Master's thesis

November 22, 2023

On variational autoencoders: theory and applications

Maksym Sevkovych

Registration number: 3330007

In collaboration with: Duckeneers GmbH

Inspector: Univ.-Prof. Dr. Ingo Steinwart

Recently in the realm of machine learning, the power of generative models has revolutionized the way we perceive data representation and creation. This thesis focuses on the captivating domain of Variational Autoencoders (VAEs), a cutting-edge class of machine learning models that seamlessly combine unsupervised learning and data generation. In the course of this thesis we embark on an expedition through the intricate architecture and mathematical elegance that underlie VAEs.

By dissecting the architecture of VAEs, we show their role as both proficient data compressors and imaginative creators. As we navigate the landscapes of latent spaces and probabilistic encodings, we uncover the essential mechanisms driving their flexibility. Applications of VAEs extend from anomaly detection to image generation. However, we will focus on the latter.

Contents

1 Preliminary

In order to fathom the topic of variational autoencoders or even autoencoders in general, we need to consider a couple of preliminary ideas. Those ideas consist mainly of neural networks and their optimization - usually being called training. In this chapter, we will tackle the conceptional idea of how to formulate neural networks in a mathematical way and furthermore, we will consider a couple of useful operations that neural networks are capable of doing. Then, we will take a look at some strategies of training neural networks. Lastly, we will consider neural networks operating on images. This discipline of machine learning is usually referred to as computer vision.

1.1 Neural networks

The idea of artificial neural networks originated from analysing mammal's brains. An accumulation of nodes - so called neurons, connected in a very special way that fire an electric impulse to adjacent neurons upon being triggered and transmit information that way. Scientists tried to mimic this natural architecture and replicate this mammal intelligence artificially. This research has been going for almost 80 years and became immensely popular recently through artificial intelligences like OpenAI's ChatGPT or Google's Bard for the broad public. But what actually is a neural network? What happens in a neural network? Those are very interesting and important questions that we will find answers for.
As already mentioned, neural networks consist of single neurons that transmit information upon being „triggered". Obviously, triggering an artificial neuron can't happen the same way as neurological neurons are being triggered through stimulus. Hence, we need to model the triggering of a neuron in some way. The idea is to filter information that does not exceed a certain stimulus threshold. This filter is usually being called activation function. Indeed, there are lots of ways of modelling such activation functions and it primarily depends on the specific use-case what exactly the activation function has to fulfil. Therefore, we define activation functions in the most general way possible.
TODO: Give a formal reference to neural networks somewhere?

Definition 1.1.1. A non-constant function $\varphi : \mathbb{R} \to \mathbb{R}$ is called an **activation function** if it is continuous.

Even though there are lots of different activation functions, we want to consider mainly the following ones, see [1, chapter 6].

Example 1.1.2. The following functions are activation functions.

Rectified Linear Unit (ReLU): $\varphi(t) = \max\{0, t\}$,

Leaky Rectified Linear Unit (Leaky ReLU): $\varphi(t) = \begin{cases} \alpha t, & t \leq 0, \\ t, & t > 0. \end{cases}$

Sigmoid: $$\varphi(t) = \frac{1}{1+e^{-t}}.$$

Now, having introduced activation functions we can introduce neurons.

Definition 1.1.3. Let $\varphi : \mathbb{R} \to \mathbb{R}$ be an activation function and $w \in \mathbb{R}^k$, $b \in \mathbb{R}$. Then a function $h : \mathbb{R}^k \to \mathbb{R}$ is called **φ-neuron** with weight w and bias b, if

$$h(x) = \varphi\left(\langle w, x\rangle + b\right), \qquad x \in \mathbb{R}^k. \tag{1.1}$$

We call $\theta \coloneqq (w, b)$ the parameters of the neuron h.

If we arrange multiple neurons next to each other, we can define a layer consisting of neurons. This way we can expand the architecture from one to multiple neurons.

Definition 1.1.4. Let $\varphi : \mathbb{R} \to \mathbb{R}$ be an activation function and $W \in \mathbb{R}^{m\times k}$, $b \in \mathbb{R}^m$. Then a function $H : \mathbb{R}^k \to \mathbb{R}^m$ is called **φ-layer** of width m with **weights** W and **biases** b if for all $i = 1, \dots, m$ the component function h_i of H is a φ-neuron with weight $w_i = W^\top e_i$ and bias $b_i = \langle b, e_i\rangle$, where e_i denotes the standard ONB of $\mathbb{R}^m$.
If we consider $\widehat{\varphi} : \mathbb{R}^k \to \mathbb{R}$ as the component-wise mapping of $\varphi : \mathbb{R} \to \mathbb{R}$, meaning $\widehat{\varphi}(v) = (\varphi(v_1), \dots, \varphi(v_k))$, we can write the φ-layer $H : \mathbb{R}^k \to \mathbb{R}^m$ as

$$H(x) = \widehat{\varphi}(Wx + b), \qquad x \in \mathbb{R}^k. \tag{1.2}$$

In the following, we will denote the weights W and biases b as **parameters of the neural network** $\theta \coloneqq (W, b)$.

Finally, we can introduce neural networks as an arrangement of multiple neural layers.

Definition 1.1.5. Let $L \in \mathbb{N}$, $\varphi_1, \dots, \varphi_L$ be activation functions and $H_1, \dots, H_L$ be φ_i-layers with parameters $\theta_i = (W_i, b_i)$ for all $i \in \{1, \dots, L\}$. Furthermore, let $\theta = (\theta_1, \dots \theta_L)$, $\varphi = (\varphi_1, \dots, \varphi_L)$ and $d_1, \dots, d_{L+1} \in \mathbb{N}$.
Then we define a **φ-deep neural network** of depth L with parameters $\theta \in \Theta$ as

$$\begin{aligned} f_{\varphi,L,\theta} : \mathbb{R}^{d_1} &\to \mathbb{R}^{d_{L+1}} \\ x &\to H_L \circ \dots \circ H_1(x), \qquad x \in \mathbb{R}^{d_1}, \end{aligned} \tag{1.3}$$

where each $H_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_{i+1}}$ is a φ_i-layer. Ultimately, d_1 describes the input dimension and d_{L+1} the output dimension of the neural network.
Lastly, we will write $f \coloneqq f_{\varphi,L,\theta}$, if the activation function φ, the depth L and the parameters θ are clear out of context.

