNN . . . 47
6.4 AE vs VAE . . . 51
6.5 Conclusions . . . 51

7 Conclusions and further work **52**

Bibliography **58**

List of Figures

2.1 Numeric systems as representations 5

3.1 A DNN neuron 17
3.2 Comparison of DNN and BNN approaches 18
3.3 Comparison of AE and VAE high-level architectures 21

5.1 Hinton diagram of the BPCA projection matrix 29
5.2 Spread of BPCA α hyperparameters 30
5.3 ELBO course during BPCA inference 30
5.4 Reconstruction of data with PCA and BPCA 31
5.5 Loss and accuracy course during BNN learning 34
5.6 Example of uncertainty handling in BNN 35
5.7 Sigma values influence on the BNN learning process 36
5.8 Mixing coefficient values influence on the BNN learning process 37
5.9 Comparison of AE and VAE latent separation 41
5.10 Comparison of AE and VAE latent visualizations 41
5.11 Comparison of AE and VAE average input reconstructions 42
5.12 Latent size influence on VAE representation separability 43
5.13 Latent size influence on VAE average input reconstruction 44

List of Tables

6.1 Representations evaluation for raw datasets 45
6.2 Representations evaluation for BPCA and PCA 46
6.3 Dimensionality chosen by BPCA for each dataset 46
6.4 Representations evaluation for BMLP and MLP 47
6.5 Representations evaluation for BCNN and CNN 48
6.6 Representations evaluation for sampling counts of BMLP 49
6.7 Representations evaluation for sampling counts of 49
6.8 NN models F1-score on a test set 50
6.9 Representations evaluation for VAE and AE 51

1 Introduction

The idea of representations, while coming from the neuroscience background, has found its adoption in the field of Artificial Intelligence in a form of *representation learning*. The learned representations should be transferable to other tasks and demonstrate certain qualities, such as limited dimensionality, embedding hidden features required for knowledge transfer, being interpretable or benefiting other predictors in their downstream tasks.

Especially recent years have brought advancement in this area. It seems, however, as if this development neglected the benefits Bayesian approach could provide. There is little to no work considering this perspective. This work introduces the problem of representation learning from a Bayesian angle.

1.1 Motivation

As already mentioned, literature on the topic of Bayesian representation learning is scarce. The existing work is usually task-specific (usually with publicly unavailable data), which prevents reproducibility and confines application opportunities. Additionally, available methods are often very novel and complex, keeping the entry threshold for utilizing them high. Moreover, no previous work has been found that explicitly defined this problem, gathered the methods and introduced the topic of Bayesian representation learning.

1.2 Aim and scope

The main objective of this work is to introduce, analyze and test a cross-section of Bayesian representation learning methods. In order to meet the objective, the following scope was defined:

- Provide a literature review and theoretical background of feature learning and Bayesian approach to it.
- Provide theoretical background for 3 Bayesian representation learning methods: Bayesian PCA, Bayesian Neural Network and Variational Autoencoder.
- Analyze those methods in an observational study, highlighting their characteristics.
- Perform downstream task testing on representations learned with those methods.

1.3 Research problems

In this work, the following problems were identified and investigated:

- How do PCA, neural networks, autoencoders, and their Bayesian counterparts function, and how do they obtain representations?
- What are possible advantages and drawbacks of using Bayesian PCA, Bayesian Neural Networks and Variational Autoencoders for representation learning?
- What characteristics do those methods demonstrate?
- How well do representations learned by them perform on downstream tasks with other classifiers?

1.4 Contribution

While this work by no means is an exhaustive survey of existing methods, it aims to introduce, apply and test a cross-section of Bayesian representation learning methods. A by-product of this is an introduction to Bayesian representation learning, which, with adequate adjustments, could be used in self-study or didactics. By testing fundamental methods of this paradigm, it provides a background and lays the foundation for the future work.

1.5 Work structure

This work consists of 7 chapters. Below, a short summary of each of them is provided.

1. chapter 1 is an introductory chapter. It explains the motivation and contribution of this work, along with enumerating its aim and stated research problems.
2. chapter 2 provides necessary theoretical introduction and background on representation learning and Bayesian approach to it.
3. chapter 3 explains the theory and mechanics of Bayesian PCA, Bayesian Neural Networks and Variational Autoencoders, with a mention of their conventional counterparts.
4. chapter 4 describes used datasets and experimental setup.
5. chapter 5 conducts an observational study on the three Bayesian representation learning methods.
6. chapter 6 performs a series of downstream task quantitative experiments on trained representations.
7. chapter 7 draws conclusions and provides ideas for possible future work.

2 Background and literature review

2.1 Representation learning

Representation learning, often referred to as *feature learning*, is a set of machine learning approaches that allows for automatic feature detection from data. This is in contrast to manual feature extraction which, while takes advantage of the domain knowledge about data and human ingenuity, is both time-consuming, labor-intensive and cannot take advantage of the latent space. A comprehensible definition of representation learning was formulated by Bengio et al. [1] in 2012:

Definition 2.1 (Representation learning) *Learning representations of the data that make it easier to extract useful information when building classifiers or other predictors.*

Definition 2.1 does not, however, delve into the core issue of what the representation is – even though the definition seems to be intuitive, it is hardly to be found in the domain-specific literature.

2.1.1 What is a representation?

As it often comes with Artificial Intelligence, being highly inspired by processes taking place in nature (in particular by the field of cognition, which is the main interest of cognitive science), it is not the sole domain whose interests revolve around representations. Not only the term itself, but also the very ontology of representations, play a central role in neuroscience literature. However, even there neither a clear consensus on all aspects, nor a unified definition of it exist [2]. Having said that, probably the most established is Definition 2.2.

Definition 2.2 (Representation (in neuroscience)) *A formal system for making explicit certain entities or types of information, together with a specification of how the system does this. [3]*

An example of such would be a numeral system, which is a writing system for expressing (representing) numbers — the Roman, Hindu–Arabic (decimal) and binary being among the most widespread. Figure 2.1 shows different representations of the same number in these systems.

Each of those representations was formed with a specific goal in mind – to simplify certain operations. The Roman system, used by the ancient Romans to count and perform other day-to-day transactions. It is non-positional [4] and governed by principles of subtraction and addition. Despite being quite primitive, it had quickly become overcomplicated due to its

XI 11 1011

Figure 2.1: Numeric systems as representations.

poor scalability [5]. It constituted a barrier to the mathematical development of the Roman Empire that had been significant to the earlier Arabic cultures. They had used the decimal numeric system, which, being multiplicative by nature (easily decomposed into powers of 10) and positional, facilitates more complex arithmetics. The binary system, using only two symbols - 0 and 1 - through advancements of the Boolean algebra applied to the electrical switches, is widely used in modern computers. In general, each of them provides a different kind of abstraction to aid a specific objective.

Despite the information content being the same in all three cases, their representation is different, which implies the orthogonality of the two. Switching between representations should not influence the original value they describe, and the transition process should be unambiguous due to explicitness (see Definition 2.2). Even though systems might share the same symbols, values expressed by them might differ (i.e., a 1 on *n-th* position will produce a different value in decimal and binary system). Because of that, a representation should be defined by the shape of the manifold on which the data lie within the representational space, as opposed to a single piece of information [6].

According to the above, almost anything can be subject to being represented, from everyday objects to complex abstract concepts. Similarly, representation learning touches upon different aspects and applications of machine learning (for particular examples, see section 2.3). A useful data representation should yield an improvement of the performance of a model, as well as facilitate the learning process – the scale of both highly depends on appropriately selected features (it is especially true for the deep learning models).

With this perspective, one could form a loose definition of *representation* in machine learning:

Definition 2.3 (Representation) *A function mapping input data to a latent vector, which is useful for a task.*

Subsequently, *representation learning* can be comprehended as an optimization problem of parameters of such function. If x denotes the data, θ the parameters and $\vec{z}$ - the latent vector, the whole can expressed as Equation 2.1.

$$f(x;\theta) \rightarrow \vec{z} \tag{2.1}$$

We expect the dimension of the representation to be not higher than the original data ($dim(\vec{z}) \leq dim(x)$), but ideally it should be much lower ($dim(\vec{z}) \ll dim(x)$).

2.1.2 What makes a good representation?

According to Definition 2.3, the output set of features (ergo the latent vector) should be *useful* for a specific *task*. The task objective serves as a falsifiability mechanism defining their usefulness. It allows for eliminating uninformative features, and thus, reducing the dimensionality. The examples in subsection 2.1.1 emphasize the importance of a *quality* representation. Hence, it is valid to ask: "*what* makes a good representation?". Surprisingly, it is an ill-defined and challenging question.

Dimensionality reduction

Yet again, cognitive science lays foundations for artificial intelligence and aids in answering fundamental questions. In 1959, Horace Barlow, identified the significance of *redundancy reduction*[1] in information processing by human brains [8]. The original idea is to "achieve economy without losing any information at all". It can be inexactly (representation learning usually assumes some information loss) translated to a dimensionality constraint defined along Equation 2.1. Due to a threat of the *curse of dimensionality*, it is desirable that the representation has minimal possible number of dimensions, while maintaining the variance level (at least for the data that is informative for the task at hand). Barlow's findings later became a base for research on disentangled representations.

Disentanglement

The human perception of the world around heavily depends on visual recognition. Representations on the early stage of visual processing are not beneficial for the object recognition, because they are - "like two sheets of paper crumpled together" - *tangled* [9]. Only at later stages it is being untangled [10], therefore transformed from "visual representations that are easy to build, but are not easily decoded, into representations that we do not yet know how to build, but are easily decoded".

Correspondingly, building a solid representation for machine learning algorithms expects separation of information about the attributes (some features vary with respect to only one attribute), simplifying setting decision boundaries for subsequent tasks. Additionally, one wants to discard the build up of an unnecessary invariance related to nuisance attributes, but without disposing of practical information about the data. It diminishes the representation's sensitivity to the data's directions of variance that are not informative to the task. This procedure, called *disentangling factors of variation* [11], is nontrivial, but yields a representation, which is more robust than the original, tangled structure.

[1] Barlow revisited the topic of redundancy reduction in 2001 [7]. He stated that "the original work was wrong in over-emphasizing the role of compressive coding and economy in neuron numbers, but right in drawing attention to the importance of redundancy". Both documents accentuate the impact of Claude Shannon's work, as well as the "entanglement" and the mutual influence of neuroscience and artificial intelligence.

Geometric priors

Despite real-world data often being incomplete (or even damaged), high-dimensional and chaotic, it often shows certain, usually hidden, regularities. A good representation should identify and capture them, ideally in a low-dimensional form. This limits the exponential space of possible functions. Such regularities can be exposed through unified geometric principles called *geometric priors* [12]. Two fundamental principles applied to physically structured data are scale separation and symmetry.

Symmetry of an object or a system is a transformation that leaves some property of the object or system unchanged or maintains its invariance. *Scale separation*, on the other hand, retains important features of a signal during the transfer onto a coarser domain.

Invariance to scale, translation, or rotation often proves to be the key to improving performance of machine learning models. It is ubiquitous (and easily visualizable) in image processing. Convolutional Neural Networks, for instance, do incorporate the idea of geometric priors. Translational symmetry is achieved through filters with shared weights, while pooling exploits scale separation. They are not, however, fully invariant to rotation, unless appropriately extended.

Smoothness

Smoothness of the representation assumes its invariance to minor, insignificant changes of the data. Formally, it is expressed that if some input x is approximately equal to some other input y ($x \approx y$), the functions of both should also be approximately equal ($f(x) \approx f(y)$). It is one of the general priors, not only for representation learning, but also for the majority of machine learning approaches.

Bengio et al. [1] argue that, while useful, this assumption alone is insufficient to overcome the *curse of dimensionality* and "advocate learning algorithms that are flexible and non-parametric but do not rely exclusively on the smoothness assumption".

Distribution

Distribution of a representation assumes spread of the representation's particular features across multiple parts of the same model — following the example of neural networks, the final feature value consisting of contributions from different neural paths. Moreover, it also induces feature abstraction and re-use, which facilitate utilization of simple representation to create more abstract ones in hierarchical models.

2.1.3 Representation learning taxonomy

shallow vs. deep

Deep learning models, through multi-layer processing, facilitate learning multiple levels of abstraction of data representations. As per the definition of godfathers of deep learning – Yann LeCun, Yoshua Bengio and Geoffrey Hinton – they are "representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level" [13]. In other words, the design of deep learning implicitly assumes learning representations. As opposed to this, the term *shallow representation learning* (also referred to as "conventional"[2] or "traditional") will include all other methods.

linear vs. nonlinear

Linear models are such that adopt an idea of a linear transformation, which can be expressed by the following equation:

$$y = X\beta + \epsilon \tag{2.2}$$

where y is a vector of observed values of a *dependent variable* (output of the model), X - a matrix consisting of vectors of *independent variables*, a parameter vector β and an unobserved random variable ϵ responsible for handling noise.

In representation learning, it translates to linear separability, ergo independence, of particular features — while a convenient assumption, it limits the models' performance on data with nonlinearities that are often encountered in practical applications. Nonlinear models, on the other hand, while more expensive computationally, are capable of capturing those dependencies to some extent. A flagship example of a nonlinear approach is deep learning, mentioned in subsubsection 2.1.3, however, the literature offers numerous extensions to linear models that allow for capturing nonlinear patterns.

global vs. local

Global feature learning methods, aim to retain global knowledge about data within the learned feature space. For instance, PCA uses orthogonal linear transformation to change the coordinate system in such a way that retains as much variance as possible. It results in a set of linearly uncorrelated variables (converted from a higher-dimensional set of conceivably correlated ones).

Such an approach, however, does not retain the data's local similarities, unlike local methods that address the issue by trying to discover the data's hidden manifold structure (thus they are sometimes referred to as *manifold learning*).

[2]Please note that in this work, the term *conventional* is often used as an opposition to *Bayesian*.

supervised vs. unsupervised

Supervised learning has emerged as one of the first and the most common machine learning paradigms. The reason for it is that our brains use it as one of the fundamental approaches to discovering the world, as well as developing and maintaining a variety of functions [14]. It is characterized by utilizing annotated, ideally very large datasets. The task is to teach models how to yield the desired output (labels). In representation learning, it is expected for such a model to generalize well, in order to be able to use the learned features in other downstream tasks or in different models.

In *unsupervised learning* (this term was also derived from cognitive science [15]), the dataset is provided without the annotated labels. Its goal is to discover hidden patterns of data and cluster it without imposed boundaries (for instance by labels, like in supervised learning). Unsupervised learning used in representation learning is supposed to reduce dimensionality, uncover exploratory factors, and train vectors without human intervention.

It is also important to mention two other paradigms that emerged in recent years, namely *self-supervised* and *semi-supervised* representation learning. The latter can handle and make use of data that is labeled only partially. It performs well when it has a large amount of them and only a small portion that is labeled [16]. In self-supervised learning, models are provided with unstructured input data, and their task is to generate the labels automatically. Those are further used in subsequent iterations as ground truths. This approach is characterized by robustness to label corruption (and adversarial examples) [17], as well as generalization ability and data efficiency [18].

generative vs. contrastive

Another taxonomy, introduced by Liu et al. [18], assumes the existence of *generative* and *contrastive* methods. The former's task is to produce a latent vector, from which they reconstruct the input, thus being able to generate data. The latter bases on an idea of measuring similarity between samples and requires negative sampling, which often demands heavy data augmentation.

2.2 Bayesian approach

The term *Bayesian* has been a buzzword in machine learning for quite some time. As with every trend, it has also been overused and its initial meaning distorted. While by no means is it a complete guide to Bayesian statistics, this section provides necessary background knowledge to understand Bayesian representation learning.

2.2.1 Bayesian vs frequentist

There is an ongoing debate in the scientific community around statistics. Sometimes it is so tempestuous that one could call it a dispute, or even an argument. The disagreement revolves around the superiority of one of two approaches, and can be summarized in one question: "Bayesian or frequentist?".

The latter, is an interpretation, in which the probability[3] is strictly related to the frequency of events. It assumes the existence of true, unknown parameters, whose values are fixed. In this case, the inference is made over hypothetical random samples of data. In the context of machine learning, it manifests itself as optimization (rather than marginalization) and usually results in a point estimate of parameters achieved by different methods, of which the most common is *Maximum Likelihood Estimation* (MLE).

Bayesian inference, on the other hand, treats the model parameters as random variables and makes probabilistic statements about their distribution. At the very foundations, it is driven by the *Bayes Theorem*, which is often expressed by Equation 2.3:

$$P(H \mid D) = \frac{P(H)P(D \mid H)}{P(D)}, \tag{2.3}$$

where H is called a *hypothesis*, and D is data.

This formula can be easily adopted to be used in a model, like in Equation 2.4a:

$$\overbrace{P(\theta \mid D, I)}^{\text{posterior}} = \frac{\overbrace{P(\theta \mid I)}^{\text{prior}}\overbrace{P(D \mid \theta, I)}^{\text{likelihood}}}{\underbrace{P(D \mid I)}_{\text{marginal likelihood}}} \tag{2.4a}$$

$$P(D \mid I) = \int P(D \mid \theta, I)P(\theta \mid I)d\theta \tag{2.4b}$$

In this case, hypothesis (H) is replaced by parameters (θ), and additional knowledge about the model (e.g., its type or hyperparameters) - I is used. The *prior distribution* specifies an assumption about the parameters without taking the data into account. The *likelihood* represents the probability of the data under the specified parameters. The *marginal likelihood* (also called *evidence*) is the distribution of the data given our additional assumption. It can also be denoted without the parameters θ being marginalized out (Equation 2.4b). Marginal distribution is the normalization of the Bayes rule and plays an important role in model comparison, however, is often intractable. Finally, the *posterior distribution* is the distribution of the parameters after "seeing" the data (and the additional assumption I).