One may realize, that the neural network is defined in a way that the activation function may vary in each layer. However, in most applications the input layer and the hidden layers ($i \in \{1, \dots, L-1\}$) share the same activation function. The last layer, often referred to as output layer, usually has a different activation function. This activation function may as well be the identity function.
A visual representation of a neural network can be found in figure 1.1

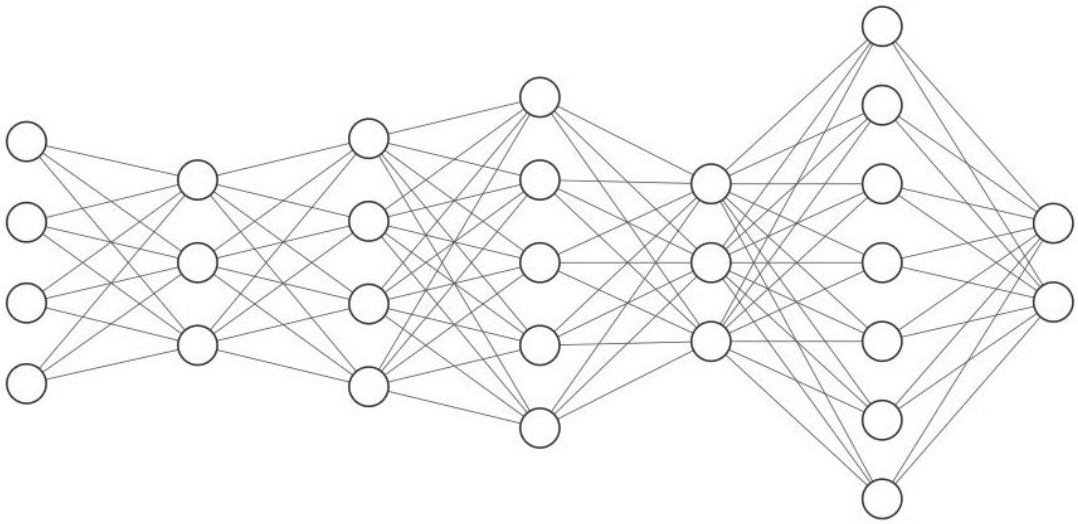


Figure 1.1: A neural network with input $x \in \mathbb{R}^4$ and output $y \in \mathbb{R}^2$. The five hidden layers have dimensions 3, 4, 5, 3 and 7 respectively. The graphic was generated with http://alexlenail.me/NN-SVG/index.html

1.2 Training of neural networks

Since we now know what neural networks are, we want to discuss how to tune them to a specific problem. This procedure is usually referred to as training of a neural network. There are many approaches of how to train a neural network. However, all of them require the definitions of quantities called loss function and risk function. The loss function is a function that measures the point-wise error of a neural network, or any other prediction function in general. This is fundamental in supervised learning. In contrast, the risk function is a function that measures the error with regard to a probability measure or an observed data set.

Definition 1.2.1. Let $d, n \in \mathbb{N}$ and $X \subseteq \mathbb{R}^d$ and $Y \subseteq \mathbb{R}^n$, that we will refer to as input and output space and $t \in \mathbb{R}^n$ be a prediction for $y \in Y$.
Then we define a **supervised loss function** $L : X \times Y \times \mathbb{R}^n \to [0, \infty)$ as a measurable function that compares a true value $y \in Y \subset \mathbb{R}^n$ to a predicted value $\widehat{y} = t$ in a suitable way.

The recently introduced loss function allows us to compare a true label to a prediction. However, since we only require it to be measurable, the function is defined very general. This is solely, because one may be interested in different loss functions in different settings, since the choice of a specific loss functions is a crucial aspect of learning theory, as it directly impacts the behaviour of learning algorithms.
We want to consider a couple of important examples for loss functions, for further reading please take a look at [1, chapter 4.3].

Example 1.2.2. Let $y \in Y \coloneqq \mathbb{R}$ and $t \in \mathbb{R}$, then the following functions are loss functions.

Squared Error Loss: $L(x, y, t) = |y - t|^2$,

Linear Error Loss: $L(x, y, t) = |y - t|$.

Now, let $y \in Y := \mathbb{R}^n$ and $t \in \mathbb{R}^n$ with $n > 1$. Then, we consider the squared error loss and the linear error loss pointwise:

Squared Error Loss: $L(x, y, t) = \sum_{i=1}^{n} |y_i - t_i|^2$,

Linear Error Loss: $L(x, y, t) = \sum_{i=1}^{n} |y_i - t_i|$,

where $y = (y_1, \ldots, y_n)$ and $t = (t_1, \ldots, t_n)$.

However, loss functions only quantify the disparity between one predicted outcome and one actual value. In order to compare the predictions over a wider range of data points, we need to introduce another quantity, the risk function.

Definition 1.2.3. Let X, Y be input and output spaces, L be a loss function and P be a probability measure on $X \times Y$.
Then we define the **risk of the function** f with regard to a loss function L as

$$\mathcal{R}_{L,\mathrm{P}}(f) = \int_{X \times Y} L\left(x, y, f(x)\right) d\,\mathrm{P}(x, y).$$

In the following, we will denote this as the L**-risk of** f.

Considering applications, one usually wants to compute the risk with regard to observed data instead of a probability measure, since it usually is unknown. In this case, the general risk function becomes more tangible, as we see in the following definition.

Definition 1.2.4. Let X, Y be arbitrary input and output spaces, $D = ((x_1, y_1), \ldots, (x_k, y_k))$ be a dataset consisting of $k \in \mathbb{N}$ data points. Furthermore, let L be a loss function and f be an arbitrary prediction function.
Then we define the **empirical risk function** as

$$\mathcal{R}_{L,D}(f) = \frac{1}{k} \sum_{i=1}^{k} L\left(x_i, y_i, f(x_i)\right). \tag{1.4}$$

We will write $\mathcal{R} := \mathcal{R}_{L,D}$ unless unclear in the given context.

With the previous definitions, we can return to considering the training of neural networks. There are many possible techniques and strategies. However, most of them rely on iteratively finding the gradient - the direction of greatest ascent of the risk function, and afterwards performing a step in the found direction. This approach dates way back to the early 19^{th} century and was introduced by Augustin L. Cauchy in the year 1847. For a brief overview please take a look at [2]. This method is immensely popular and thus, can be discovered in numerous literary works. We will mostly draw inspiration form [3, chapter XV]. For in-depth explanation, please take a look at the reference.
Let us consider a (non-linear) real-valued function f, which is defined on a real normed space X. We assume that f is bounded below on X and aim to find an element $x^* \in X$ such that

$$f(x) \geq f(x^*),$$

for all $x \in X$. Hence, x^* minimizing the function f. To solve this problem, one usually constructs a sequence (x_n) that minimizes f, in the sense that

$$\lim_{n\to\infty} f(x_n) = \inf_{x\in X} f(x).$$

In certain cases, one can construct such a sequence such that it convergence to an optimum x^*. If the considered function f is assumed to be continuous, then this element will be a solution to the proposed problem.
To construct such a sequence, we assume that the function f is differentiable. This means, that the derivative

$$\frac{\partial f(x)}{\partial z} = \frac{1}{\|z\|} \lim_{h\to 0^+} \frac{f(x+hz) - f(x)}{h},$$

exists at each point $x \in X$ and for every direction $z \in X$. We call such a function simply **differentiable**. If the function and its derivative are additionally continuous, we call it **continuously differentiable**.
Furthermore, we assume that there exists a direction for which the derivative takes minimum value. This direction is essentially the negative direction of the gradient of f. If we now perform a step towards the recently found direction, we completed an iteration of the so called steepest descent method - or nowadays better known as the gradient descent algorithm. This algorithm we now want to define a little more precisely.
Let X be a normed space and $f : X \to \mathbb{R}$ be a continuously differentiable function. Furthermore, let there be a direction of steepest descent in every $x \in X$ and $x_0 \in X$ an arbitrary (initial) element. Assume that we already found the k-th iterate $x^{(k)}$, then we define the $(k+1)$-th iterate by

$$x^{(k+1)} = x^{(k)} + \gamma_{k+1}\, z_{k+1}, \tag{1.5}$$

where $z^{(k+1)}$ denotes the direction of steepest descent at $x^{(k)}$. The numerical parameter $\gamma^{(k+1)}$ can be found in various ways. However, we propose the following one.
We specify a sequence of positive numbers (γ_k) such that $\gamma_k \to 0$ and $\sum_{k\geq 1} \gamma_k = \infty$. Furthermore, we assume the z_k to be normalized for all $k \in \mathbb{N}$, i.e. $\|z_k\| = 1$. Hence, the descent value at the k-th iteration is γ_k.
Furthermore, we note at this point that we can establish a connection between the problem of minimizing a functional and that of solving a linear functional equation as follows.
Let U be a self-adjoint operator on a Hilbert space $\mathcal{H}$. We assume that the operator U is below bounded by $m > 0$ and above bounded by $M > 0$. Now, lets consider the linear functional equation

$$Ux = y, \tag{1.6}$$

since the inverse operator U^{-1} exists (because the eigenvalue $\lambda = 0$ does not belong to the spectrum, see [3, theorem IX.5.3]), there exists one unique solution to 1.6, for each $y \in \mathcal{H}$. Now, we define the functional

$$F(x) = \langle Ux, x\rangle - (\langle x, y\rangle + \langle y, x\rangle)\,. \tag{1.7}$$

With the help of the functional (1.7) we can formulate the following theorem.

Theorem 1.2.5. *Let $\mathcal{H}$ be a Hilbert space. A solution $x^* \in \mathcal{H}$ of* (1.6) *yields a minimum of* (1.7)*. Conversely, if* (1.7) *attains a minimum at $x' \in \mathcal{H}$, then x' is a solution of* (1.6)*: that is: $x' = x^*$.*

Proof. Since we assumed $x' \in \mathcal{H}$ to be a solution of (1.6), we can express F as

$$F(x) = \langle Ux, x\rangle - \langle x, Ux^*\rangle - \langle Ux^*, x\rangle = \langle U(x - x^*), x - x^*\rangle - \langle Ux^*, x^*\rangle\,. \tag{1.8}$$

Using the boundedness of U we receive

$$F(x) \geq m\,\langle x - x^*, x - x^*\rangle - \langle Ux^*, Ux^*\rangle \geq \langle Ux^*, Ux^*\rangle = F(x^*).$$

That is, $x^* \in \mathcal{H}$ indeed minimizes the functional F.
to prove the second part of the theorem, we will use the same inequality.

$$0 = F(x') - F(x^*)\,\langle U(x' - x^*), x' - x^*\rangle \geq m\,\langle (x' - x^*), x' - x^*\rangle\,,$$

and therefore, $x' = x^*$. □

Furthermore, we realise that the application of the gradient descent algorithm to the functional (1.7) leads to a sequence that converges to x^*, which is a solution to (1.6) as we saw in theorem 1.2.5. Now, lets take a look at how applying the gradient descent algorithm to the functional F would look like, in particular.

Corollary 1.2.6. *Let $\mathcal{H}$ be a Hilbert space and F be a functional as defined in* (1.7)*.*
Then the gradient descent algorithm applied to the functional F with arbitrary initial iterate $x^{(0)} \in \mathcal{H}$ looks like

$$x^{(k)} = x^{(k-1)} - \gamma_{k-1}\, z_{k-1},$$

where $z_k \in \mathcal{H}$ and $\gamma_k > 0$ look like

$$z_k = Ux^{(k-1)} - y \qquad \text{and} \qquad \gamma_k = \frac{\langle z_k, z_k\rangle}{\langle Uz_k, z_k\rangle}.$$

Proof. Let $x, z \in \mathcal{H}$, then

$$\begin{aligned} F(x + z) &= \langle U(x + z), x + z\rangle - (\langle x + z, y\rangle + \langle y, x + z\rangle) \\ &= \langle Ux, x\rangle - (\langle x, y\rangle + \langle y, x\rangle) + (\langle Ux - y, z\rangle + \langle z, Ux - y\rangle) + \langle Uz, z\rangle \\ &= F(x) + (\langle Ux - y, z\rangle + \langle z, Ux - y\rangle) + \langle Uz, z\rangle\,. \end{aligned} \tag{1.9}$$

Considering the derivative in direction z gives us

$$\frac{\partial F}{\partial z}(x) = \frac{1}{\|z\|}\left(\langle Ux - y, z\rangle + \langle z, Ux - y\rangle\right) = \frac{2\langle Ux - y, z\rangle}{\|z\|}.$$

Thus the direction of steepest descent at $x^{(0)} \in \mathcal{H}$ is given by $z_1 = Ux^{(0)} - y$.
Lastly, to determine the descent value we consider the equation

$$\lim_{h \to 0^+} \frac{F(x^{(0)} + hz_1) - F(x^{(0)})}{h} = 0,$$

where with the use of equation (1.9), we have

$$F(x^{(0)} - z_1) = F(x^{(0)}) - 2\langle z_1, z_1\rangle + \langle Uz_1, z_1\rangle$$

what ultimately leads to the step size

$$\gamma_1 = \frac{\langle z_1, z_1\rangle}{\langle Uz_1, z_1\rangle}.$$

In exactly the same way, γ_k and z_k can be determined for all $k > 1$ and hence, all assertions from the theorem are proven. □

The gradient descent algorithm with parameters as asserted in 1.2.6 does converge, as we have seen in the proof. However, one might pose the question how quickly it converges. This question we want to consider in the following theorem.

Theorem 1.2.7. *Let the same assumptions as in corollary 1.2.6 hold. Then the constructed sequence $(x^{(n)})$ with $(z_k), (\gamma_k)$ as in corollary 1.2.6, converges to $x^* \in \mathcal{H}$. Its speed of convergence is given by*

$$\left\|x^{(n)} - x^*\right\| \leq \frac{\|z_1\|}{m}\left(\frac{M-m}{M+m}\right)^n,$$

for all $n \in \mathbb{N}_0$.