[3] In this work the term *probability distribution* is often shortened to both *probability* or *distribution*. While in statistics they represent different concepts (probability distribution is a mathematical function that describes the probability of different possible values of a variable), in artificial intelligence they are often used interchangeably.

The above is sometimes simplified to the form: "the posterior is proportional to the likelihood and the prior" and denoted as:

$$posterior \propto likelihood \times prior \tag{2.5}$$

2.2.2 Bayesian inference in representation learning

In Bayesian representation learning, latent vectors $\mathbf{z}$ are not maximum a posteriori point estimations, but rather realizations from a posterior distribution, obtained by approximating the intractable marginal distribution.

In general, there are two types of approximation techniques — stochastic and deterministic. The most common examples of them in Bayesian setting are, respectively, sampling, especially *Markov Chain Monte Carlo* (MCMC), and *Variational Inference* (VI).

Markov Chain Monte Carlo

The former, Markov Chain Monte Carlo, is a family of methods that, in theory, is able to provide exact sampling from the required distribution. In practice, however, it is not possible due to finite resources. The results achieved can be reliable when the sample is large enough. This is, in turn, computationally intensive; hence these methods are usually used for small-scale tasks with limited data. As the name suggests, it draws from two principles - *Markov Chain* and *Monte Carlo*.

The latter is a concept of performing a random walk by drawing *independent and identically distributed* (i.i.d) samples from a target distribution and estimating the desired quantity (such as variance or mean of the drawn samples). Introducing the "Markov Chain" part, however, weakens the i.i.d. constraint by requiring the dependency of the subsequent variables in a way that a given variable is dependent on its direct predecessor, which satisfies the Markov property:

$$p(x^i \mid x^{i-1}, \ldots, x^1) = p(x^i \mid x^{i-1}). \tag{2.6}$$

Furthermore, MCMC methods require a phase called *burn-in*, because they are sensitive to their starting point. During this phase, the initial samples are removed until the Markov chain converges to its stationary distribution, which adds to the computational intensity of the task.

Variational Inference

Variational inference methods are typically faster than MCMC because, instead of trying to approximate the target distribution, they redefine the problem to optimization with a simpler alternative (called *variational distribution*). The variational distribution must be flexible and complex enough to provide a reasonable approximation to the true posterior, while being simple enough to facilitate optimization. Because, in fact, there is a difference between the distributions, it needs to be quantified and considered in the inference task.

Kullback-Leibler divergence (KL or D_{KL}) is a score that measures how much information is lost when one distribution is approximated with another (and because of that, it is sometimes called *relative entropy*). It fulfills the basic criteria for being such a measure:

$$\begin{cases} D_{KL}(p \parallel q) \geq 0 & \text{(2.7a)} \\ D_{KL}(p \parallel q) = 0, \quad \text{if p(x) = q(x)} & \text{(2.7b)} \\ D_{KL}(p \parallel q) \neq KL(q \parallel p) & \text{(2.7c)} \end{cases}$$

Kullback-Leibler divergence in the considered case can be defined as:

$$D_{KL}\left[q(\theta) \parallel p(\theta \mid D)\right] = \int q(\theta) \log \frac{q(\theta)}{p(\theta \mid D)}\, d\theta. \tag{2.8}$$

The task is to minimize the KL divergence between the true and variational distributions, or more precisely, optimize the parameters of the latter. Because the unwanted term $p(\theta \mid D)$ is present in Equation 2.8, the Kullback-Leibler divergence is still intractable directly. A concept that can aid it, by transforming it into an optimization problem, is *Evidence Lower Bound* (ELBO). Compared to the log-evidence of the likelihood function:

$$log\, p(D;\theta) \geq \mathbb{E}_q \left[log \frac{p(D,\theta)}{q(\theta)} \right], \tag{2.9}$$

which, considering the aforementioned, can be transformed into:

$$D_{KL}\left[q(\theta) \parallel p(\theta \mid D)\right] = log\, p(D) - \mathbb{E}_q \left[log\, p(D,\theta) - log\, q(\theta)\right] \tag{2.10}$$

where the term on the right-hand side is ELBO, and the other is evidence. It is important to note that because the evidence does not depend on q, it can be assumed constant. Furthermore, as in Equation 2.7a, the KL is always non-negative. Hence, a tractable operation of maximizing the ELBO is equivalent to minimizing the KL divergence, which allows for the inference.

2.3 Bayesian representation learning

If one were to determine the precursor of representation learning, it would undoubtedly be Principal Component Analysis. The groundwork for it was laid in 1901 by Karl Pearson [19]. Later, in 1936, Harold Hotelling [20] named and developed it to its prevailing form. It was not until 1998 that its Bayesian counterpart was introduced by Christopher Bishop [21]. It is also this method that could be considered one of the first Bayesian representation learning methods.

Representation learning in general was not widely popular until ca. 2006, when both the advancement of technology facilitated the development of deep learning. Only then the progress on feature learning techniques, which are often computationally expensive, could also be advanced.

Even though theory for the Bayesian approaches has been available for a long time (a specific case of Bayes' theorem, which laid the foundation for the Bayesian statistics, was published already in 1763 [22]), it was not until a few decades ago that these methods were actually applied in machine learning [21, 23, 24] and they are being rediscovered only recently [25, 26, 27, 28, 29, 30, 31, 32].

For the Bayesian representation learning, the field is even more scarce. Among the aforementioned Bayesian methods, there exist those, that enable representation learning, however, they are usually not studied from that perspective. In fact, only a few works discuss them, and even if — they are either task-specific (thus hardly universal) or very complex novel approaches [33, 34, 35, 36].

3 Methods introduction

A quality theoretical introduction is a part of every quality dissertation. In order to use an apparatus, it is crucial to understand how and why it works. This chapter introduces the methods that will be used in the further study. It provides theoretical background, as well as necessary comments.

3.1 PCA and Bayesian PCA

As mentioned in section 2.3, *Principal Component Analysis* (PCA) is one of the first and simplest representation learning methods. Compared to other existing models, it is so trivial, that sometimes considered only in terms of a dimensionality reduction technique. It does, however, fulfill the criteria of being a feature learning algorithm. Except being global (see subsubsection 2.1.3), it is also a linear, unsupervised, generative, conventional approach.

The aforementioned characteristics, along with having a Bayesian counterpart (see subsubsection 3.1), cause the PCA to be an appropriate experimental subject of this study.

PCA

The idea behind Principal Component Analysis is relatively simple—it tries to project data onto a lower-dimensional space while preserving as much variance as possible. It is achieved by finding a best-fitting line (or a hyperplane, because the procedure usually applies to higher dimensions), which minimizes the average squared distance from the points to the line. Because PCA is closely related to paragraph 3.1 (PPCA is a generalization of PCA), its algorithm will be explained in the following paragraphs.

BPCA

Bayesian Principal Component Analysis (BPCA) [21] is an extension to the idea of conventional PCA. BPCA is based on *Probabilistic PCA* [37][38], which is in turn a special case of *Factor Analysis* [39]. All of them, are very much alike with PCA in terms of their main objective — to achieve dimensionality reduction by discarding uninformative features. Unlike conventional PCA, however, all of them combined possess one distinct feature — they introduce into the Bayesian approach step by step. They were introduced in the same manner in the following paragraphs.

Factor Analysis

Factor Analysis (FA) is one of the most fundamental generative models. Although it's considered one of the simplest of its kind, the complete mathematical theory behind is rather complex and has been extensively reviewed in the literature [39][40][41]. Thus, it was kept out of the scope of this work. It is, however, important to comprehend basic concepts behind Factor Analysis in order to grasp the mode of action of BPCA.

As the name suggests, factor analysis assumes the existence of *factors* - in this case, a set of unobserved variables. Each of those latent variables ($x_i \in \mathbb{R}^q$) is considered to be a generator of one, or ideally more, observations ($t_n \in \mathbb{R}^d$). Each data point can be obtained by performing the following procedure:

1. Using a projection matrix $\mathbf{W} \in \mathbb{R}^{qxd}$, project linearly the latent q-dimensional variable x_i onto a d-dimensional space[1].
2. Apply a linear translation ($\boldsymbol{\mu} \in \mathbb{R}^d$).
3. Add Gaussian noise ($\boldsymbol{\epsilon} \in \mathbb{R}^d$) with a covariance matrix ($\boldsymbol{\Psi} \in \mathbb{R}^{dxd}$).

What is important is that the covariance matrix is diagonal, because of the independence of the noise within particular dimensions.

This produces a linear relationship between the observable and latent, which is expressed as Equation 3.1a. The unobserved variable is assumed to be from the zero-mean unit variance Gaussian (see Equation 3.1b). The noise is likewise a Gaussian distribution, but parametrized by a covariance matrix $\boldsymbol{\Psi}$ instead of identity (see Equation 3.1c).

$$\mathbf{t} = \mathbf{W}\mathbf{x} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \tag{3.1a}$$

$$\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{3.1b}$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Psi}) \tag{3.1c}$$

A natural consequence of a latent-variable formulation of Factor Analysis is that it can be solved iteratively through the expectation-maximization (EM) procedure [42].

Probabilistic PCA

A shift from the factor analysis to the *Probabilistic Principal Component Analysis* (also *Probabilistic PCA*, or simply *PPCA*) is quite straightforward - PPCA is Factor Analysis, but with an isotropic noise ($\boldsymbol{\Psi} = \sigma^2\mathbf{I}$ in Equation 3.1c), in other words, equal in all dimensions of data space. Because of that, it is assumed that data points are independent of each other's latents.

The aforementioned change implies a multivariate Gaussian distribution of observable variables ($\mathbf{t}$) conditioned on latent variables ($\mathbf{x}$), as depicted in Equation 3.2.

[1]The matrix is sometimes also referred to as a *loading matrix*, and thus denoted by $\mathbf{L}$.

$$p(\mathbf{t} \mid \mathbf{x}) = \mathcal{N}(\mathbf{W}\mathbf{x} + \boldsymbol{\mu}, \sigma^2\mathbf{I}) \tag{3.2}$$

In this setting, Probabilistic Principal Component Analysis is closer to conventional Principal Component Analysis than Factor Analysis. As the noise's variance approaches 0, the probabilistic factor essentially disappears, and PPCA is reduced to PCA (see Equation 3.3) [43].

$$\lim_{\sigma^2 \to 0} \mathbb{P}(\boldsymbol{\epsilon}) = \lim_{\sigma^2 \to 0} \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}) = \mathcal{N}(\mathbf{0}, \lim_{\sigma^2 \to 0}(\sigma^2)\mathbf{I}) \tag{3.3}$$

Probabilistic PCA, similarly to factor analysis, can be solved by the Expectation–Maximization algorithm, however, it also has a closed-form solution of maximum likelihood estimation extracts the principal components of the observations [37].

Bayesian PCA

One of the biggest drawbacks of both the conventional PCA, and PPCA is a need for specifying the number of principal components to be retained. One of the hyperparameters of the Probabilistic PCA is the latent space dimensionality. An exhaustive search of its space would in many cases be computationally intractable. *Bayesian Principal Component Analysis* (known as *Bayesian PCA*, or *BPCA*) goes one step further by treating this dimensionality as another latent variable, and thus allowing its automatic selection.

As expressed by Christopher M. Bishop [21]: "Armed with the probabilistic reformulation of PCA [...], a Bayesian treatment of PCA is obtained by first introducing a prior distribution

$$p(\mu, \mathbf{W}, \sigma^2)$$

over the parameters of the model.". The corresponding posterior distribution

$$p(\mu, \mathbf{W}, \sigma^2 \mid D),$$

where D is a set of observed d-dimensional vectors ($D = \{\mathbf{t}_n\}; n \in \{1, \ldots, N\}$), can be attained by normalizing a product of the likelihood function and the prior. Finally, marginalizing over parameters yields the predictive posterior.

Avoiding discrete model search in order to control the latent space effective dimensionality (parallel to the number of retained principal components) requires introducing a *hierarchical prior*

$$p(\mathbf{W} \mid \boldsymbol{\alpha})$$

over matrix $\mathbf{W}$, governed by a vector of hyperparameters $\boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_q\}$ (the dimensionality is restricted to be $q \leq d - 1$).

Further derivations are lengthy and are not crucial for understanding the method itself, nor the representation learning process, thus are not covered by this work (they can be found in the original paper).

Conclusions

An important takeaway point is, that Bayesian PCA can obtain a representation in a Bayesian fashion. The advantages of such include, but are not limited to, uncertainty handling of the model parameters, which can result in a better quality representation and more understanding of its creation process. The Bayesian treatment provides direct benefits in the form of an automatic selection of a number of output features.

3.2 NN and Bayesian NN

Neural Networks

Neural Networks are deep, supervised, discriminative models, which consist of at least two connected layers of nodes—input and output, and usually with one or more hidden layers in between (see Figure 3.2a). In Figure 3.1 a visualization of a neuron is presented. Each neuron has weights w incoming from inputs x of the previous layer (with an addition of a bias term b), and an activation function f, which yields an output z.

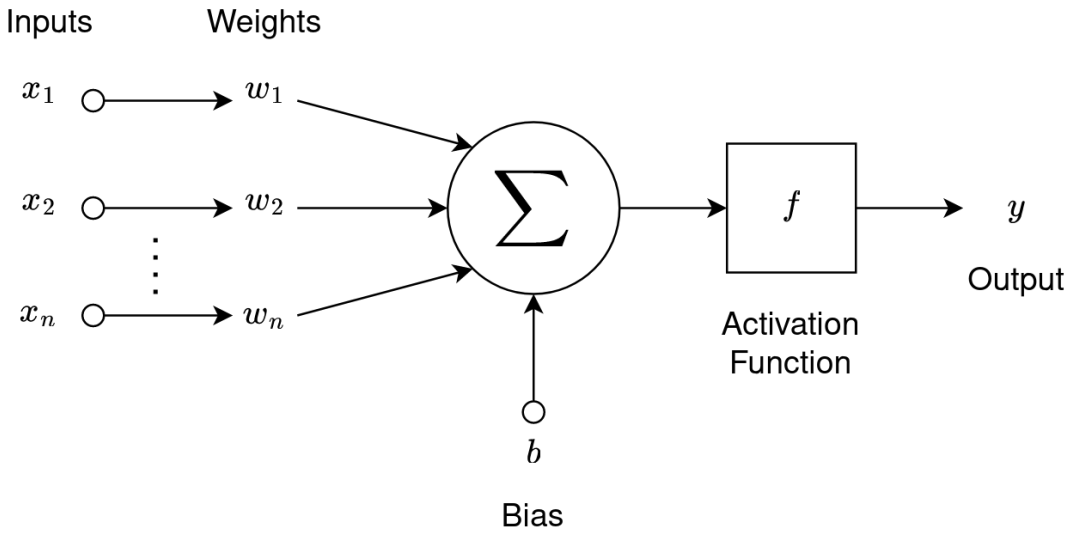
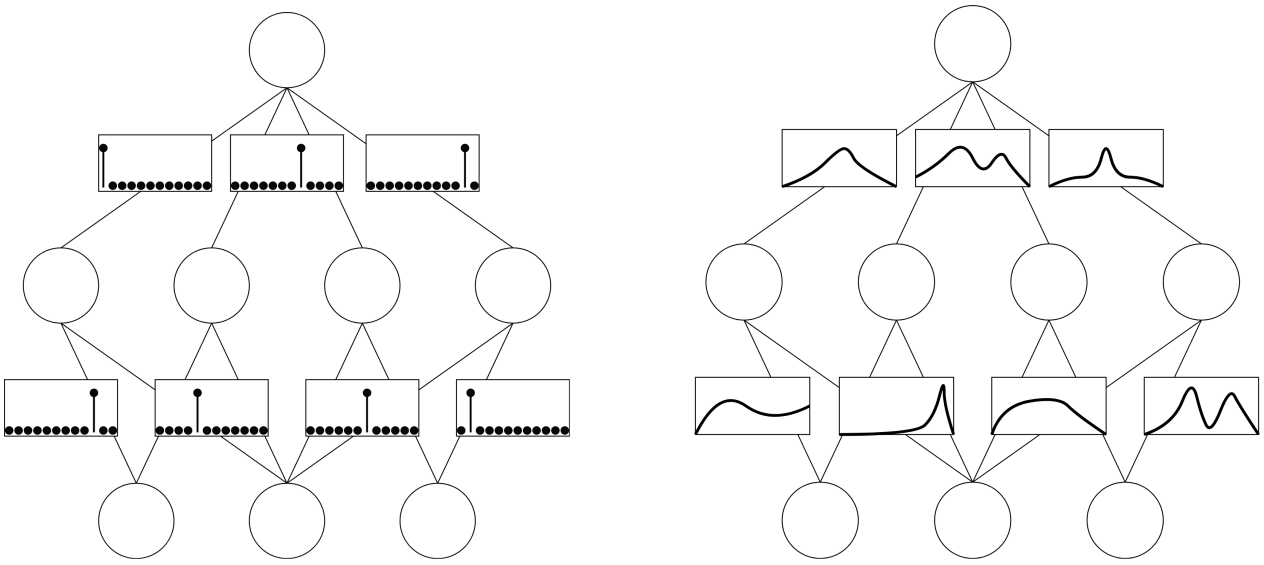


Figure 3.1: A visualization of a Deep Neural Network's neuron.