Proof. Firstly, we rewrite the equation (1.6) to

$$\begin{aligned} & Ux = y \\ \iff \quad & 0 = ky - kUx \\ \iff \quad & x = x - kUx + ky. \end{aligned} \tag{1.10}$$

We chose the numerical factor $k > 0$ in a way, that the operator $T = I - kU$ has the smallest possible norm. Since we assumed U to be lower bounded by m and upper bounded by M, the operator T needs to be lower bounded by $1 - km$ and upper bounded by $1 - kM$, where the minimal norm will occur if

$$1 - km = -(1 - kM).$$

This leads to

$$k = \frac{2}{m + M}.$$

Therefore, for $\|T\|$ holds

$$\|T\| = 1 - km = 1 - \frac{2m}{m+M}, \tag{1.11}$$

as well as

$$\|T\| = kM - 1 = \frac{2M}{m+M} - 1. \tag{1.12}$$

Combining the equations (1.11) and (1.12) leads to

$$\|T\| = \frac{M-m}{M+m}.$$

Moreover, applying (1.10) and rewriting it gives us

$$x_1' = Tx^{(0)} + ky = x^{(0)} - k(Ux^{(0)} - y) = x^{(0)} - kz_1. \tag{1.13}$$

Let us introduce the operator $V = U^{1/2}$ and plug it into (1.8). Note that since U is a self-adjoint operator, so is V (see [3, theorem V.6.2]).

$$\begin{aligned} F(x) &= \langle U(x-x^*), x-x^*\rangle - \langle Ux^*, x^*\rangle \\ &= \langle V(x-x^*), V(x-x^*)\rangle - \langle Vx^*, Vx^*\rangle \\ &= \|V(x-x^*)\|^2 - \|Vx^*\|^2 \end{aligned} \tag{1.14}$$

Considering the inequality $F(x_1') \geq F(x_1)$, which holds due to the fact that x_1 is the iterate performed with optimal descent value (at least the author proposes? Feels somewhat awkward though), leads to

$$F(x^{(1)}) - F(x^*) \leq F(x_1') - F(x^*),$$

which we can rewrite again with the use of (1.14)

$$\left\|V(x^{(1)} - x^*)\right\| \leq \left\|V(x_1' - x^*)\right\|. \tag{1.15}$$

Since equation (1.8) was the rewritten equation (1.6), we can write with the definition of T

$$x^* = Tx^* + ky,$$

subtracting this equation from (1.13) gives us

$$x_1' - x^* = T(x^{(0)} - x^*),$$

applying the operator V on both sides gives us at last

$$V\left(x_1' - x^*\right) = TV(x^{(0)} - x^*).$$

Considering the norm leads to

$$\begin{aligned} \left\|V\left(x_1' - x^*\right)\right\| &= \left\|TV(x^{(0)} - x^*)\right\| \\ &\leq \|T\| \left\|V(x^{(0)} - x^*)\right\| \\ &= \frac{M-m}{M+m} \left\|V(x^{(0)} - x^*)\right\| \end{aligned}$$

Therefore, with inequality (1.15) we have

$$\left\|V(x^{(1)} - x^*)\right\| \leq \frac{M-m}{M+m} \left\|V(x^{(0)} - x^*)\right\|.$$

If we apply exactly the same arguments for all $k \in \{1, \dots n\}$, then we have the following bound for the n-th iterate

$$\left\|V(x^{(n)} - x^*)\right\| \leq \frac{M-m}{M+m} \left\|V(x^{(n-1)} - x^*)\right\|,$$

which leads to the bound

$$\left\|V(x^{(n)} - x^*)\right\| \leq \left(\frac{M-m}{M+m}\right)^n \left\|V(x^{(0)} - x^*)\right\|.$$

Furthermore, we realise that the function $t^{-1/2}$ is continuous in $[m, M]$ and therefore on the spectrum of U, which leads to the observation that the inverse operator $V^{-1/2}$ exists.
Moreover, with the help of [3, theorem V.6.2] (which regards the spectrum S_U of the operator U) the following equations hold

$$\begin{aligned}\left\|V^{-1}\right\| &= \max_{t \in S_U} \frac{1}{\sqrt{t}} = \frac{1}{\sqrt{m}}, \\ \|V\| &= \max_{t \in S_U} \sqrt{t} = \sqrt{M}.\end{aligned}$$

Combining the above results, we receive

$$\begin{aligned}\left\|x^{(n)} - x^*\right\| = \left\|V^{-1}V\left(x^{(n)} - x^*\right)\right\| &\leq \left\|V^{-1}\right\| \left\|V\left(x^{(n)} - x^*\right)\right\| \\ &\leq \left\|V^{-1}\right\| \left(\frac{M-m}{M+m}\right)^n \left\|V\left(x^{(0)} - x^*\right)\right\| \\ &\leq \frac{1}{\sqrt{m}} \left(\frac{M-m}{M+m}\right)^n \left\|V\left(x^{(0)} - x^*\right)\right\|. \qquad (1.16)\end{aligned}$$

However, since the unknown element x^* appears on the right-hand side, we want to somehow remove it. In order to do this, we consider

$$\left\|V(x^{(0)} - x^*)\right\| = \left\|V^{-1}U(x^{(0)} - x^*)\right\| \leq \left\|V^{-1}\right\| \left\|U(x^{(0)} - x^*)\right\| = \frac{\|z_1\|}{\sqrt{m}}.$$

Plugging this result into (1.16), gives us

$$\left\|x^{(n)} - x^*\right\| \leq \frac{\|z_1\|}{m} \left(\frac{M-m}{M+m}\right)^n.$$

This is exactly the inequality asserted in the theorem. □

TODO: Here we should show how to compute the gradient of a risk, where a neural network is considered as a prediction function. This would show that in applications it is very expensive to compute the gradient with regard to the whole dataset, which is the motivation for SGD.

However, the gradient descent algorithm relies on computing the gradient of the risk function in each iteration, which oftentimes is too expensive and therefore is being relaxed by only

considering the gradient in one sample (or multiple samples -¿ mini-batch). T NOTE: A very promising reference seems to be chapter 4.2 (p.39) in Saad David - On-Line Learning in Neural Networks. Furthermore, I should mention 13.3.2 in Sra, Nowozin, Wright as further reference!

However, in order to apply the gradient descent algorithm to train neural networks, we have to consider how to actually compute the gradient of the empirical risk function, where we consider a neural network as prediction function.

Lemma 1.2.8. *Let $f_\theta : \mathbb{R}^d \to \mathbb{R}$ be a neural network with parameters $\theta \in \Theta$, arbitrary depth $L_n \in \mathbb{N}$ and arbitrary activation function φ. Furthermore, let D be a dataset of length $k \in \mathbb{N}$ and L be an arbitrary loss function.*
The gradient of the risk function $\mathcal{R}(\cdot)$ with regard to the neural network f_θ and thus the parameters θ look as follows

$$\partial_\theta \mathcal{R}\left(f_\theta\right) = \frac{1}{k} \sum_{i=1}^{k} \partial_\theta L\left(x_i, y_i, f_\theta(x_i)\right).$$

Hence, it is the average of gradients in all data points $(x_i, y_i) \in D$.

Proof. To prove the assertion we simply use the definition 1.2.4 of the empirical risk function and consider the linearity property of derivatives.

$$\begin{aligned}\partial_\theta \mathcal{R}\left(f_\theta\right) &= \partial_\theta \frac{1}{k} \sum_{i=1}^{k} L\left(x_i, y_i, f_\theta(x_i)\right) \\ &= \frac{1}{k} \sum_{i=1}^{k} \partial_\theta L\left(x_i, y_i, f_\theta(x_i)\right).\end{aligned}$$

□

With the previous definitions and results we can formulate the gradient descent algorithm for a neural network.

Corollary 1.2.9. *Let $f_\theta : \mathbb{R}^d \to \mathbb{R}$ be a neural network with parameters $\theta \in \Theta$, arbitrary depth $L \in \mathbb{N}$ and arbitrary activation function φ. Let $(\gamma_t)_{t\in\mathbb{N}}$ be a sequence with $\gamma_t \to 0$ and D be a dataset of length $k \in \mathbb{N}$.*
Then one can train the neural network f_θ with the gradient descent algorithm proposed in (1.5). In this setting, the algorithm looks as follows

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_{t-1}\, \partial_\theta \mathcal{R}\left(f_{\theta^{(t-1)}}\right),$$

where the gradient can be computed as in lemma 1.2.8

TODO: Name some properties (convergence, rate, etc.) and reference them

This is a valuable result, since this way one can iteratively optimize any convex function. Such iterative methods are powerful in numerical settings, where one could use a machine to compute the result. However, there is one problem: in many practical cases it is way to expensive to compute the gradient with regard to the whole dataset, if the dataset becomes significantly large. This lead to a bunch of approaches on how to make this algorithm more efficient, one of those being the stochastic gradient descent algorithm.

Theorem 1.2.10. *Let $f_\theta : \mathbb{R}^d \to \mathbb{R}$ be a neural network with parameters $\theta \in \Theta$, arbitrary depth $L \in \mathbb{N}$ and arbitrary activation function φ. Let $(\gamma_t)_{t\in\mathbb{N}}$ be a sequence with $\gamma_t \to 0$ and D be a dataset of length $k \in \mathbb{N}$.*
*Then we define the t-th iterate of the **stochastic gradient descent algorithm** by*

$$\theta^{(t)} = \theta^{(t-1)} - \gamma_{t-1}\, \partial_{\theta,i}\, \mathcal{R}\left(f_{\theta^{(t-1)}}\right), \tag{1.17}$$

with $i \in \{1, \dots, k\}$ and $\partial_{\theta,i}\, \mathcal{R}\left(f_{\theta^{(t)}}\right)$ denoting the gradient with regard to the i-th data tuple $(x_i, y_i) \in D$.

TODO: Name some properties (convergence, rate, etc.) and reference them

Lastly, we want to consider another powerful optimization algorithm that adapts learning rates based on past gradient magnitudes and momenta - it is called „Adaptive Moment Estimation (ADAM)“.
However, in order to formulate the algorithm formally we need to introduce stochastic moments first. We do so analogously to [5, chapter 5]

Definition 1.2.11. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space, X be a random variable over Ω and $k \in \mathbb{N}$. Then we define the **k-th moment** of X as

$$m_k \coloneqq \mathbb{E}\left[X^k\right] = \int_{-\infty}^{+\infty} x^k f(x) dx.$$

Furthermore, we define the **k-th central moment** of X as

$$\mu_k \coloneqq \mathbb{E}\left[(X-\mu)^k\right] = \int_{-\infty}^{+\infty} (x-\mu)^k f(x) dx,$$

where we denote $\mu = \mathbb{E}[X]$.

Remark 1.2.12. Let the same assumptions as in definition 1.2.11 hold. Usually, the first moment is referred to as **mean** and the second central moment as **variance** of the random variable X.

Now, we can formulate the proposed algorithm itself. For further reading please take a look at [6] or [1, chapter 8].

Algorithm 1 ADAM optimizer

Let $g_t := \nabla_\theta f_t(\theta)$ denote the gradient, i.e. the vector of partial derivatives of f_t w.r.t. θ evaluated at time step t. Furthermore, let $g_t^2 := g_t \odot g_t$ denote the element-wise square of g_t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector

$m_0, v_0 \leftarrow 0$ (Initialize 1^{st} and 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize time step)
while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at time step t,
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ▷ Update biased first moment estimate,
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ ▷ Update biased second moment estimate,
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Compute bias-corrected first moment estimate,
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Compute bias-corrected second moment estimate,
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ ▷ Update parameters with bias- corrected moments,
end while
return θ_t ▷ Return Resulting parameters.

However, algorithm 1 was later proven to be not converging in certain settings, see [7]. The authors proposed another approach to the optimization problem, where they first introduced a general formulation of the algorithm, see 2. Afterwards, they proposed an alternative approach called the AMSGrad algorithm. In contrast to ADAM, AMSGrad uses the maximum of past squared gradients rather than the exponential average to update the parameters. This way the authors were able to fix the issues of the original algorithm ADAM.
But in order to introduce AMSGrad formally, we need to define some quantities first.

Definition 1.2.13. Let $\mathcal{F} \subset \mathbb{R}^d$ be a set of points. We say that $\mathcal{F}$ has **bounded diameter** $D_\infty < \infty$ if for all x, y in $\mathcal{F}$ holds

$$\|x, y\|_\infty < D_\infty.$$

Definition 1.2.14. Let $y \in \mathbb{R}^d$, $\mathcal{F} \subset \mathbb{R}^d$ with bounded diameter D_∞ and $X : \mathbb{R}^d \to \mathbb{R}^d$ an arbitrary operator. Then we define the **X-projection of y onto $\mathcal{F}$** as

$$\Pi_{\mathcal{F},X}(y) = \min_{x \in \mathcal{F}} \left\| X^{1/2}(x - y) \right\|.$$

If the operator X is the identity $\mathbb{1}$, we reduce the notation to $\Pi_{\mathcal{F}} := \Pi_{\mathcal{F},\mathbb{1}}$.

Now, we introduce a short technical assumption concerning the recently introduced projection.

Lemma 1.2.15. *Let $\mathcal{F} \subset \mathbb{R}^d$ be a set of points and $\Pi_{\mathcal{F},X}$ be an X-projection with operator X. Then the following assertion holds for all $x^* \in \mathcal{F}$.*

$$\Pi_{\mathcal{F},X}(x^*) = x^*.$$

Proof. Let's first consider the definition of the projection $\Pi_{\mathcal{F}}$.

$$\Pi_{\mathcal{F},X}\left(x^*\right) = \min_{x\in\mathcal{F}} \left\|X^{1/2}(x - x^*)\right\|.$$

Hence, for the point x' that minimizes the right hand side holds

$$x' = \arg\min_{x\in\mathcal{F}} \left\|X^{1/2}(x - x^*)\right\|,$$

what can be considered equally as

$$\iff \qquad x' = \arg\min_{x\in\mathcal{F}} \left(x - x^*\right).$$

But since we assumed that $x^* \in \mathcal{F}$, it follows that $x' = x^*$ and the assertion holds. □

Another quantity we need to introduce is the so called regret of an algorithm. It essentially quantifies how much an algorithm would have performed better if it had known the best action in advance. In other words, the regret quantifies the cost of not making the optimal decisions at each step. This is a common approach in online learning settings.