This way, weights of the current node are a linear transformation of the previous one. It can be denoted as follows:

$$z^{(i+1)} = W^{(i+1)}x^{(i)} + b^{(i+1)} \tag{3.4}$$

In the deterministic setting, weights directly correspond to trainable parameters. They are optimized with a backpropagation algorithm that uses gradient descent to optimize the cost function or the error of the model. This results in point estimates of the weights, which is visualized in Figure 3.2a.



(a) DNN with point estimates (b) BNN with distributions over weights

Figure 3.2: A comparison of Deep Neural Network point-estimate (a) and Bayesian Neural Network (b) distributions-over-weights approaches.

Conventional neural networks, do come with certain disadvantages. They tend to be miscalibrated, and thus overconfident in their predictions [44]. "A neural network can represent many models consistent with our observations. By selecting only one, in a classical procedure, we lose uncertainty when the models disagree for a test point" [27].

Bayesian Neural Networks

Bayesian Neural Networks (BNN) is a family of methods that extend conventional neural networks with Bayesian inference of posterior distributions on parts of their architecture, most often either weights (see Figure 3.2b) or activation functions. They are sometimes described as neural networks with stochastic elements trained in a Bayesian framework [23]. Introducing the Bayesian factor results in certain advantages with the conventional approach.

The most important is a realistic expression of uncertainty and calibration. Gathering the uncertainty helps to reduce the network's variance over the prediction of a specific data point. It provides an edge of being able to assess the model quality not only in terms of how many times it was wrong, but also by how much. Additionally, because of the Bayesian setting, it is possible to encode a priori knowledge about the network parameters. In certain situations, it results in less complex learning of more accurate models.

Considering how convoluted the concepts of neural networks and Bayesian approaches are, it is surprisingly how simple it is to combine both of them. There is a wide range of possible approaches to designing a functioning model. Because of the multiplicity of possibilities, the following paragraphs will focus only on the approach adopted in this work.

Bayes by Backprop

Bayes by Backprop (BBB) [26] is a method from the variational inference family that is adapted for neural networks. In its essence, it is supposed to replace conventional backpropagation (thus its name) with learning the parameters of an approximate posterior by unbiased Monte Carlo estimates of the gradients.

In this setting, a stochastic neural network can be seen as an ensemble model, in which each network can be drawn from a common and learned probability distribution by sampling the weights. The exact Bayesian inference on the weights of a neural network is intractable (due to a large space of parameters). Because of that, the goal is to find parameters θ that minimize the Kullback-Leibler divergence between the variational distribution $q(w \mid \theta)$ and the posterior $P(w \mid D)$:

$$\begin{aligned} \theta_{opt} &= \arg\min_\theta D_{KL}\left[q(w \mid \theta) \parallel P(w \mid D)\right] \\ &= \arg\min_\theta \int q(w \mid \theta) \log \frac{q(w \mid \theta)}{P(w)\, P(D \mid w)}\, dw \\ &= \arg\min_\theta D_{KL}\left[q(w \mid \theta) \parallel P(w)\right] - \mathbb{E}_{q(w|\theta)}\left[log\, P(D \mid w)\right], \end{aligned} \tag{3.5}$$

where w are the weights. In order to obtain the unbiased estimates of the gradients, a Monte Carlo simulation can be performed. Then, the optimal weights are as follows:

$$\theta_{opt} = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{n} log\, q(\mathbf{w}^{(i)} \mid \theta) - log\, P(\mathbf{w}^{(i)}) - log\, P(D \mid \mathbf{w}^{(i)}), \tag{3.6}$$

where $\mathbf{w}^{(i)}$ denotes the i-th Monte Carlo sample drawn from the variational posterior $q(\mathbf{w}^{(i)} \mid \theta)$, and N is the number of Monte Carlo estimations. Note that the last term - $log\, P(D \mid \mathbf{w}^{(i)})$ is a likelihood function of the conventional neural network.

Reparameterization trick

As a consequence of the sampling procedure being indifferentiable, the parameter optimization process requires reparameterization [28]. To obtain the weight w, if assuming a Gaussian, every learnable parameter will be defined by the parameters of this distribution — mean μ and variance σ^2, then:

$$\theta = \mu + \sigma + \epsilon, \tag{3.7}$$

where ϵ is noise and $\epsilon \sim \mathcal{N}(0, 1)$. It is also the only intractable element. Because, however, only μ and σ are important, it can be skipped. In such manner, gradients can be induced:

$$\Delta_\mu = \frac{\partial f}{\partial \theta} + \frac{\partial f}{\partial \mu} \tag{3.8}$$

$$\Delta_\sigma = \frac{\partial f}{\partial \theta} \frac{\epsilon}{\sigma} + \frac{\partial f}{\partial \sigma}. \tag{3.9}$$

Parameters are updated the following way:

$$\mu^{(t+1)} = \mu^t - \alpha\Delta_\mu \tag{3.10}$$
$$\sigma^{(t+1)} = \sigma^t - \alpha\Delta_\sigma \tag{3.11}$$

Representations

Because neither DNN nor BNN defines representation explicitly, it might not seem obvious how to obtain it. This is, in fact, reasonably simple — the outputs of each layer of the network are a separate representation, and it is enough to extract it from there. While designing a network with representation learning in mind, it is important to consider the right architecture (for instance, dimensionality of the desired layer).

3.3 AE and Variational AE

Autoencoders

Autoencoders, introduced by Rumelhart et al. in *Learning internal representations by error propagation* [45], are a type of unsupervised deep neural network. They consist of two parts — an *encoder* and a *decoder*. The former transforms the input into a latent space, which is an informative representation. The decoder's goal is to reconstruct the inputs from the latent space [46]. In other words, because of the imposed task, an autoencoder is a model mapping an input X to an output X' using its internal, latent representation Z, which is visualized in Figure 3.3a. It is important, because the representation can be extracted and used in other models.

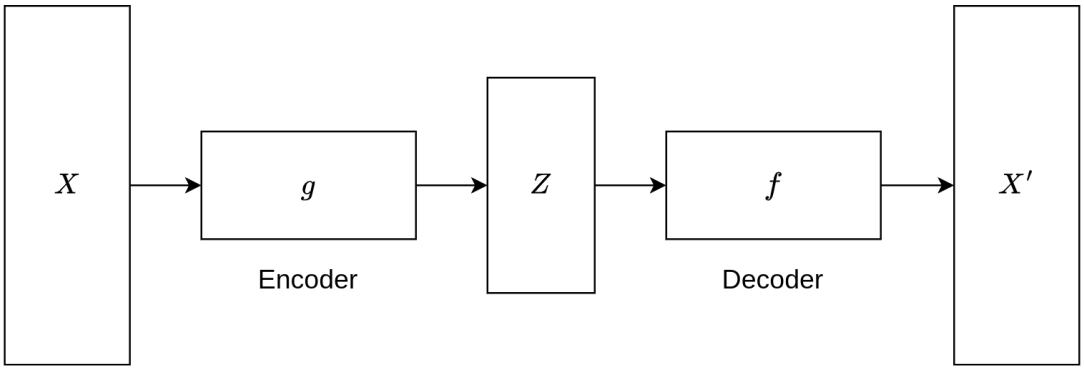
In general, autoencoders seek to satisfy the following equation:

$$argmax_{g,f}\, \mathbb{E}\left[\Delta(x, f \circ g(x))\right], \tag{3.12}$$

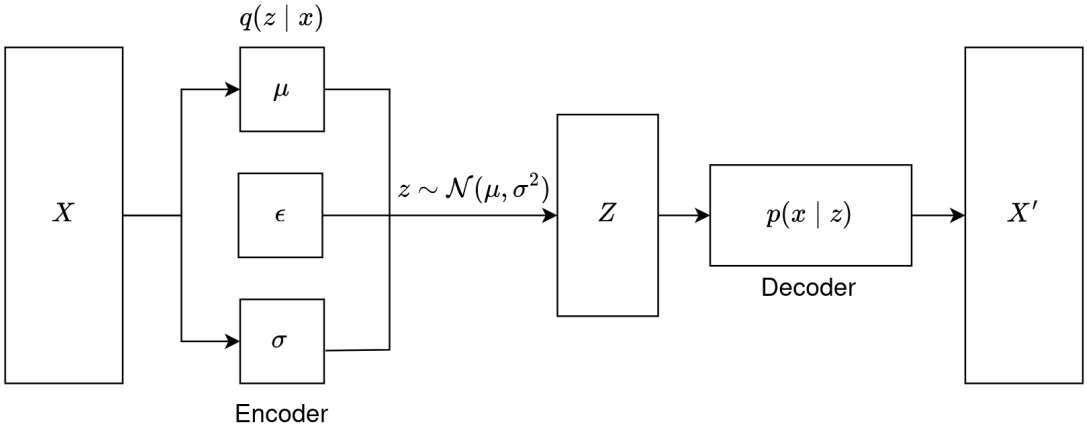
where $g : \mathbb{R}^d \to \mathbb{R}^q$ is a function that the encoder is trying to learn and $f : \mathbb{R}^q \to \mathbb{R}^d$ - a function of the decoder (d being the input and q the latent dimension).

What is substantial in the context of this work is that removing nonlinear operations from AE results in a latent space being identical to the one created by PCA [47]. This means that autoencoders are a generalization of PCA, and if implemented linearly, could be the PCA itself. This makes them a perfect choice for conducting experiments on them and comparing to the rest of the methods.

Autoencoders do face problems that can be problematic in certain applications. One of the main are being prone to overfitting, lacking explainability (due to the black-box nature), or being deterministic. While some of them can be aided by tricks or extensions (e.g., dropout for overfitting), these solutions are usually lacking in other areas.



(a) Autoencoder architecture



(b) Variational autoencoder architecture

Figure 3.3: A comparison of Autoencoder (a) and Variational Autoencoder (b) high-level architectures.

Variational autoencoders

Variational autoencoder (VAE) is a special example of an autoencoder [48]. Its goal remains the same; however, it is achieved not by learning functions and mapping a sample to a fixed latent vector but rather as a distribution over the latent space with a regularization term added to the reconstruction loss. This ensures the generative properties, inhibits overfitting, and improves the quality of the latent space properties.

Introducing variational approach to autoencoders is relatively similar to the pure ELBO (see paragraph about Variational Inference, especially Equation 2.10). Because both the encoder and decoder are in fact separate models, both distributions are conditional - $p(x \mid z)$ and $q(z \mid x)$ respectively, where x is the data and z is the latent vector. This way, the ELBO inequality (Equation 2.9) can be expressed as:

$$\log p(x) \geq \underbrace{\mathbb{E}_{z \sim q(z|x)} \left[\log p(x \mid z)\right]}_{\text{reconstruction loss}} - \underbrace{D_{KL} \left[q(z \mid x) \parallel p(z)\right]}_{\text{posterior approximation regularization}} \tag{3.13}$$

Therefore, maximizing the ELBO and minimizing the Kullback-Leibler divergence maximizes the log-evidence.

Reparameterization trick

Due to the latent variable being sampled from a distribution (instead of being deterministic), gradients cannot be backpropagated as in autoencoders. Similarly to Bayes by Backprop, a reparameterization trick needs to be used. Analogously, ϵ is drawn from a fixed distribution $\mathbb{N}(0, 1)$ and z takes a form of:

$$z = \mu + \sigma \odot \epsilon, \tag{3.14}$$

where $\odot$ is an element-wise product.

4 Experimental setting

Comparing different approaches does not always come down only to quantitative research. It is especially true for the ones providing more value than simply improving a score of a given metric. This is exactly the case of Bayesian methods — their main goal is usually not to compete with other models, but to provide additional information by capturing uncertainty. Due to the above, the conducted experiments focused on assessing not only the quantitative aspect, but also a comparative analysis in a broader sense.

4.1 Datasets

In order to minimize the number of variables during experiments, the chosen datasets were kept similar, both in terms of the appliable downstream tasks, but also attributes (such as sample size, modality, etc.). For the results to be reliable and not susceptible to bias, however, datasets, while being akin, need to provide enough variance. As proved by Cheplygina and Tax [49], this is usually the case, even for datasets from the same source.

Besides, it was desirable for the datasets to be well-established in the community. The reason is twofold: thorough knowledge about the dataset and avoiding unnecessary issues related to incorrectly created bespoke datasets (while there might be doubts about their flawlessness, such as annotation issues of MNIST, they are still much more robust and tested than experiment-specific ones).

Another crucial characteristics was the size of the dataset. The ones with too few samples would not be enough to train and test representations reliably. The chosen dataset needed to be large enough to eliminate this risk.

The choice was to select only image classification datasets, since it brings even more advantages, which the specificity of conducted experiments required, such as simple and intuitive visualizations of the learning process' intermediate steps, or broader space of possible experiments (e.g. translation or rotation invariance).

Three datasets with gradual complexity were chosen. This allowed for thorough examination of tested models, no matter their performance, and resulted in a more cross-sectional study. The following were used:

- MNIST [50, 51]

  A set of 70000 single-channel images of handwritten digits (split into proportion of

6 : 1 of train to test data). Their size is fixed to 28x28 pixels (although, for the purpose of some experiments, depending on the use case, its size was changed) and content centered. Due to its simplicity (relatively small size of images, uncomplicated content), large sample size (around 7000 instances per class) and portraying a real-world problem, it has become a default dataset for testing models.

- CIFAR-10 [52]

  A collection of 60000 RGB[1] photos (50000 in train and 10000 in test sets), which are a curated subset of the *80 Million Tiny Images*[2] data. Content of the 32x32 pixels images includes 10 mutually exclusive classes of real-world objects, much more complex than in MNIST. Because of that, the downstream task is also more intricate and presents an assay for simple classical approaches.

- CIFAR-100 [52]

  CIFAR-100 is just an extended version of the CIFAR-10 dataset (with the same base data, image parameters and train-to-test split ratio) — it yields 100 classes, but with only 600 examples per category. As a result of that, only the most modern and convoluted architectures achieve an accuracy score of over 90%. It posed the greatest challenge both to representation learning and evaluation methods.

4.2 Observational study

A diligent research expects an appropriate context and perspective, which is impossible to achieve without understanding how the used tools work in practice. For this reason, an observational study was conducted. In contrast to the Quantitative study, instead of comparing representations, it focuses more on providing information on the practical use, as well as detecting possible benefits and risks of the methods.

The implicit viewpoint of this work requires analyzing Bayesian representation learning methods — this means the Bayesian elements were treated with priority. Wherever possible, a reference to the feature learning was mentioned.

Because all three used methods differ significantly on the different datasets and procedures, the study of each particular is unique and differs from one another. While the examples (especially visualizations) shown are a result of a single run of a given experiment, they are an approximation representing numerous runs executed during the preparation of this study.

For the study brevity and clarity, only the MNIST dataset was used. If not specified otherwise, a stratified subset of 10000, 28x28 images served as a training set, with additional

[1] An additive color model that is operates on 3 channels representing colors — red, green and blue.

[2] The *80 Million Tiny Images* dataset has been formally withdrawn on 29th June 2020 as a consequence of offensive images and derogatory terms as categories found in it. [53]

1000 test samples in a test set where needed.

The results of this study can be found in Observational study.

4.3 Quantitative study

In order to assess the quality of the representations on a downstream task, a quantitative study was conducted. It aims to answer a question about how good the representations truly are in a real-world application.

All three pairs of methods were analyzed. At first, the representations were trained using specific methods. The training duration was 50 epochs for deep models (neural networks and autoencoders) and 5000 iterations for PCAs. In order to provide reliable results, training of each set of hyperparameters of every model was repeated 30 times, with different random seed values.

As the second step, all the representations were evaluated with 5 simple classifiers, each trained with 5-fold cross-validation: Logistic Regression, Support Vector Machine, Naive Bayes, Multilayer Perceptron and Decision Tree. The used metric was a *F1-score*, which was micro-averaged over classes.

In the end, all scores were gathered, mean and standard deviation were calculated and rounded to three decimal places. After that, the results were presented in Quantitative study results. In each table, there are specified dataset, model and possible additional parameters, along with the F1-score values.

In order to keep a balance between data used for learning representations and their evaluation, each test set of the datasets described in section 4.1 was designated to the latter task. That being said, the train parts became a dataset on their own. This means that in the feature learning stage they were split again to train, test, or validation sets according to the needs.

5 Observational study

5.1 Bayesian PCA

In this section, an observational study was conducted for Bayesian PCA. Only its most significant aspects related to being a Bayesian representation method were analyzed — some observations were only noted in the last section.

As a consequence of the characteristics of BPCA (explained throughout this section, and summarized in Further observations) for the purpose of the following experiments, the MNIST dataset was reduced to contain 1800 samples of size 8x8 (firstly, 2-pixel padding was removed and only then 20x20 images were scaled down).

To evaluate the methods reliably, if not stated differently, the following parameters were adopted:

- Number of components (PCA) — the result of automatic selection of the corresponding BPCA execution,
- number of iterations (BPCA) - 2500,
- gamma distribution hyperparameters (BPCA) — all set to 0.001.

5.1.1 Number of principal components

In conventional PCA, the number of principal components is determined by looking at the cumulative explained variance ratio and arbitrarily setting the cutoff threshold. This implies testing the retained variance for each of the components (or at least a large subset of them). As already mentioned (in the paragraph about Bayesian PCA of **??**), BPCA is capable of establishing the number of principal components automatically by estimating the effective dimensionality of the latent space.