Definition 1.2.16. Let $T \in \mathbb{N}$, $\mathcal{F} \subset \mathbb{R}^d$ be a set of points and $f_t(\theta_t)$ a stochastic objection function with parameters $\theta_t \in \mathcal{F}$ at time step t. Then we define the **regret** as the function

$$R_T = \sum_{t=1}^{T} f_t(\theta_t) - \min_{\theta\in\mathcal{F}} \sum_{t=1}^{T} f_t(\theta).$$

Algorithm 2 Generic Adaptive Method Setup

Let g_t as in algorithm 1 and let $T \in \mathbb{N}$. Furthermore, let Θ be a parameter space.

Require: $\{\alpha_t\}_{t=1}^{T}$: Stepsizes
Require: $\{\phi_t, \psi_t\}_{t=1}^{T}$: Sequence of functions
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: $\theta_1 \in \Theta$: Initial parameter vector

1: **for** $t = 1, \ldots, T$ **do**
2: $\quad g_t \leftarrow \nabla_\theta f_t(\theta_t)$ ▷ Get gradients w.r.t. stochastic objective at time step t,
3: $\quad m_t \leftarrow \phi_t\left(g_1, \ldots, g_t\right)$ ▷ Update biased first moment estimate,
4: $\quad V_t \leftarrow \psi_t\left(g_1, \ldots, g_t\right)$ ▷ Update biased second moment estimate,
5: $\quad \widehat{\theta}_{t+1} \leftarrow \theta_t - \alpha_t m_t / \sqrt{V_t}$ ▷ Compute biased updated parameters,
6: $\quad \theta_{t+1} \leftarrow \Pi_{\Theta,\sqrt{V_t}}\left(\widehat{\theta}_{t+1}\right)$ ▷ Unbias updated parameters,
7: **end for**
8: **return** θ_t ▷ Return resulting parameters.

We realize that upon defining ϕ_t and ψ_t in algorithm 2 in a suitable way, we can obtain various familiar algorithms. We want to consider them in the following example.

Example 1.2.17. Let the same assumptions as in algorithm 2 hold.

1. Let $(\phi_t)_t$ and $(\psi_t)_t$ be defined as

$$\phi_t(g_1, \dots, g_t) = g_t,$$
$$\psi_t(g_1, \dots, g_t) = \mathbb{1}.$$

Then the resulting update rule looks like

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t g_t,$$

which is exactly the SGD algorithm as proposed in theorem 1.2.10.

2. Let ϕ_t and ψ_t be defined as

$$\phi_t(g_1, \dots, g_t) = (1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i,$$
$$\psi_t(g_1, \dots, g_t) = (1-\beta_2) \operatorname{diag}\left(\sum_{i=1}^{t} \beta_2^{t-i} g_t^2 \right).$$

Then the resulting update rule looks like

$$m_t \leftarrow (1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i,$$
$$v_t \leftarrow (1-\beta_2) \operatorname{diag}\left(\sum_{i=1}^{t} \beta_2^{t-i} g_t^2 \right),$$

which is the same as in algorithm 1 (without the bias-correction, but the argument still holds, see [7]). Furthermore, the X-projection is obsolete for the choice of $X = \mathbb{1}$ and $\mathcal{F} = \mathbb{R}^d$. This follows directly from lemma 1.2.15.

With the help of the general algorithm we proposed in algorithm 2, we can introduce an algorithm, which can be proven to converge in a non-convex setting. Since neural networks ultimately are non-convex functions, this is exactly what we are interested in.

Algorithm 3 AMSGrad Optimizer

Let the same assumptions as in algorithm 2 hold.

Require: $\{\alpha_t\}_{t=1}^T$: Stepsizes
Require: $\{\beta_{1t}\}_{t=1}^T, \beta_2$, with $\beta_{1t}, \beta_2 \in [0, 1)$: Decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: $\theta_1 \in \Theta$: Initial parameter vector

$m_0, v_0 \leftarrow 0$ (Initialise 1st and 2nd moment vector)
for $t = 1, \dots, T$ **do**
  $g_t \leftarrow \nabla_\theta f_t(\theta_t)$ ▷ Get gradients w.r.t. stochastic objective at time step t,
  $m_t \leftarrow \beta_{1t} m_{t-1} + (1-\beta_{1t}) g_t$ ▷ Update biased first moment estimate,
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ ▷ Update biased second moment estimate,
  $\widehat{v}_t \leftarrow \max\{\widehat{v}_{t-1}, v_t\}$ ▷ Compute bias-corrected first moment estimate,
  $\widehat{V}_t \leftarrow \operatorname{diag}(\widehat{v}_t)$ ▷ Compute bias-corrected second moment estimate,
  $\theta_{t+1} \leftarrow \Pi_{\Theta, \sqrt{\widehat{V}_t}} \left(\theta_t - \alpha_t m_t / \sqrt{\widehat{v}_t} \right)$ ▷ Update parameters,
end for
return θ_{t+1} ▷ Return resulting parameters.

The AMSGrad optimizer, defined in algorithm 3 does indeed converge, as proven in [7, theorem 4]. We take a quick look at the theorem and its proof. However, for a deeper understanding of the alternative algorithms, please refer to [7], since this would go beyond the scope of this thesis' topic.
First, we need an auxiliary lemma ans cite it from [7, lemma 4], but since their proof does not align with the proof of the original paper [8, lemma 3] we will adjust the proof to the original one.