The retrieval of this number can be done in two ways. First, count the nonzero vectors $\mathbf{w}_i$ of the matrix $\mathbf{W}$. Second, it can be inferred directly from $\boldsymbol{\alpha}$ (which is the vector of hyperparameters), because it influences the columns of the said matrix through a conditional distribution $p(\mathbf{W} \mid \boldsymbol{\alpha})$. Both approaches are mathematically related and easily visualizable.

Projection matrix visualization

The $\mathbf{W}$ matrix can be visualized using a *Hinton diagram*. It is a visualization technique for two-dimensional numerical arrays that represents its values as squares. The color of the

square indicates the sign of a number (white represents a positive number, black indicates negative), and the occupied area is proportional to the magnitude of the value.[1]

The result of fitting the Bayesian PCA model is shown in Figure 5.1. In the Hinton diagram of the projection matrix for the scaled 64-dimensional (images of size 8x8) MNIST dataset, 16 zeroed-out columns[2] indicate the discarded dimensions. The rest 48 is thus the number of principal components retained in the model. It will also be treated as a reference in later paragraphs.

Hyperparameter vector $\boldsymbol{\alpha}$

As already stated, because the columns of the matrix $\mathbf{W}$ are controlled by the α hyperparameter vector through the conditional distribution, the number of principal components can be inferred directly from it. The "zeroing-out" of a weight vector $\mathbf{w}$ (represented by a column in $\mathbf{W}$) occurs when the corresponding α has a large value. Setting a minimum threshold for the values defines the number of principal components (see Figure 5.2).

Due to the reasons mentioned in the previous paragraph, determining the number of principal components of the hyperparameter vector $\boldsymbol{\alpha}$ is more reliable. It is also the approach assumed in most implementations.

5.1.2 Inference

The used implementation of BPCA can track a component that is at the heart of the variational inference framework, *Evidence Lower Bound*. Although it is useful in tracking and evaluating the learning process, it can also provide important information on the underlying uncertainty (as explained in subsection 2.2.2)

Figure 5.3 presents the course of ELBO over the whole learning duration. The graph shows that the learning process stabilizes after $\sim 1000 - 1500$ epochs. Despite fluctuations (which are usually signs of attempts to continue exploring the search space), the plot seems to be approaching a constant asymptote.

5.1.3 Representation reconstruction

Both approaches to PCA have the possibility of inverting their transforms. In this manner, reconstruction of the original values is possible, and visual evaluation of the representations is feasible. Visualizations of data reconstruction by PCA and BPCA were presented in Figure 5.4.

[1] The original idea comes from David Warde-Farley on the SciPy Cookbook [54].

[2] Please note that some vectors' squares of the zeroed-out columns are slightly visible. The cause of this is that the values are not, in fact, equal to 0. They are considered to be "zeroed-out" if they are below a defined threshold (similarly to paragraph Hyperparameter vector $\boldsymbol{\alpha}$).

It can be noticed that while the latter is quite accurate (with only slight smudges around some numbers), the former does not always cope with it correctly. It is especially visible when comparing numbers 7, 8, and 9, where conventional PCA adds artifacts to the background. This is due to either imposing the number of principal components inferred from BPCA, or BPCA's latent approach yielding a better representation.

5.1.4 Further observations

Under particular conditions, Bayesian PCA behaves similarly to its conventional counterpart. It does, however, possess certain qualities that other cannot demonstrate, namely the ability to infer the number of principal components automatically, as well as insight into uncertainty handling.

While useful, it does not come without drawbacks. As the number of samples in a dataset grows, BPCA is susceptible to a kind of overfitting: the effective dimensionality of the latent space approaches the dimensionality of the dataset ($q_{eff} \to d - 1$). Although it copes well with higher dimensionalities of smaller datasets, the computational complexity grows very quickly, especially if one was to trace the log-likelihood and ELBO. This may still be better than a grid search over dimensionality selection (e.g., in a mixture distribution with multiple components), but is not a desirable characteristic. It was also noticed that BPCA is quite sensitive to the prior distribution parameters, which often lead to issues with overflowing or diminishing values.

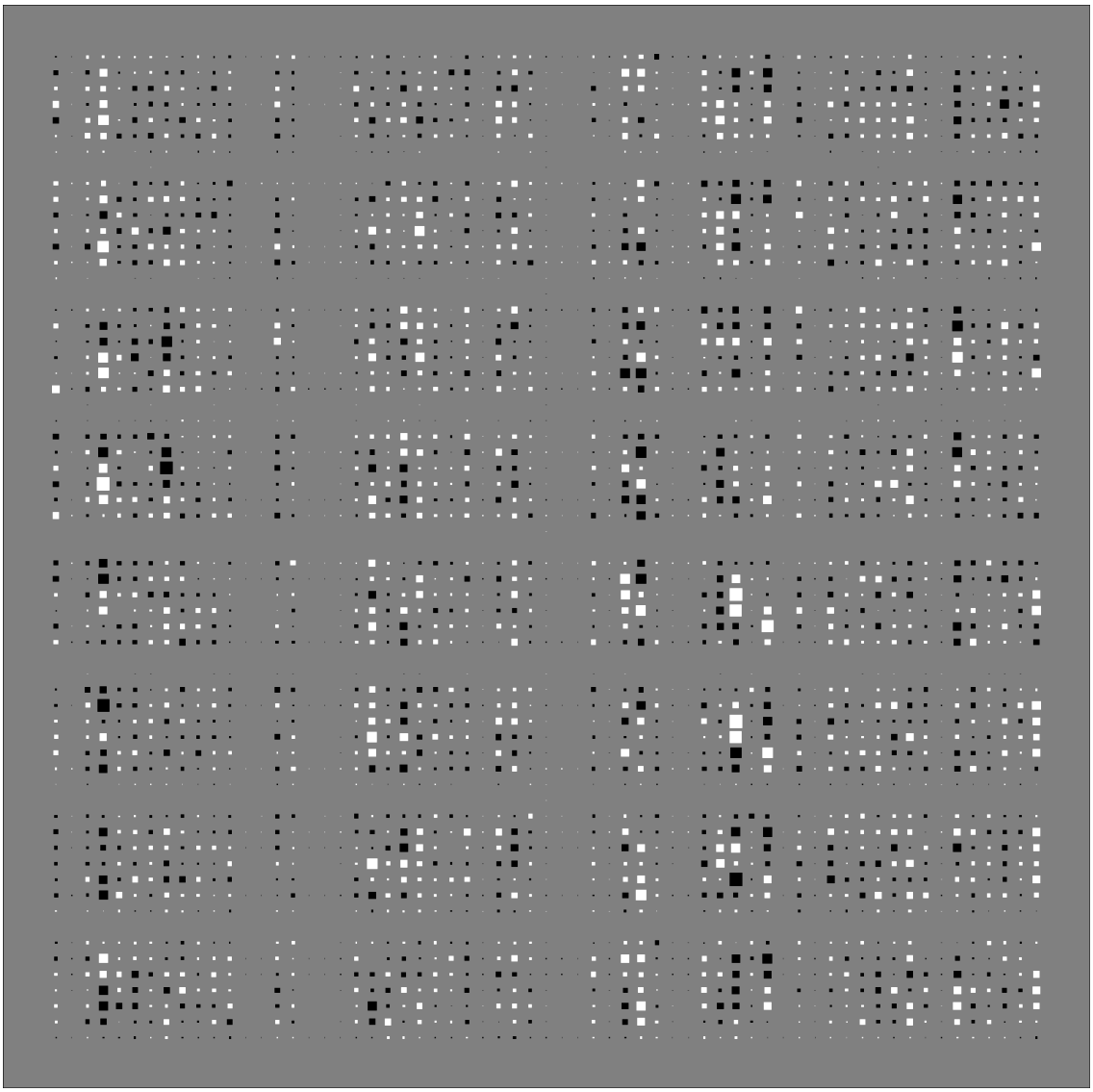


Figure 5.1: Hinton diagram of the BPCA projection matrix for a MNIST data set in 64 dimensions (8x8 digits). The model's effective dimensionality, and thus number of principal components, is $q_{eff} = 48$.

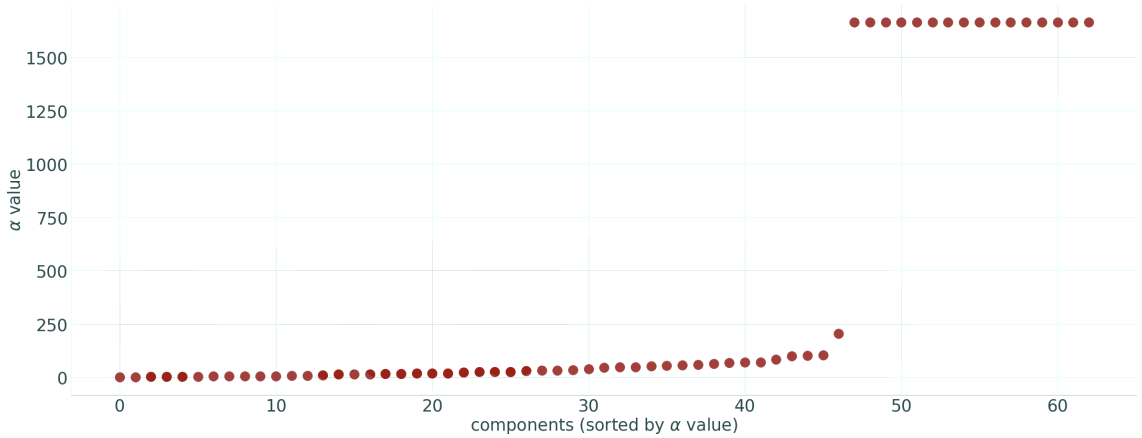
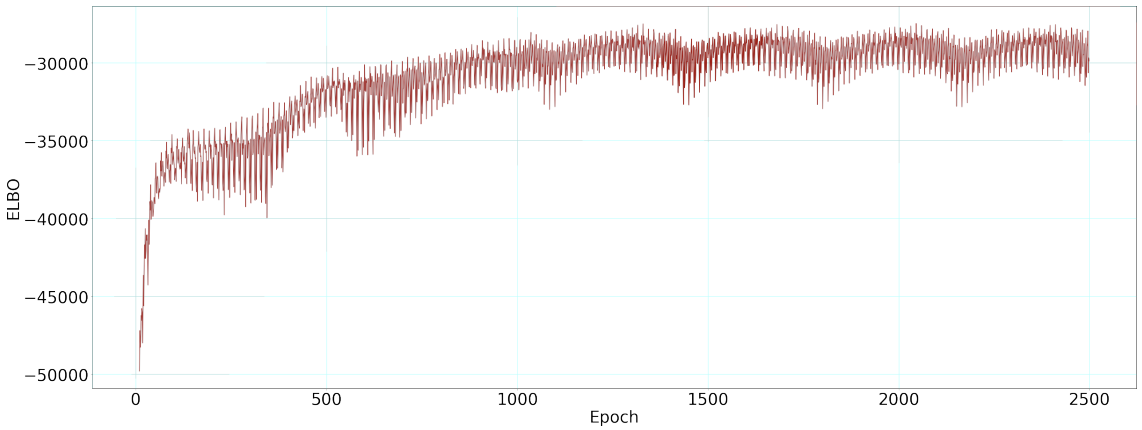
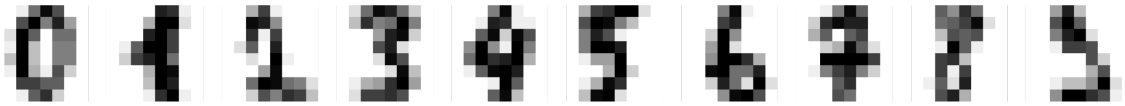


Figure 5.2: A spread of the hyperparameters α across components, sorted by their value in a non-decreasing order. Two groups of values are clearly distinguishable - "large", which zero-out columns of the $\mathbf{W}$ matrix, and "small", which correspond with principal components. Note that since the components are sorted by the value of α, the number on the x-axis might not necessarily overlap with their number in the search space.

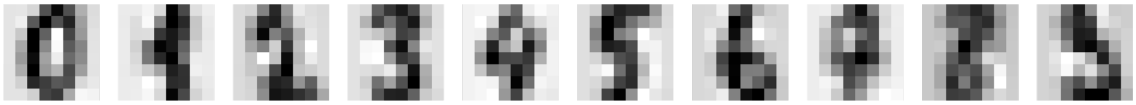


(a) ELBO over epochs

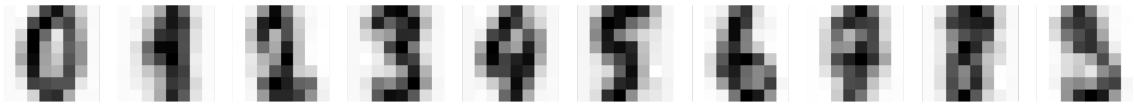
Figure 5.3: Graph shows Evidence Lower Bound values over model training epochs. Note that first 10 epochs were omitted for better legibility.



(a) Original data



(b) PCA reconstruction



(c) Bayesian PCA reconstruction

Figure 5.4: Visualizations of data reconstruction with PCA (b) and Bayesian PCA (c).

5.2 Bayesian Neural Network

As explained in section 3.2, in order to handle uncertainty, Bayesian Neural Networks introduce a few changes, some of which were analyzed in this section. If not defined otherwise, the following assumptions were made:

- the architecture used is a multilayer perceptron with 2 layers and a hidden size of 128,
- the prior distribution is a scale mixture of two Gaussian densities [26] with a *mixing coefficient* (π) equal to 0.5, $\mu_1 = \mu_2 = 0$, $\sigma_1 = 1$ and $\sigma_2 = 1 * 10^{-6}$,
- learning was conducted for 20 epochs, with batch size 64 and Adam optimizer with learning rate of $1 * 10^{-3}$,
- a subset of 10000 samples from the 28x28 MNIST dataset was used.

5.2.1 Learning process

In the case of Bayesian Neural Networks, the learning process is rather similar to their conventional counterpart. While the loss function is counted differently (see Equation 3.6), it is minimized the same way as in conventional neural networks. Similarly, with other measures, such as *accuracy* or *F1*. In Figure 5.5 sample graphs of loss and accuracy of BNN learning process were presented.

5.2.2 Uncertainty

One of the most significant features of Bayesian methods is a possibility of reaching some level of explainability through uncertainty handling. In Bayesian Neural Networks, if distributions are applied over weights, it is possible to quantify confidence of the network in each of the predicted weights. In contrast to conventional neural networks (where a *softmax* activation layer is used on non-probabilistic outputs to imitate this behavior [30, 55]), uncertainty estimation is based on true probabilities.

In Figure 5.6 an example uncertainty analysis was presented. It takes into consideration predicted classes, which are in fact extracted from the last layer only, because they are easily visualizable. Nevertheless, the same rules apply when measuring uncertainty for any of the layers, thus also any chosen representation. As a result of the Bayesian treatment, one can not only assess which classes the model confuses, but also to what extent. Additionally, sampling from a distribution allows for more robust and less computationally expensive evaluation. Because the inference in conventional neural networks is deterministic, only re-training the network multiple times would be equivalent to Monte Carlo draws in the Bayesian setting.

5.2.3 Prior distributions

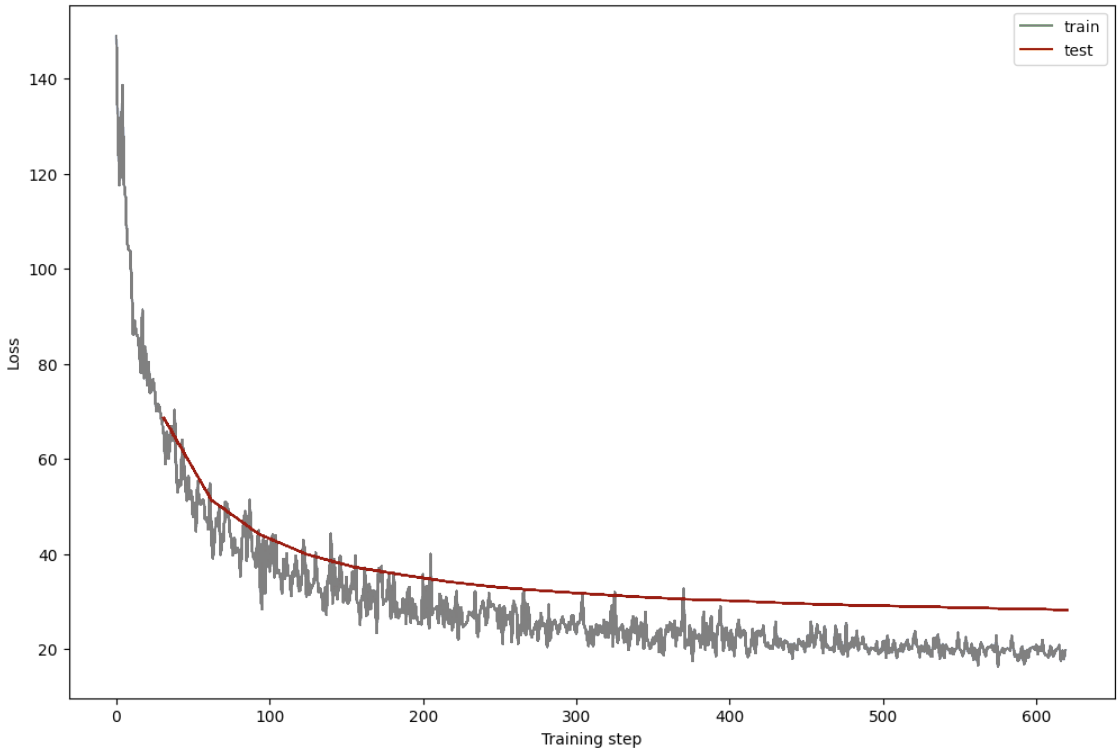
One of the advantages that Bayesian Neural Networks have over the conventional is taking into consideration the prior distribution of weights. Usually, however, the knowledge

is vague or non-existent. Thus, a study on sensitivity of the learning to the prior distribution parameters was conducted.

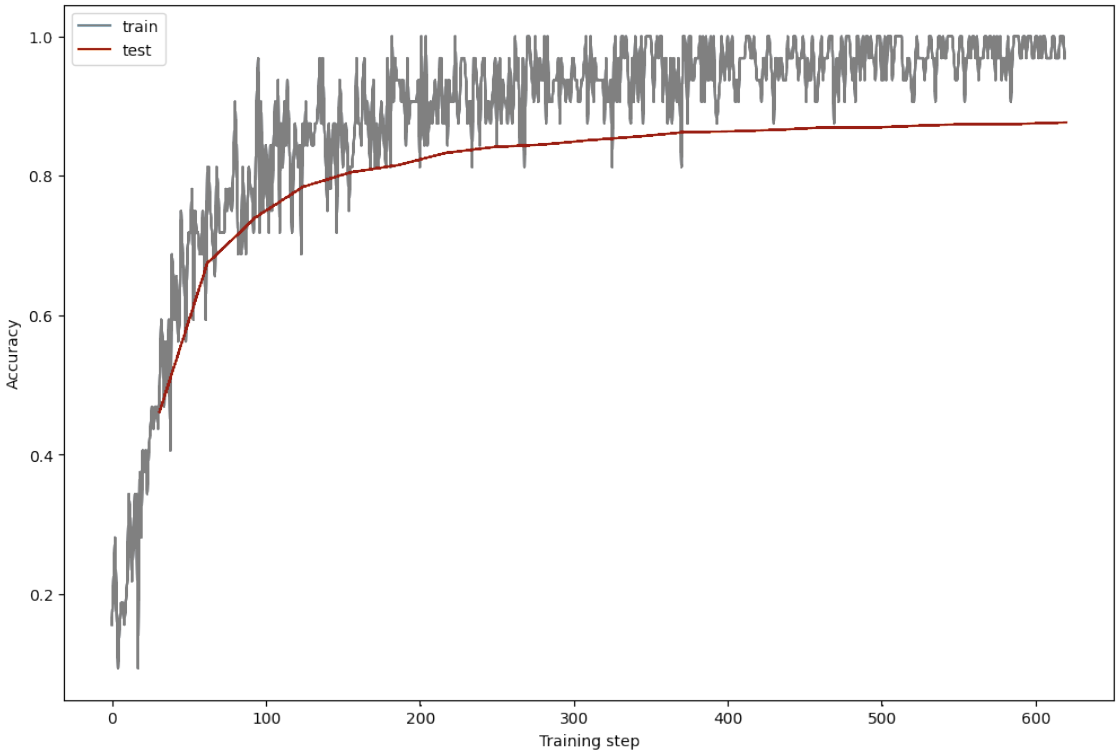
Figure 5.7 shows loss and accuracy curves for different sigma values. As can be seen, different sigma values do not influence the learning process greatly — the loss decreases and the accuracy increases. The curves are stable for both the train and test batch of data.

Figure 5.8 shows loss and accuracy curves for different mixing coefficient values. Similarly to the experiments with sigma parameters, also different proportions of the two distributions do not disrupt the learning process. This is including the two extreme values of 0 and 1, where only one of the distribution is active.

While the search was in no way exhaustive, the values used were wide-ranging. Based on the learning curves, a conclusion can be made that even though the parameters of the prior distributions vary, the method is apt to converge.

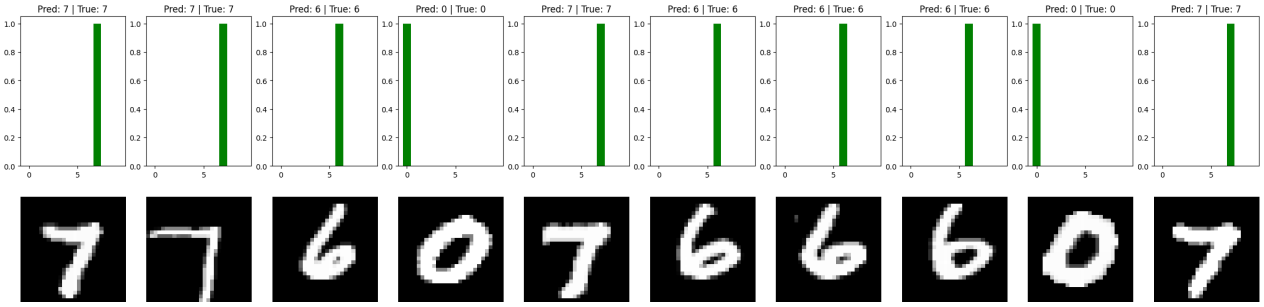


(a) Loss over epochs

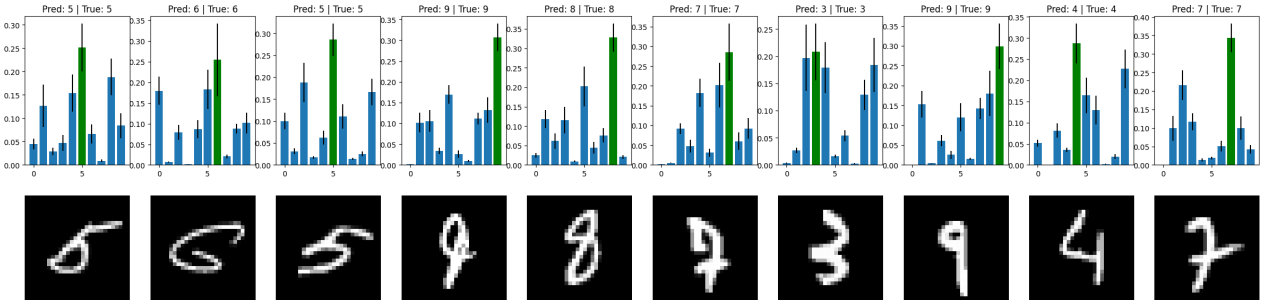


(b) Accuracy over epochs

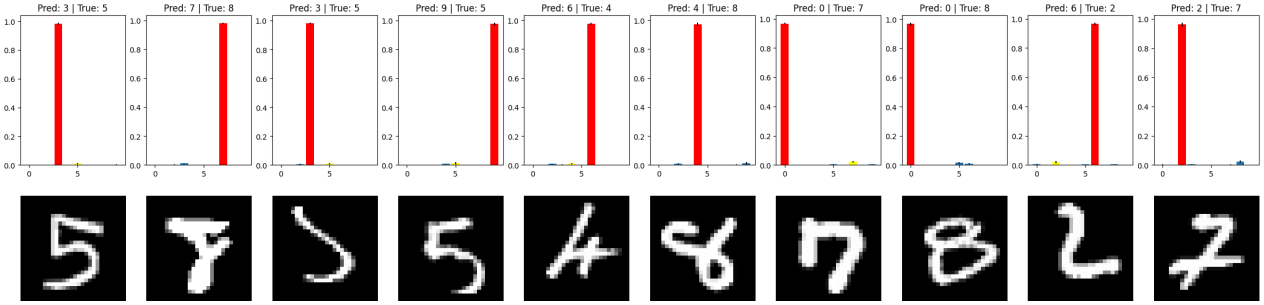
Figure 5.5: Graphs show the Loss (a) and Accuracy (b) values over BNN model training epochs.



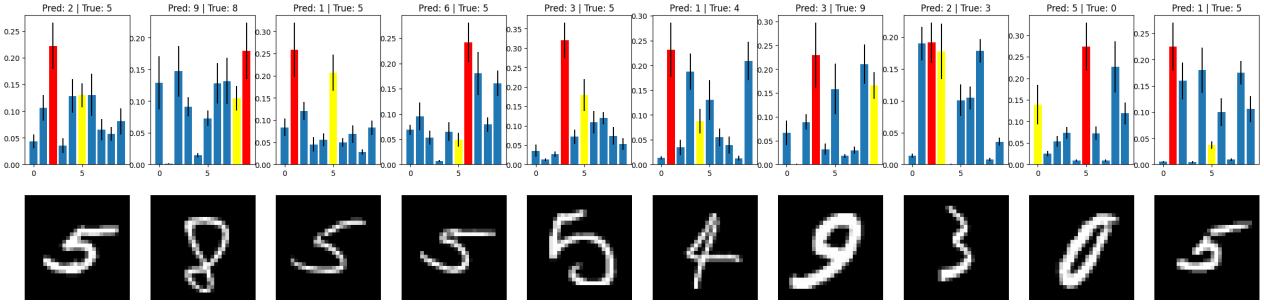
(a) Top 10 high confidence correct predictions



(b) Top 10 low confidence correct predictions

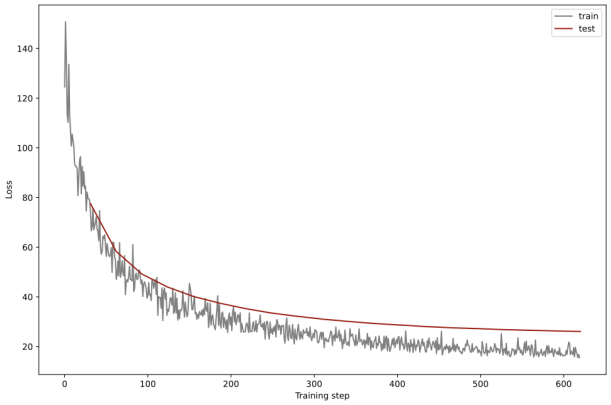


(c) Top 10 high confidence wrong predictions

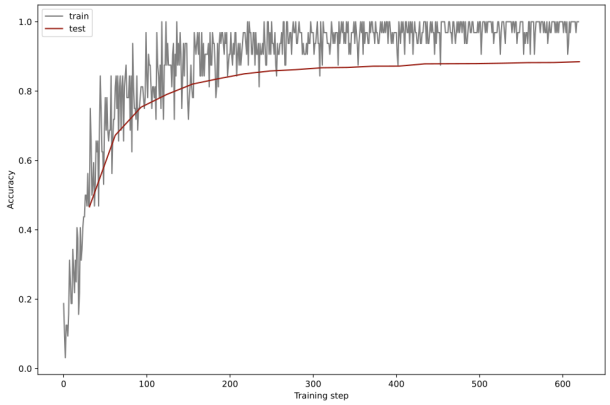


(d) Top 10 low confidence wrong predictions

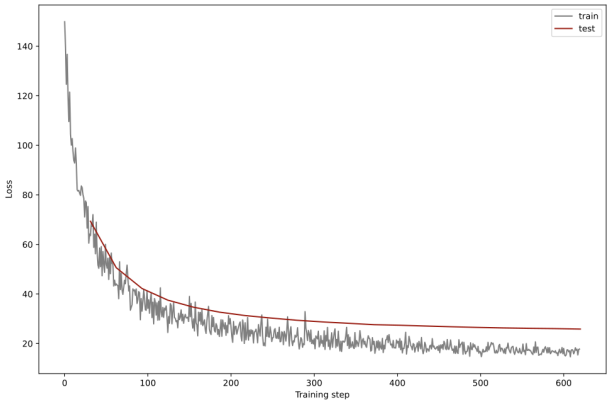
Figure 5.6: An example of uncertainty handling in Bayesian Neural Networks. Bar plots present correct and wrong predictions with the highest and lowest confidence. Green bars represent confidences in correctly predicted classes, red — confidences in incorrectly predicted classes, yellow — confidence in correct classes in case of an incorrect prediction, blue — confidences in other classes. Bars include *whiskers* to indicate maximum and minimum values over $N = 10$ Monte Carlo draws. Below each plot, there is an image of a corresponding sample.



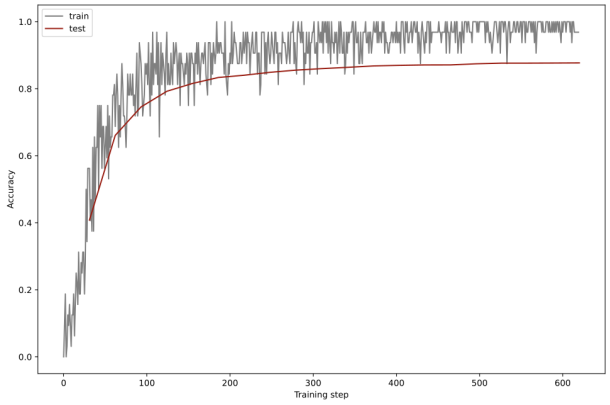
(a) Loss; $\sigma_1 = 1 * 10^{-1}$, $\sigma_2 = 1 * 10^{-6}$



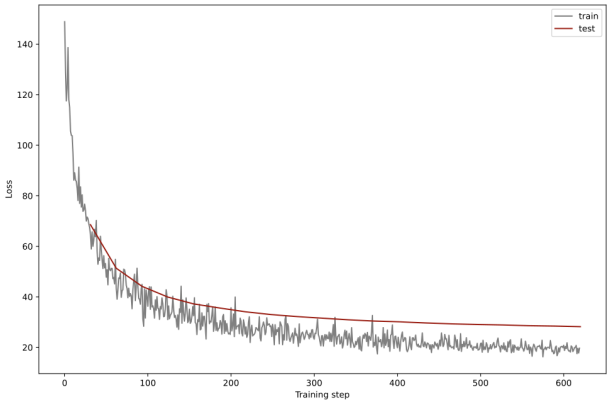
(b) Accuracy; $\sigma_1 = 1 * 10^{-1}$, $\sigma_2 = 1 * 10^{-6}$



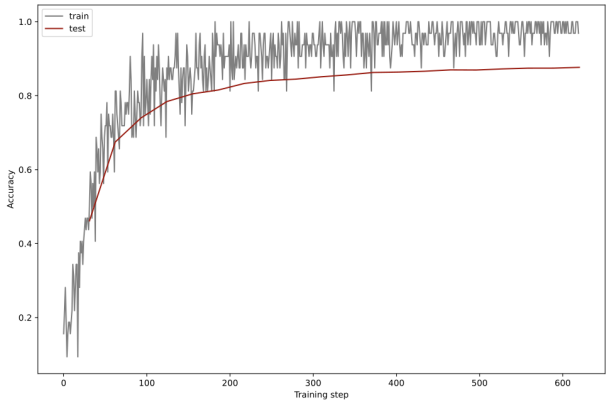
(c) Loss; $\sigma_1 = 5 * 10^{-1}$, $\sigma_2 = 1 * 10^{-6}$



(d) Accuracy; $\sigma_1 = 5 * 10^{-1}$, $\sigma_2 = 1 * 10^{-6}$

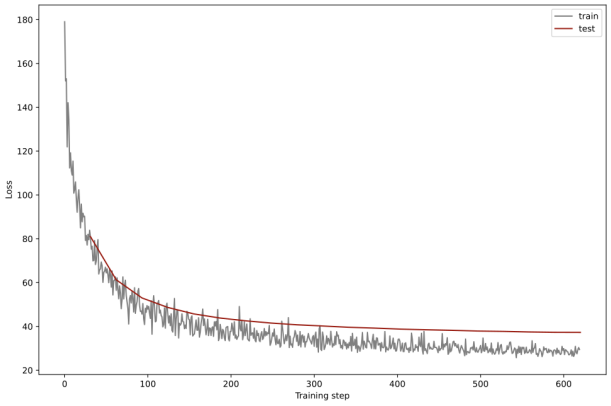


(e) Loss; $\sigma_1 = 1$, $\sigma_2 = 1 * 10^{-6}$

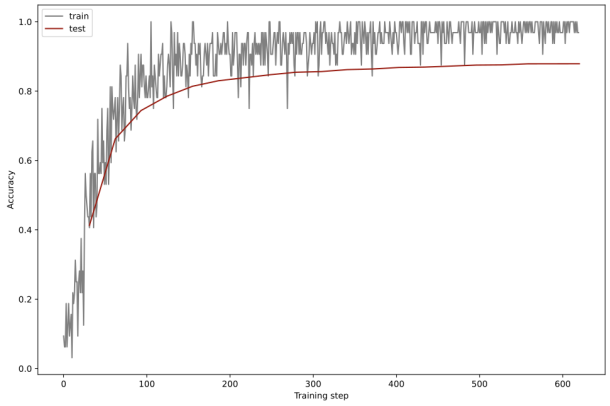


(f) Accuracy; $\sigma_1 = 1$, $\sigma_2 = 1 * 10^{-6}$

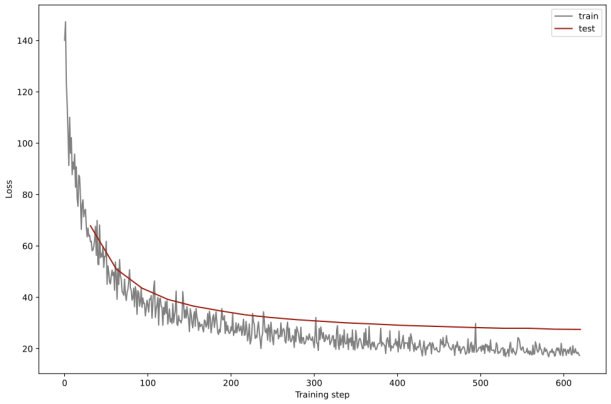
Figure 5.7: Sigma values influence on the Bayesian Neural Network learning process.