Lemma 1.2.18. *Let $\mathcal{S}_+^{\mathrm{d}}$ denote the set of all positive definite $d \times d$-matrices and let $Q \in \mathcal{S}_+^{\mathrm{d}}$. Furthermore, let $\mathcal{F} \subset \mathbb{R}^d$ be a feasible convex set.*
Suppose $z_1, z_2 \in \mathbb{R}^d$ and $u_1 = \min_{x \in \mathcal{F}} \|Q^{1/2}(x - z_1)\|$ as well as $u_2 = \min_{x \in \mathcal{F}} \|Q^{1/2}(x - z_2)\|$. Then the following inequality holds

$$\left\|Q^{1/2}\left(u_1 - u_2\right)\right\| \leq \left\|Q^{1/2}\left(z_1 - z_2\right)\right\|.$$

Proof. We begin with defining

$$B(u, z) \coloneqq \frac{1}{2}\left\|Q^{1/2}\left(u - z\right)\right\|^2 = \frac{1}{2}\left(u - z\right)^{\mathsf{T}} Q\left(u - z\right). \tag{1.18}$$

Hence, we can write

$$u_1 = \arg\min_{x \in \mathcal{F}} B(x, z_1). \tag{1.19}$$

If we now consider the gradient of (1.18), we receive

$$\nabla_x B(x, z_1) = \nabla_x \frac{1}{2}\left(\left(x - z_1\right)^{\mathsf{T}} Q\left(x - z_1\right)\right) = Q\left(x - z_1\right).$$

Therefore, it holds that

$$Q\left(u_1 - z_1\right)^{\mathsf{T}}\left(u_2 - u_1\right) \geq 0,$$

otherwise for δ sufficiently small it would mean that $u_1 + \delta\left(u_2 - u_1\right) \in \mathcal{F}$, due to the convexity of $\mathcal{F}$, and therefore would be closer to z_1 than u_1. This contradicts the assumption, that u_1 is the projection, i.e. fulfils equation (1.2).
With the exact same argument it holds that

$$Q\left(u_2 - z_2\right)^{\mathsf{T}}\left(u_1 - u_2\right) \geq 0,$$

If we combine those two inequalities, we receive

$$\begin{aligned} & Q\left(u_1 - z_1\right)^{\mathsf{T}}\left(u_2 - u_1\right) + Q\left(u_2 - z_2\right)^{\mathsf{T}}\left(u_1 - u_2\right) \geq 0 \\ \iff \quad & Q\left(u_1 - z_1\right)^{\mathsf{T}}\left(u_2 - u_1\right) - Q\left(u_2 - z_2\right)^{\mathsf{T}}\left(u_2 - u_1\right) \geq 0. \end{aligned}$$

These inequalities are due to $Q \in \mathcal{S}_+^{\mathrm{d}}$ equivalent to

$$\begin{aligned} & \left(u_1 - z_1\right)^{\mathsf{T}} Q\left(u_2 - u_1\right) - \left(u_2 - z_2\right)^{\mathsf{T}} Q\left(u_2 - u_1\right) \geq 0, \\ \iff \quad & \left(z_2 - z_1\right)^{\mathsf{T}} Q\left(u_2 - u_1\right) - \left(u_2 - u_1\right)^{\mathsf{T}} Q\left(u_2 - u_1\right) \geq 0, \\ \iff \quad & \left(z_2 - z_1\right)^{\mathsf{T}} Q\left(u_2 - u_1\right) \geq \left(u_2 - u_1\right)^{\mathsf{T}} Q\left(u_2 - u_1\right). \end{aligned}$$

For readability reasons we now define $\widehat{u} \coloneqq (u_2 - u_1)$ and $\widehat{z} \coloneqq (z_2 - z_1)$. Therefore, we receive

$$\widehat{z}^{\mathsf{T}} Q \widehat{u} \geq \widehat{u}^{\mathsf{T}} Q \widehat{u}.$$

Moreover, since $Q \in \mathcal{S}_+^{\mathrm{d}}$ it holds that

$$\begin{aligned} & (\widehat{z} - \widehat{u})^{\mathsf{T}} Q (\widehat{z} - \widehat{u}) \geq 0 \\ \iff \qquad & \widehat{z}^{\mathsf{T}} Q \widehat{z} - 2\widehat{u}^{\mathsf{T}} Q \widehat{z} + \widehat{u}^{\mathsf{T}} Q \widehat{u} \geq 0. \end{aligned}$$

Thus,

$$\begin{aligned} \widehat{z}^{\mathsf{T}} Q \widehat{z} &\geq 2\widehat{u}^{\mathsf{T}} Q \widehat{z} - \widehat{u}^{\mathsf{T}} Q \widehat{u} \\ &\geq 2\widehat{u}^{\mathsf{T}} Q \widehat{u} - \widehat{u}^{\mathsf{T}} Q \widehat{u} = \widehat{u}^{\mathsf{T}} Q \widehat{u}. \end{aligned}$$

Considering the previous definitions, we achieved

$$\left\| Q^{1/2} (u_1 - u_2) \right\|^2 = \widehat{u}^{\mathsf{T}} Q \widehat{u} \leq \widehat{z}^{\mathsf{T}} Q \widehat{z} = \left\| Q^{1/2} (z_1 - z_2) \right\|.$$

Taking the square root of both sides leads to the assertion. □

Theorem 1.2.19. *Let $(\theta_t)_{t \in \mathbb{N}}$ and $(v_t)_{t \in \mathbb{N}}$ be sequences as in algorithm 3, $\alpha_t = \alpha/\sqrt{t}$ with $\alpha > 0$ and $\beta_1 = \beta_{11}$, $\beta_{1t} \leq \beta_1$ for all $t \in \{1, \dots, T\}$ and $\gamma = \beta_1/\sqrt{\beta_2} < 1$. Assume that Θ has bounded diameter D_∞ and $\|\nabla f_t(\theta)\|_\infty \leq G_\infty$ for all $\theta \in \Theta$ and $t \in \{1, \dots, T\}$.*
Then for θ_t generated using the AMSGrad algorithm (algorithm 3) the following bound for the regret holds

$$R_T \leq \frac{D_\infty^2 \sqrt{T}}{\alpha(1-\beta_1)} \sum_{i=1}^{d} \widehat{v}_{T,i}^{1/2} + \frac{D_\infty^2}{(1-\beta_1)^2} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{\beta_{1t} \widehat{v}_{t,i}^{1/2}}{\alpha_t} + \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)^2(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \|g_{1:T,i}\|_2,$$

where we denote for readability reasons $g_{1:t} \coloneqq (g_1, \dots, g_t)$ and with $g_{j,i}$ and with $v_{j,i}$ we denote the i-th component of g_j and v_j, respectively.

Proof. First, we observe with the definition of the projection

$$\theta_{t+1} = \Pi_{\Theta, \sqrt{\widehat{V}_t}} \left(\theta_t - \alpha_t \widehat{V}_t^{-1/2} m_t \right) = \min_{\theta \in \Theta} \left\| \widehat{V}_t^{1/4} \left(\theta - \left(\theta_t - \alpha_t \widehat{V}_t^{-1/2} m_t \right) \right) \right\|.$$