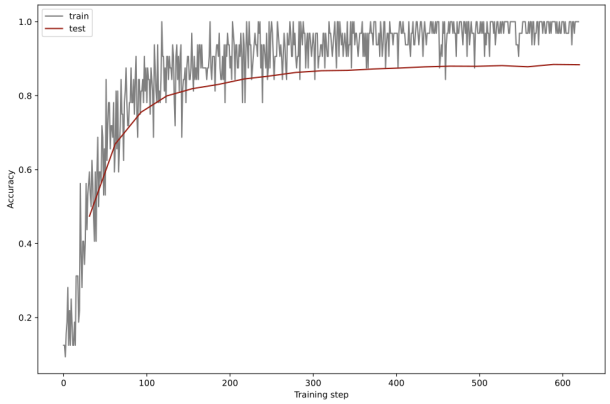
(g) Loss; $\sigma_1 = 1 * 10^1$, $\sigma_2 = 1 * 10^{-6}$



(h) Accuracy; $\sigma_1 = 1 * 10^1$, $\sigma_2 = 1 * 10^{-6}$

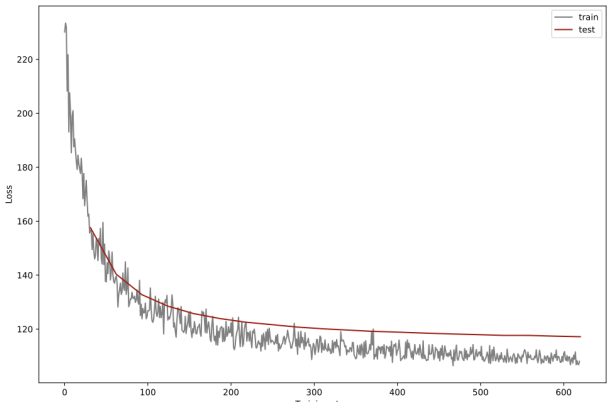


(i) Loss; $\sigma_1 = 1$, $\sigma_2 = 1 * 10^{-3}$

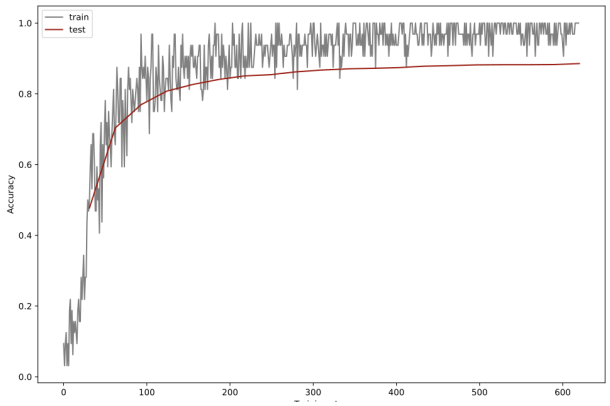


(j) Accuracy; $\sigma_1 = 1$, $\sigma_2 = 1 * 10^{-3}$

Sigma values influence on the Bayesian Neural Network learning process (cont.)

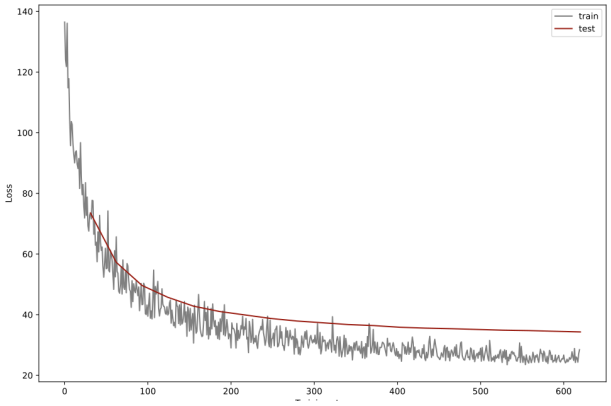


(a) Loss; $mc = 0$

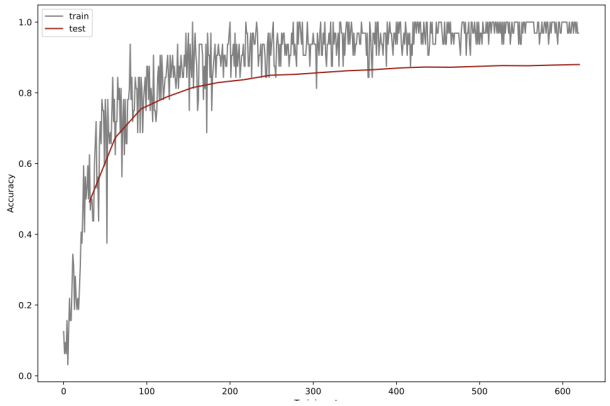


(b) Accuracy; $mc = 0$

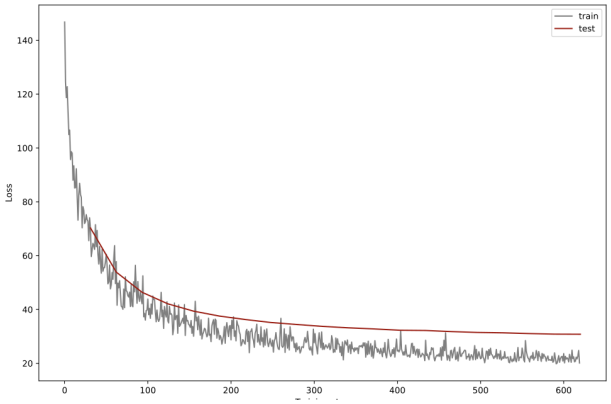
Figure 5.8: Mixing coefficient values influence on the Bayesian Neural Network learning process.



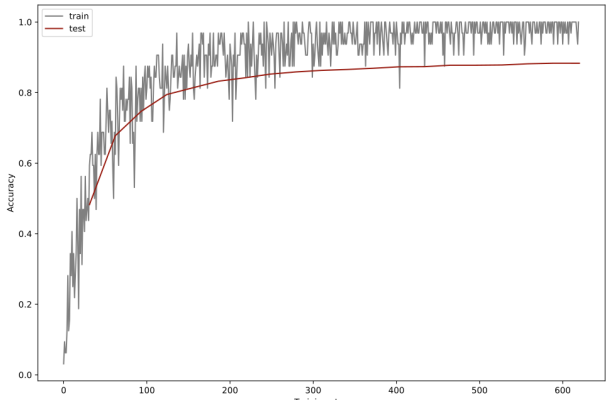
(c) Loss; $mc = 0.1$



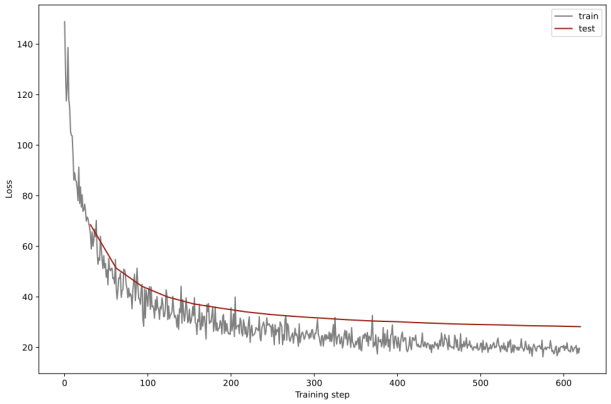
(d) Accuracy; $mc = 0.1$



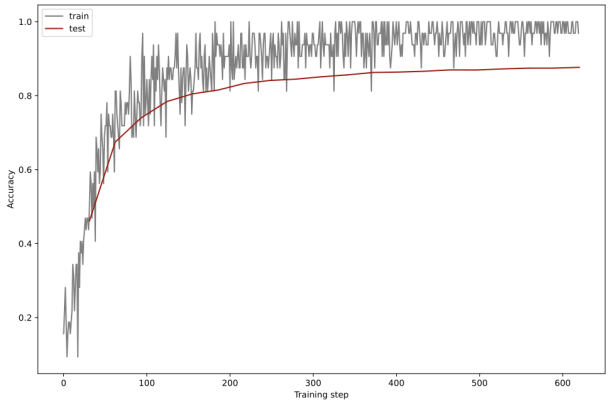
(e) Loss; $mc = 0.25$



(f) Accuracy; $mc = 0.25$

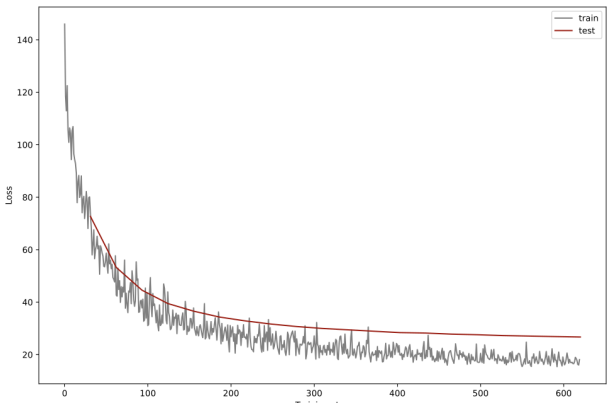


(g) Loss; $mc = 0.5$

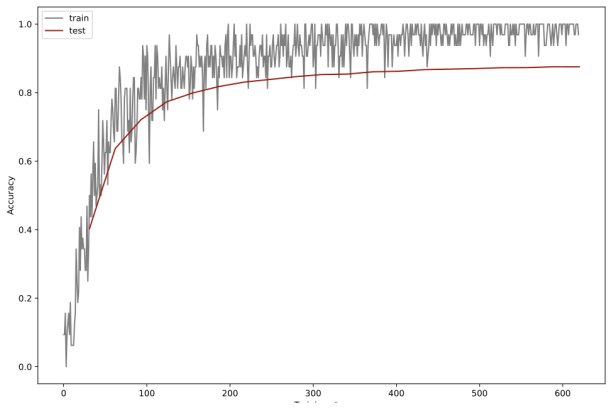


(h) Accuracy; $mc = 0.5$

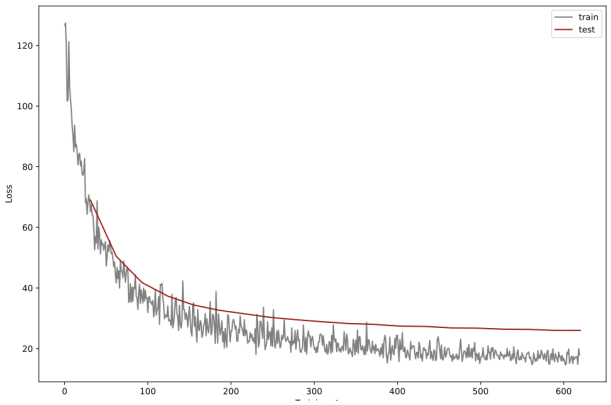
Figure 5.8: Mixing coefficient values influence on the Bayesian Neural Network learning process (cont.).



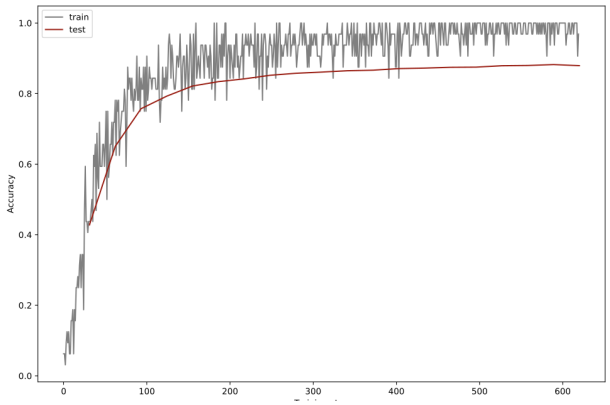
(i) Loss; $mc = 0.75$



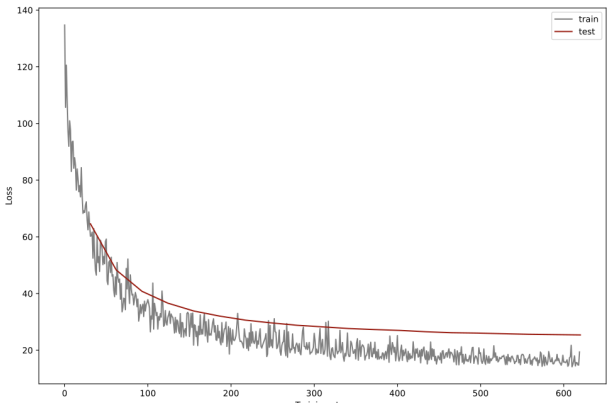
(j) Accuracy; $mc = 0.75$



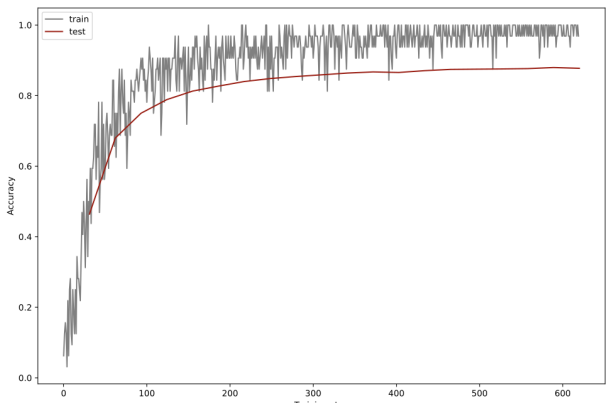
(k) Loss; $mc = 0.9$



(l) Accuracy; $mc = 0.9$



(m) Loss; $mc = 1$



(n) Accuracy; $mc = 1$

Figure 5.8: Mixing coefficient values influence on the Bayesian Neural Network learning process (cont.).

5.3 Variational Autoencoder

Variational Autoencoders, basing on the architecture of neural networks heavily, will have a lot in common in terms of the way they work. This section will focus on the aspects that are unique to VAE, and if possible — compare with AE. Again, useful assumptions were made:

- the architecture used for both encoder and decoder is a multilayer perceptron with 2 layers and a hidden size of 128, and the latent size (between the encoder and the decoder) of 10,
- learning was conducted for 20 epochs, with batch size 64 and Adam optimizer with learning rate of $1 * 10^{-3}$,
- a subset of 10000 samples from the 28x28 MNIST dataset was used.

5.3.1 Latent separability

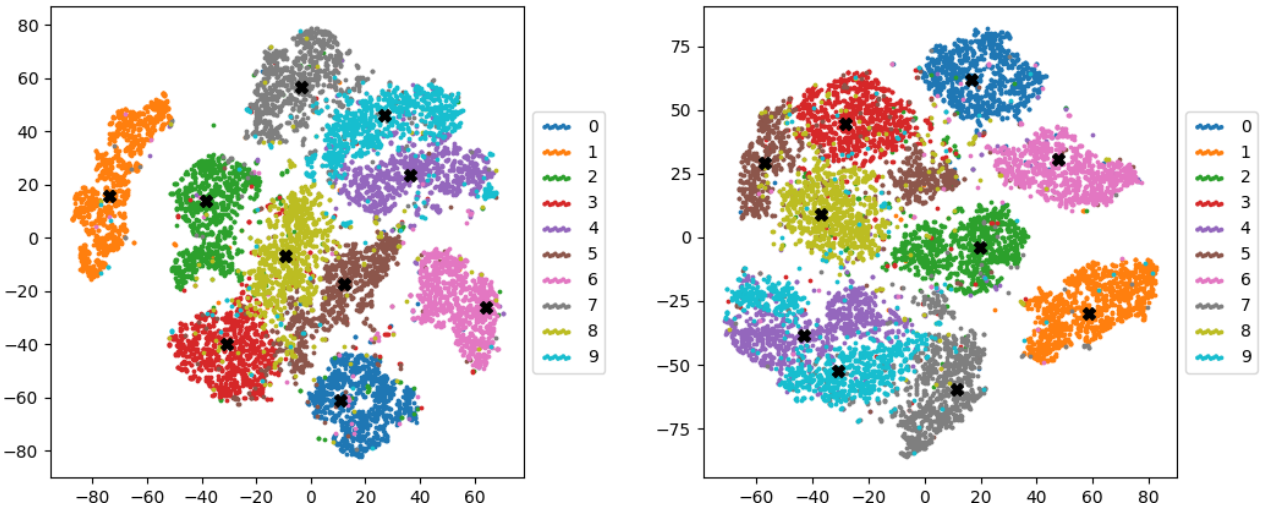
Latent separability in autoencoders shows how much the representation is able to distinguish between classes (please note that this is not identical to being disentangled, which needs a different type of measurement [56]). In Figure 5.9 a *t-distributed stochastic neighbor embedding* (t-SNE) of autoencoder and variational autoencoder representations were shown. Labels of particular classes were saved and annotated on the plots for easier comparability. While the main goal of t-SNE is to capture structure (in the sense that neighboring points in the input space will tend to be neighbors in the low dimensional space). While not deterministic, it preserves the clusters together (although it does not preserve distances) [57].

That being said, it can be seen that both AE and VAE do cope with latent separability similarly. Classes that are separated the best are 0, 1, 6. They also have issues with the same two groups of classes, and there is not much difference in intensity of the error.

5.3.2 Latent space quality

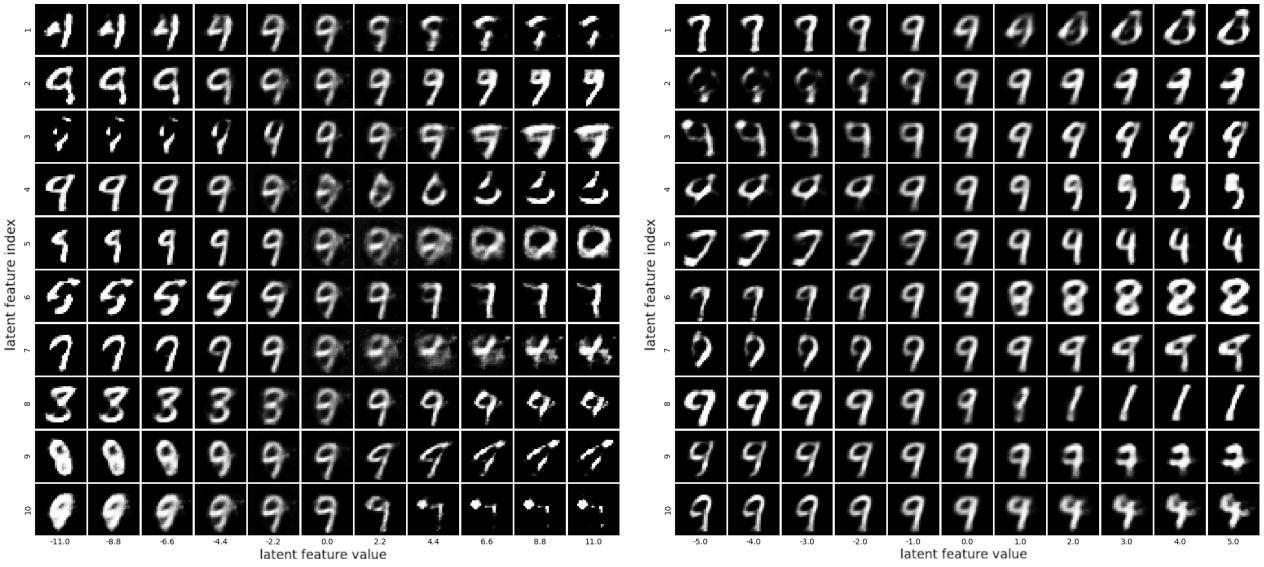
One of the most significant features of the well-trained autoencoders is being able to reconstruct the input from the latent space. Similarly to conventional neural networks, classical autoencoders cannot represent many latent vectors, as only one must be selected. Variational autoencoders, on the other hand, learn a distribution, thus (at least in theory) retain more information, which should result in better quality latent space, and thus input reconstruction.

This hypothesis proves true in latent space visualizations in Figure 5.10. They were prepared by reconstructing (passing through the decoder) the latent space. Presented images are a result of visualizing features for a latent code of digit 9. It is visible that particular squares are of higher quality in the VAE's space. The digits are more robust, well-defined and less blurry. Additionally, more of them resemble the actual digit 9.



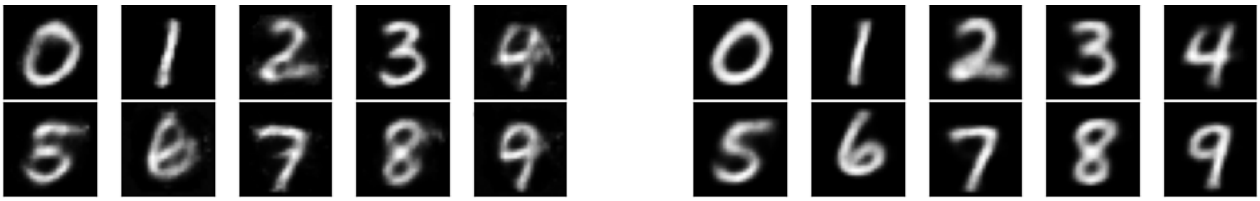
(a) t-SNE of the AE representation (b) t-SNE of the VAE representation

Figure 5.9: A comparison of Autoencoder (a) and Variational Autoencoder (b) t-SNE separation of representation.



(a) AE — latent visualization (b) VAE — latent visualization

Figure 5.10: A comparison of Autoencoder (a) and Variational Autoencoder (b) latent space visualizations. Presented images are a result of visualizing features for a latent code of digit 9.



(a) AE — average input reconstruction

(b) VAE — average input reconstruction

Figure 5.11: A comparison of Autoencoder (a) and Variational Autoencoder (b) average input reconstructions.

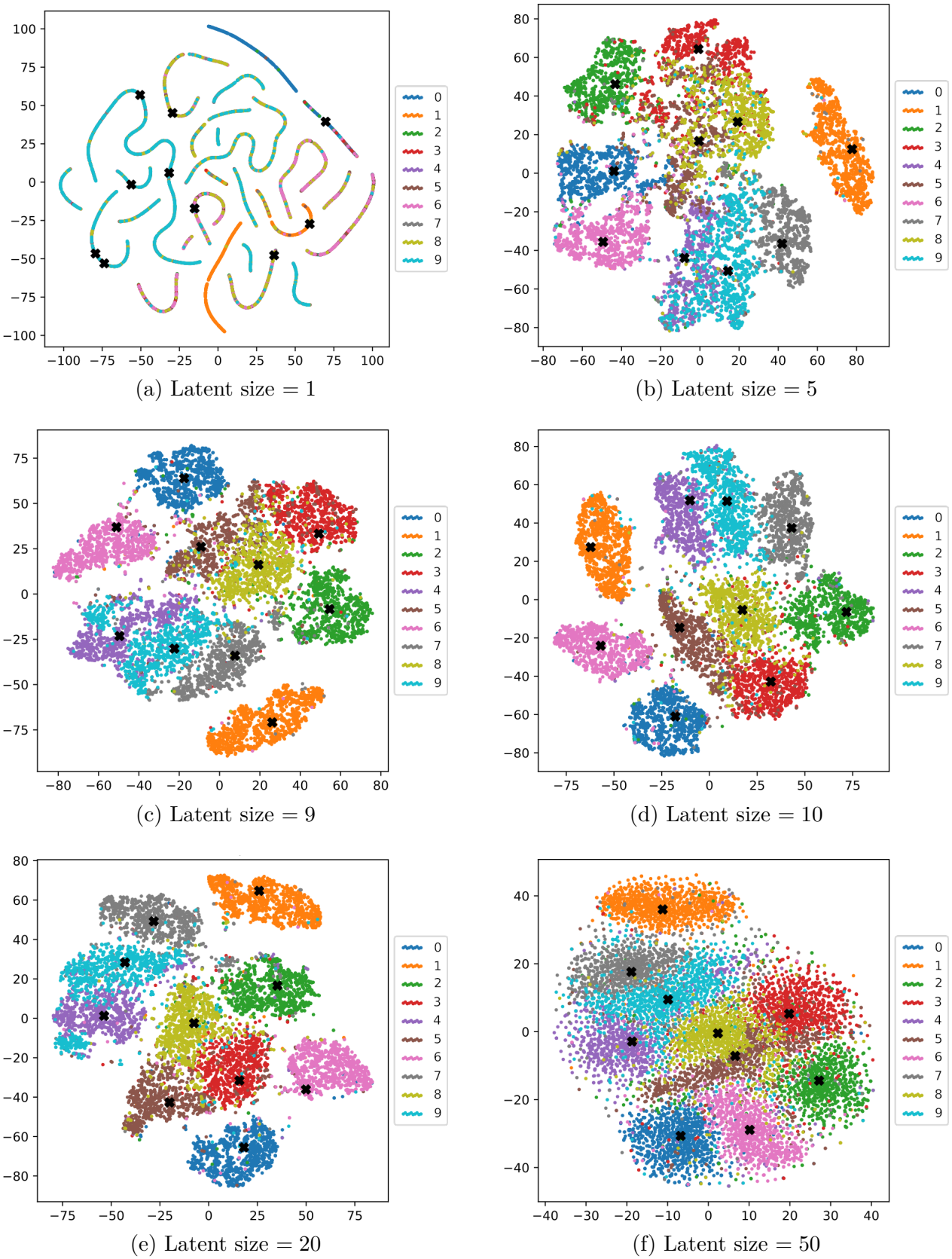
A similar observation can be made in Figure 5.11, where average reconstructions of the input were presented.

5.3.3 Latent size analysis

A discussion on the importance of latent representations cannot neglect an inherent aspect of it — the size and how it influences its quality. For that reason, two experiments have been conducted. Both separability and quality of the latent space was compared over different latent sizes.

Figure 5.12 shows t-SNE visualizations over a range of latent sizes. Too large values cause the clusters to scatter and fuse, which is a sign of poor separability — the larger the value, the bigger spread occurs. Excessively small values will obtain highly separated clusters. While the elongated curve-like shape of clusters for the size equal to 1 is not necessarily an issue (it is simply a result of how t-SNE processes data), the heavy mixing of classes within those clusters can raise some doubts.

To supplement the separability analysis, Figure 5.13 provides averages of input reconstructions for each class. In general, larger values yield better results (though a slight regression is visible for excessively large ones). With the increase of the latent size, the numbers seem less blurry and more recognizable (with the lowest values being almost illegible). Due to the trend being reverse to that from the separability analysis, the middle values seem optimal and because they are not fixed, the latent size should be treated as a hyperparameter.



(a) Latent size = 1

(b) Latent size = 5

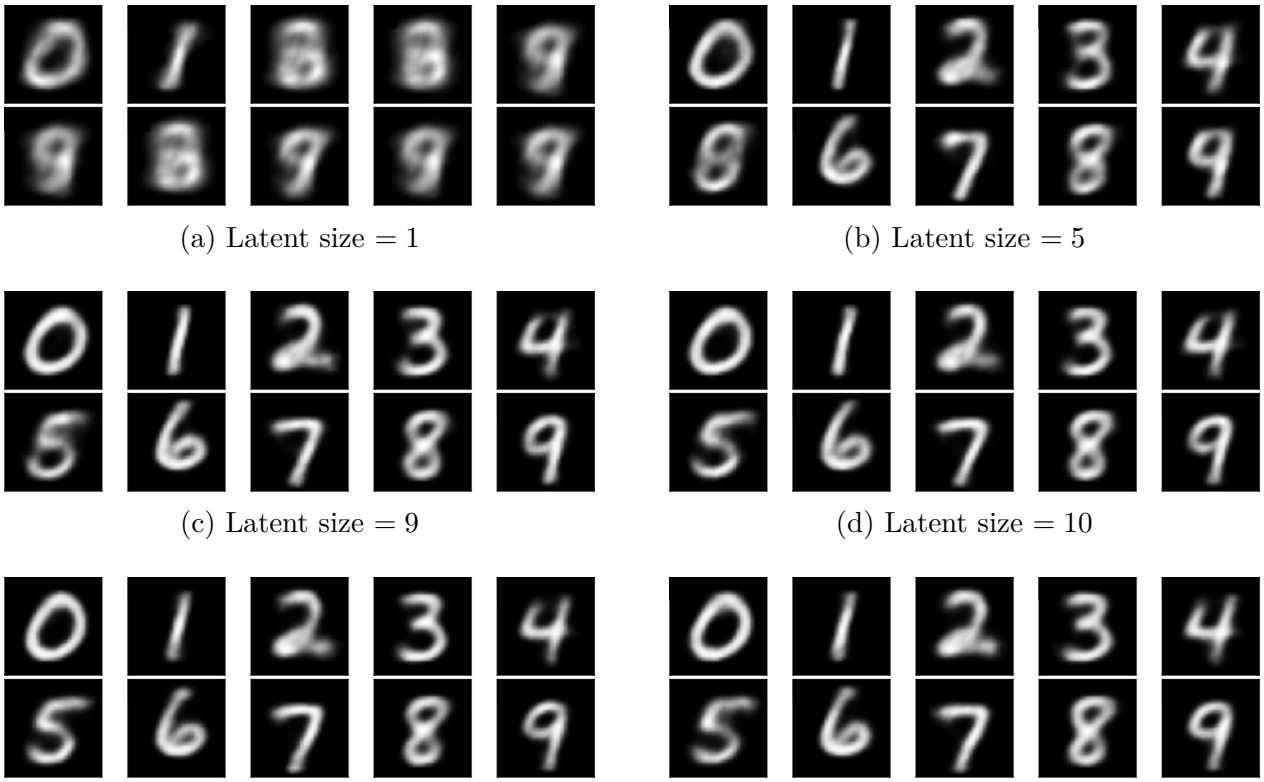
(c) Latent size = 9

(d) Latent size = 10

(e) Latent size = 20

(f) Latent size = 50

Figure 5.12: Latent size influence on Variational Autoencoder representation separability.



(a) Latent size = 1

(b) Latent size = 5

(c) Latent size = 9

(d) Latent size = 10

(e) Latent size = 20

(f) Latent size = 50

Figure 5.13: Latent size influence on Variational Autoencoder average input reconstruction.

6 Quantitative study results

To make this chapter more legible and easier to understand, a few assumptions were:

- Results shown in the tables are always sorted from the most simple, to the most complex dataset. They are denoted with their names capitalized.
- Names of the representation learning models were denoted with their abbreviations. For Principal Component Analysis and Neural Networks, their Bayesian counterparts got a *B* (for *Bayesian*) prefix; for Autoencoders, a *V* (for *Variational*) prefix.
- Classification results were presented in last five columns and denoted as follows:
  - Logistic Regression — LR,
  - Support Vector Machine — SVM,
  - Naive Bayes — NB,
  - Multilayer Perceptron — MLP,
  - Decision Tree — DT.
- Classifiers' scores were placed in their respective columns and denoted as *mean±std*.
- The best score for each dataset was written in bold.

6.1 Raw data

Learned representations should facilitate the training process, in relation to a raw dataset. For this reason, evaluations needed to be run on unprocessed data first. This set a reference point for further experiments. Table 6.1 shows results of this run. The results are surprising — models that are the best for one dataset might perform the worst in the other and vice versa. This, however, confirms a correct choice of the datasets and classifiers for evaluation.

Dataset	**LR**	**SVM**	**NB**	**MLP**	**DT**
MNIST	0.876±0.023	0.916±0.018	0.578±0.023	**0.908±0.019**	0.797±0.026
CIFAR10	0.284±0.011	0.286±0.013	**0.293±0.007**	0.1±0.0	0.246±0.006
CIFAR100	0.089±0.004	**0.144±0.003**	0.106±0.005	0.01±0.0	0.054±0.005

Table 6.1: Representation evaluation results for raw datasets.

6.2 PCA vs BPCA

In this experiment, both conventional Principal Component Analysis and its Bayesian counterpart implementations were executed with their respective default values. In the first place, BPCA was run to obtain the number of principal components, which was then fed to the corresponding PCA training.

Dataset	Model	LR	SVM	NB	MLP	DT
MNIST	BPCA	0.611±0.081	0.437±0.078	0.263±0.065	0.636±0.095	0.525±0.071
	PCA	0.249±0.302	0.741±0.025	0.176±0.125	**0.843±0.018**	0.727±0.026
CIFAR10	BPCA	0.337±0.032	0.217±0.045	0.23±0.031	**0.376±0.028**	0.214±0.015
	PCA	0.223±0.003	0.226±0.005	0.158±0.007	0.227±0.006	0.202±0.008
CIFAR100	BPCA	**0.097±0.033**	0.085±0.038	0.045±0.02	0.096±0.029	0.038±0.012
	PCA	0.051±0.006	0.05±0.004	0.046±0.003	0.036±0.004	0.046±0.003

Table 6.2: Representation evaluation results of Bayesian and conventional PCA.

Results of the experiment are presented in Table 6.2. While for MNIST the scores are lower than the reference (and BPCA achieves lower scores for some classifiers), for CIFAR10 and CIFAR100 they are similar or slightly better. BPCA is usually better than its conventional counterpart, however the scores are not as good as expected. This can be related to a hypothesis put in subsection 5.1.4. BPCA copes well with higher dimensionalities, but performs poorly for larger samples of data. This could result in a relatively large number of principal components being chosen by BPCA in each run.

Such phenomenon might occur due to the large sample size of each dataset. In table Table 6.3 minimum and maximum number of principal components chosen by BPCA for each dataset was presented. It confirms the unfortunate hypothesis.

Dataset (dim)	PC [min-max]
MNIST (784)	776 - 782
CIFAR10 (3072)	3068 - 3070
CIFAR100 (3072)	3069 - 3070

Table 6.3: Minimum and maximum dimensionality chosen by BPCA for each dataset. In braces, next to its name, the original dimensionality of a dataset was denoted.

6.3 NN vs BNN

Bayesian Neural Networks as a representation learning method in the literature is the most scarce of all three considered approaches. To balance this, this section is more extensive — two neural network architectures were tested and additionally to the planned experiments, a single hyperparameter study was conducted.

The two aforementioned architectures are simple Multilayer Perceptron and Convolutional Neural Network. The former consists of two, fully connected layers. The first mapping a flattened vector onto a hidden vector, and the other — yielding output classes from the previous. The CNN is simply a copy of LeCun's *LeNet* [51].

In this section, the following useful defaults were assumed:

- batch size = 64,
- hidden size = 84 (equal to the representation dimensionality),
- all activation functions are $ReLU$,
- starting learning rate = 0.00075,
- an Adam optimizer [31].

Table 6.4 and Table 6.5 show results of the experiments for the MLP and CNN, respectively. The first of them performs well — scores are predominantly better than the reference, and Bayesian models usually prevail too. CNN achieves even better results. Considering how simple this architecture is in comparison with modern advancements, the difference in scores in reference to raw data is tremendous (over a dozen percentage points of F1-score in some cases).

Dataset	**Model**	**LR**	**SVM**	**NB**	**MLP**	**DT**
MNIST	BMLP	**0.968±0.01**	0.966±0.008	0.916±0.024	0.966±0.009	0.867±0.022
	MLP	0.967±0.008	0.964±0.008	0.933±0.018	0.97±0.007	0.855±0.021
CIFAR10	BMLP	**0.475±0.011**	0.474±0.01	0.224±0.039	0.431±0.018	0.3±0.011
	MLP	0.467±0.013	0.465±0.013	0.385±0.015	0.444±0.012	0.293±0.012
CIFAR100	BMLP	0.156±0.012	**0.157±0.011**	0.026±0.008	0.131±0.013	0.075±0.005
	MLP	0.136±0.028	0.136±0.026	0.097±0.021	0.126±0.021	0.067±0.011

Table 6.4: Representation evaluation results of Bayesian and conventional Multilayer Perceptron.

Dataset	Model	LR	SVM	NB	MLP	DT
MNIST	BCNN	**0.989±0.004**	0.987±0.005	0.98±0.009	0.988±0.005	0.972±0.009
	CNN	0.989±0.005	0.987±0.006	0.983±0.007	0.989±0.005	0.97±0.01
CIFAR10	BCNN	**0.63±0.014**	0.625±0.014	0.575±0.016	0.572±0.024	0.474±0.015
	CNN	0.612±0.014	0.604±0.013	0.567±0.019	0.529±0.019	0.443±0.012
CIFAR100	BCNN	**0.263±0.014**	0.245±0.014	0.146±0.033	0.177±0.013	0.1±0.006
	CNN	0.255±0.01	0.236±0.01	0.176±0.015	0.176±0.011	0.103±0.007

Table 6.5: Representation evaluation results of Bayesian and conventional Convolutional Neural Network.

Additionally, a simple hyperparameter study was conducted for both architectures. The tested factor was the number of Monte Carlo samplings during the Bayesian inference. Table 6.6 and Table 6.7 show results for values 1, 10 and 20. It is visible that increasing the number of samplings produces better results, even if so slightly. Nevertheless, it is important to remember that it also causes longer inference times.

Although not necessarily straightforward, these results confirm that the internal representations created by neural networks in order to facilitate their own learning might be useful also to other models. In order to support this hypothesis, F1 scores from neural network models learning were presented in Table 6.8. It appears that not only the representations are useful, but they can also improve the results of subsequent classifiers.

Model	Dataset	Samples	LR	SVM	NB	MLP	DT
BMLP	MNIST	1.0	0.955±0.013	0.95±0.013	0.926±0.022	0.956±0.011	0.838±0.021
		10.0	**0.968±0.01**	0.966±0.008	0.916±0.024	0.966±0.009	0.867±0.022
		20.0	**0.968±0.01**	0.966±0.008	0.909±0.029	0.966±0.009	0.87±0.02
	CIFAR10	1.0	0.437±0.012	0.436±0.012	0.191±0.036	0.382±0.022	0.284±0.011
		10.0	0.475±0.011	0.474±0.01	0.224±0.039	0.431±0.018	0.3±0.011
		20.0	**0.476±0.01**	0.475±0.011	0.237±0.041	0.437±0.013	0.298±0.01
	CIFAR100	1.0	0.142±0.007	0.142±0.008	0.023±0.006	0.118±0.009	0.07±0.004
		10.0	0.156±0.012	0.156±0.011	0.026±0.008	0.131±0.013	0.075±0.005
		20.0	0.16±0.011	**0.161±0.01**	0.033±0.009	0.143±0.011	0.075±0.006

Table 6.6: Representation evaluation results for different sampling counts of Bayesian Multilayer Perceptron.

Model	Dataset	Samples	LR	SVM	NB	MLP	DT
BCNN	MNIST	1.0	0.983±0.007	0.981±0.007	0.972±0.01	0.983±0.006	0.954±0.013
		10.0	**0.989±0.004**	0.987±0.005	0.98±0.009	0.988±0.005	0.972±0.009
		20.0	**0.989±0.004**	0.987±0.005	0.973±0.009	0.988±0.005	0.976±0.009
	CIFAR10	1.0	0.586±0.014	0.576±0.015	0.539±0.021	0.471±0.016	0.434±0.015
		10.0	0.63±0.014	0.625±0.014	0.575±0.016	0.572±0.024	0.474±0.015
		20.0	**0.634±0.013**	0.629±0.013	0.571±0.014	0.590.016	0.479±0.013
	CIFAR100	1.0	0.229±0.017	0.213±0.016	0.13±0.023	0.15±0.016	0.094±0.007
		10.0	0.263±0.014	0.245±0.014	0.146±0.033	0.177±0.013	0.1±0.006
		20.0	**0.273±0.009**	0.255±0.009	0.172±0.019	0.188±0.008	0.102±0.007

Table 6.7: Representation evaluation results for different sampling counts of Bayesian Convolutional Neural Network.

Dataset	Model	F1-score
MNIST	BCNN	0.983±0.001
	BMLP	0.958±0.002
	CNN	**0.989±0.001**
	MLP	0.975±0.001
CIFAR10	BCNN	**0.626±0.009**
	BMLP	0.465±0.006
	CNN	0.613±0.011
	MLP	0.461±0.006
CIFAR100	BCNN	**0.304±0.013**
	BMLP	0.162±0.011
	CNN	0.293±0.007
	MLP	0.136±0.028

Table 6.8: Neural Networks F1-score on a test set.

6.4 AE vs VAE

The architecture of both versions of the autoencoder base on ResNet18 [58]. Again, the following defaults were assumed:

- batch size = 64,
- hidden size = 256 (equal to the representation dimensionality),
- starting learning rate = 0.00075,
- an Adam optimizer [31].

In Table 6.9 there are results of the experiments on Autoencoder and Variational Autoencoder. What is unanticipated is the representations produced by the former coped better with all the dataset and classifier combinations. Furthermore, scores of the Variational Autoencoder are often slightly worse than the reference. This might have been caused by many factors, however, the most suspected is lack of hyperparameter optimization, which is out of the scope of this work[1]. Unfortunately, on top of being complex theoretically, autoencoders (especially variational) are computationally expensive.

Dataset	Model	LR	SVM	NB	MLP	DT
MNIST	AE	**0.922±0.014**	0.909±0.018	0.844±0.034	0.954±0.012	0.706±0.029
	VAE	0.806±0.036	0.791±0.027	0.815±0.032	0.783±0.034	0.679±0.031
CIFAR10	AE	**0.456±0.016**	0.445±0.014	0.365±0.016	0.412±0.014	0.198±0.008
	VAE	0.266±0.011	0.246±0.01	0.275±0.011	0.205±0.009	0.192±0.008
CIFAR100	AE	0.155±0.009	0.145±0.009	**0.162±0.009**	0.126±0.009	0.041±0.005
	VAE	0.044±0.005	0.042±0.005	0.052±0.005	0.035±0.005	0.037±0.004

Table 6.9: Representation evaluation results of Variational and conventional Autoencoder.

6.5 Conclusions

Tested representation learning approaches do prove to be useful for downstream tasks of other classificators. Bayesian methods, besides the advantages enumerated in chapter 3 and chapter 5, often bring better results than their conventional counterparts. This is partly due to the way representations are produced, which has higher information retention. The latent space itself also has better properties due to natural regularization. It is important, however, to carefully adjust hyperparameters, especially those responsible for the Bayesian inference.

[1]Please note that before starting the quantitative study, a limited hyperparameter optimization was conducted. Although not exhaustive, it was deemed sufficient to proceed.

7 Conclusions and further work

All the goals defined in section 1.2 were met across this work, particularly, the main objective of introducing, analyzing and testing a cross-section of Bayesian representation learning models. In chapter 2 an extensive literature review and theoretical approach to Bayesian representation learning were provided. chapter 3 presented theoretical background for the 3 chosen Bayesian representation learning methods, their observational analysis was conducted in chapter 5 and the quantitative study of the downstream task testing in chapter 6.

From the conducted experiments, important conclusions need to be made:

- Bayesian treatment enables real quantification of uncertainty, which is unavailable for conventional methods. It encourages better model interpretability.
- Bayesian representation learning methods preserve more important information about the input. By naturally regularizing the latent space, they obtain its better properties. This directly translates to better results of representation evaluation on downstream tasks.
- Seemingly simpler Bayesian algorithms, such as BPCA, are not necessarily quicker. Often they do not provide a way for batch processing, which requires operating on a whole dataset at once, which is not optimal.
- Some Bayesian models, similarly to conventional, might be sensitive to hyperparameters. If possible, an optimization is recommended for satisfactory results.
- Bayesian approach, while saving computational effort in a global scale (by learning distributions, and not point estimates), is more expensive from the perspective of training individual models. For analytical purposes, where large numbers of experiments need to be run, it is recommended to use well-established implementations, ideally GPU-enabled or at least concurrency-enabled.

While diligent, the conducted study is not exhaustive. The topic would benefit from the following future work and improvements:

- adding more datasets with different downstream tasks (e.g., regression),
- considering different aspects of measuring representations quality (e.g., disentanglement, geometric priors, etc.),
- testing more diverse and advanced models (semi-supervised, contrastive, etc.),
- implementing ablation study (both on model elements and data),

- further hyperparameter optimization, with an emphasis on prior distributions, but also improving VAE's quality.

Bibliography

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1798–1828, 6 2012.

[2] O. Vilarroya, "Neural representation. a survey-based analysis of the notion," *Frontiers in Psychology*, vol. 8, 2017.

[3] D. Marr, *Vision: a computational investigation into the human representation and processing of visual information*. W.H. Freeman, 1982.

[4] M. Hazewinkel, *Encyclopaedia of Mathematics*. Springer Netherlands, 1990.

[5] G. Ifrah, *The Universal History of Numbers: From Prehistory to the Invention of the Computer*. Harvill, 1998.

[6] J. A. F. Thompson, Y. Bengio, E. Formisano, and M. Schönwiesner, "How can deep learning advance computational modeling of sensory information processing?," 9 2018.

[7] H. Barlow, "Redundancy reduction revisited," *Network: computation in neural systems*, vol. 12, p. 241, 2001.

[8] H. B. Barlow, "Sensory mechanisms, the reduction of redundancy, and intelligence," *Mechanisation of thought processes*, 1959.

[9] J. J. DiCarlo and D. D. Cox, "Untangling invariant object recognition," *Trends in Cognitive Sciences*, vol. 11, pp. 333–341, 8 2007.

[10] J. DiCarlo, D. Zoccolan, and N. Rust, "How does the brain solve visual object recognition?," *Neuron*, vol. 73, pp. 415–434, 2 2012.

[11] A. Szabo, Q. Hu, T. Portenier, M. Zwicker, and P. Favaro, "Challenges in disentangling independent factors of variation," 11 2017.

[12] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *arXiv preprint arXiv:2104.13478*, 2021.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[14] E. Knudsen, "Supervised learning in the brain," *The Journal of Neuroscience*, vol. 14, pp. 3985–3997, 7 1994.

[15] H. B. Barlow, "Unsupervised learning," *Neural computation*, vol. 1, pp. 295–311, 1989.

[16] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, pp. 307–392, 2019.

[17] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, "Using self-supervised learning can improve model robustness and uncertainty," *Advances in Neural Information Processing Systems*, vol. 32, 6 2019.

[18] X. Liu, F. Zhang, Z. Hou, Z. Wang, L. Mian, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE Transactions on Knowledge and Data Engineering*, 6 2020.

[19] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, pp. 559–572, 1901.

[20] H. Hotelling, "Relations between two sets of variates," *Biometrika*, vol. 28, pp. 321–377, 1936.

[21] C. Bishop, "Bayesian pca," vol. 11, MIT Press, 1998.

[22] M. Bayes and M. Price, "An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s.," *Philosophical Transactions (1683-1775)*, vol. 53, pp. 370–418, 1763.

[23] D. J. C. MacKay, "A practical bayesian framework for backpropagation networks," *Neural Computation*, vol. 4, pp. 448–472, 1992.

[24] D. J. C. Mackay, "Probable networks and plausible predictions - a review of practical bayesian methods for supervised neural networks," *Network: Computation in Neural Systems*, vol. 6, pp. 469–505, 1995.

[25] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, "Hands-on bayesian neural networksa tutorial for deep learning users," *IEEE Computational Intelligence Magazine*, vol. 17, pp. 29–48, 5 2022.

[26] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *32nd International Conference on Machine Learning, ICML 2015*, vol. 2, pp. 1613–1622, 5 2015.

[27] A. G. Wilson and P. Izmailov, "Bayesian deep learning and a probabilistic perspective of generalization," vol. 33, pp. 4697–4708, Curran Associates, Inc., 2020.

[28] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," 2015.

[29] D. Chen, X. Cao, L. Wang, F. Wen, and J. Sun, "Bayesian face revisited: A joint formulation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7574 LNCS, pp. 566–579, 2012.

[30] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," vol. 48, pp. 1050–1059, PMLR, 8 2016.

[31] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 12 2014.

[32] S. Wang, L. Duan, N. Yang, and J. Dong, "Person re-identification with deep dense feature representation and joint bayesian," *Proceedings - International Conference on Image Processing, ICIP*, vol. 2017-September, pp. 3560–3564, 2 2018.

[33] K. Nozawa, P. Germain, and B. Guedj, "Pac-bayesian contrastive unsupervised representation learning," vol. 124, pp. 21–30, PMLR, 3 2020.

[34] Z. Meng, R. McCreadie, C. Macdonald, and I. Ounis, "Variational bayesian representation learning for grocery recommendation," *Information Retrieval Journal*, vol. 24, pp. 347–369, 2021.

[35] O. Barkan, A. Caciularu, I. Rejwan, O. Katz, J. Weill, I. Malkiel, and N. Koenigstein, "Representation learning via variational bayesian networks," *International Conference on Information and Knowledge Management, Proceedings*, pp. 78–88, 10 2021.

[36] J. A. Zavatone-Veth, A. Canatar, B. Ruben, and C. Pehlevan, "Asymptotics of representation learning in finite bayesian neural networks," 2021.

[37] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society, series B*, vol. 61, pp. 611–622, 1999.

[38] S. Roweis, "Em algorithms for pca and spca," vol. 10, MIT Press, 1997.

[39] B. Fruchter, *Introduction to factor analysis.* Van Nostrand, 1954.

[40] H. H. Harman, *Modern factor analysis.* University of Chicago press, 1976.

[41] D. Child, *The essentials of factor analysis.* Cassell Educational, 1990.

[42] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, pp. 1–22, 9 1977.

[43] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Factor analysis, probabilistic principal component analysis, variational inference, and variational autoencoder: Tutorial and survey," 1 2021.

[44] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *CoRR*, vol. abs/1706.04599, 2017.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature 1986 323:6088*, vol. 323, pp. 533–536, 1986.

[46] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," 2020.

[47] E. Plaut, "From principal subspaces to principal components with linear autoencoders," 2018.

[48] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[49] V. Cheplygina and D. M. J. Tax, "Characterizing multiple instance datasets," pp. 15–27, 7 2018.

[50] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[51] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2323, 1998.

[52] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[53] A. Birhane and V. U. Prabhu, "Large image datasets: A pyrrhic win for computer vision?," *Proceedings - 2021 IEEE Winter Conference on Applications of Computer Vision, WACV 2021*, pp. 1536–1546, 1 2021.

[54] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, I. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . Contributors, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[55] T. Pearce, A. Brintrup, and J. Zhu, "Understanding softmax confidence and uncertainty," *CoRR*, vol. abs/2106.04972, 2021.

[56] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," 2017.

[57] L. der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, 2008.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," pp. 770–778, 2016.

[59] M. J. Kochenderfer, T. A. Wheeler, and K. H. Wray, *Algorithms for Decision Making*. MIT PRESS, 2022.

[60] Z. Meng, R. McCreadie, C. Macdonald, and I. Ounis, "Variational bayesian context-aware representation for grocery recommendation," 9 2019.

[61] N. Friedman, "The bayesian structural em algorithm," 1 2013.

[62] G. Zhong, L. N. Wang, X. Ling, and J. Dong, "An overview on data representation learning: From traditional feature learning to recent deep learning," *Journal of Finance and Data Science*, vol. 2, pp. 265–278, 11 2016.

[63] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[64] Y. Lin, "Representation learning: a brief overview," 2019.

[65] L. V. Jospin, W. Buntine, F. Boussaid, H. Laga, and M. Bennamoun, "Hands-on bayesian neural networks – a tutorial for deep learning users," 7 2020.

[66] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database."

[67] F. Pedregosa, G. Varoquaux, A. Gramfort, B. M. V., Thirion, O. Grisel, M. Blondel, R. P. P., Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[68] D. Blei, A. Kucukelbir, and J. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, 8 2016.

[69] C. P. Robert, G. Casella, and G. Casella, *Monte Carlo statistical methods*, vol. 2. Springer, 1999.

[70] J. R. Stuart and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall, 2009.

[71] H. Wang and D.-Y. Yeung, "A survey on bayesian deep learning," 2016.

[72] P. Esposito, "Blitz - bayesian layers in torch zoo (a bayesian deep learing library for torch)," 2020.

[73] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[75] W. Falcon, J. Borovec, A. Wälchli, N. Eggert, J. Schock, J. Jordan, N. Skafte, Ir1dXD, V. Bereznyuk, E. Harris, T. Murrell, P. Yu, S. Præsius, T. Addair, J. Zhong, D. Lipin, S. Uchida, S. Bapat, H. Schröter, B. Dayma, A. Karnachev, A. Kulkarni, S. Komatsu, Martin.B, J.-B. SCHIRATTI, H. Mary, D. Byrne, C. Eyzaguirre, cinjon, and A. Bakhtin, "Pytorchlightning/pytorch-lightning: 0.7.6 release," May 2020.