Furthermore, with lemma 1.2.15 it follows, that $\Pi_{\Theta, \sqrt{\widehat{V}_t}}(\theta^*) = \theta^*$ for all $\theta^* \in \Theta$.
Now, using lemma 1.2.18 with $u_1 = \theta_{t+1}$, $u_2 = \theta^*$ and $Q = \widehat{V}_t^{1/4}$ gives us

$$\begin{aligned} \left\| \widehat{V}_t^{1/4} (\theta_{t+1} - \theta^*) \right\|^2 &\leq \left\| \widehat{V}_t^{1/4} \left(\theta_t - \alpha_t \widehat{V}_t^{-1/2} m_t - \theta^* \right) \right\|^2 \\ &= \left\| \widehat{V}_t^{1/4} (\theta_t - \theta^*) \right\|^2 + \alpha_t^2 \left\| \widehat{V}_t^{-1/4} m_t \right\|^2 - 2\alpha_t \langle m_t, \theta_t - \theta^* \rangle \\ &= \left\| \widehat{V}_t^{1/4} (\theta_t - \theta^*) \right\|^2 + \alpha_t^2 \left\| \widehat{V}_t^{-1/4} m_t \right\|^2 - 2\alpha_t \langle \beta_{1t} m_{t-1} + (1 - \beta_{1t}) g_t, \theta_t - \theta^* \rangle. \end{aligned}$$

If we now rearrange the last inequality, we receive

$$\begin{aligned}
&\left\|\widehat{V}_t^{1/4}\left(\theta_{t+1}-\theta^*\right)\right\|^2-\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2-\alpha_t^2\left\|\widehat{V}_t^{-1/4}m_t\right\|^2\\
&\quad\leq -2\alpha_t\left\langle\beta_{1t}m_{t-1}+\left(1-\beta_{1t}\right)g_t,\theta_t-\theta^*\right\rangle,\\
\iff\quad&\left\|\widehat{V}_t^{1/4}\left(\theta_{t+1}-\theta^*\right)\right\|^2-\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2-\alpha_t^2\left\|\widehat{V}_t^{-1/4}m_t\right\|^2\\
&\quad\leq -2\alpha_t\beta_{1t}\left\langle m_{t-1},\theta_t-\theta^*\right\rangle-2\alpha_t\left(1-\beta_{1t}\right)\left\langle g_t,\theta_t-\theta^*\right\rangle,\\
\iff\quad&\left\|\widehat{V}_t^{1/4}\left(\theta_{t+1}-\theta^*\right)\right\|^2-\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2-\alpha_t^2\left\|\widehat{V}_t^{-1/4}m_t\right\|^2\\
&\quad+2\alpha_t\beta_{1t}\left\langle m_{t-1},\theta_t-\theta^*\right\rangle\leq -2\alpha_t\left(1-\beta_{1t}\right)\left\langle g_t,\theta_t-\theta^*\right\rangle,\\
\iff\quad&-\frac{1}{2\alpha_t\left(1-\beta_{1t}\right)}\left[\left\|\widehat{V}_t^{1/4}\left(\theta_{t+1}-\theta^*\right)\right\|^2-\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2-\alpha_t^2\left\|\widehat{V}_t^{-1/4}m_t\right\|^2\right.\\
&\quad\left.+2\alpha_t\beta_{1t}\left\langle m_{t-1},\theta_t-\theta^*\right\rangle\right]\geq\left\langle g_t,\theta_t-\theta^*\right\rangle,\\
\iff\quad&\frac{1}{2\alpha_t\left(1-\beta_{1t}\right)}\left[\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2-\left\|\widehat{V}_t^{1/4}\left(\theta_{t+1}-\theta^*\right)\right\|^2\right]+\frac{\alpha_t}{2\left(1-\beta_{1t}\right)}\left\|\widehat{V}_t^{-1/4}m_t\right\|^2\\
&\quad-\frac{\beta_{1t}}{1-\beta_{1t}}\left\langle m_{t-1},\theta_t-\theta^*\right\rangle\geq\left\langle g_t,\theta_t-\theta^*\right\rangle,
\end{aligned}$$

And if we now apply the Cauchy-Schwarz inequality and the Young's inequality, this leads to

$$\begin{aligned}
\iff\quad&\left\langle g_t,\theta_t-\theta^*\right\rangle\leq\frac{1}{2\alpha_t\left(1-\beta_{1t}\right)}\left[\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2-\left\|\widehat{V}_t^{1/4}\left(\theta_{t+1}-\theta^*\right)\right\|^2\right]\\
&\quad+\frac{\alpha_t}{2\left(1-\beta_{1t}\right)}\left\|\widehat{V}_t^{-1/4}m_t\right\|^2+\frac{\beta_{1t}}{2\left(1-\beta_{1t}\right)}\alpha_t\left\|\widehat{V}_t^{-1/4}m_{t-1}\right\|^2\\
&\quad+\frac{\beta_{1t}}{2\alpha_t\left(1-\beta_{1t}\right)}\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2.
\end{aligned}\tag{1.20}$$

The next step is a common approach in bounding the regret, where we will use the convexity of the function f_t in each step. is it convex? I think so, because f_t is NOT the neural net here. f_t is defined as „A loss function f_t (to be interpreted as the loss of the model with the chosen parameters in the next minibatch) is then revealed, and the algorithm incurs loss $f_t(x_t)$".

$$\begin{aligned}
&\sum_{t=1}^{T}f_t(\theta_t)-f_t(\theta^*)\leq\sum_{t=1}^{T}\left\langle g_t,\theta_t-\theta^*\right\rangle,\\
&\leq\sum_{t=1}^{T}\left[\frac{1}{2\alpha_t\left(1-\beta_{1t}\right)}\left[\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2-\left\|\widehat{V}_t^{1/4}\left(\theta_{t+1}-\theta^*\right)\right\|^2\right]\right.\\
&\quad+\frac{\alpha_t}{2\left(1-\beta_{1t}\right)}\left\|\widehat{V}_t^{-1/4}m_t\right\|^2+\frac{\beta_{1t}}{2\left(1-\beta_{1t}\right)}\alpha_t\left\|\widehat{V}_t^{-1/4}m_{t-1}\right\|^2\\
&\quad\left.+\frac{\beta_{1t}}{2\alpha_t\left(1-\beta_{1t}\right)}\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2\right],
\end{aligned}$$

where we used inequality (1.20) in the first step.

If we now consider the following inequality for all $t = 1, \dots, T$

$$\begin{aligned}
&\frac{\alpha_t}{2\left(1-\beta_{1t}\right)}\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 + \frac{\beta_{1t}}{2\left(1-\beta_{1t}\right)}\alpha_t\left\|\widehat{V}_t^{-1/4} m_{t-1}\right\|^2 \\
&= \frac{\alpha_t}{2\left(1-\beta_{1t}\right)}\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 + \frac{\beta_{1t}}{2\left(1-\beta_{1t}\right)}\alpha_t\left\|\widehat{V}_t^{-1/4}\left(\frac{m_t-(1-\beta_{1t})g_t}{\beta_{1t}}\right)\right\|^2, \\
&\leq \frac{\alpha_t}{2\left(1-\beta_{1t}\right)}\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 + \frac{1}{2\left(1-\beta_{1t}\right)}\alpha_t\left\|\widehat{V}_t^{-1/4} m_t\right\|^2, \\
&\leq \frac{\alpha_t}{1-\beta_1}\left\|\widehat{V}_t^{-1/4} m_t\right\|^2,
\end{aligned}$$

then this leads ultimately to

$$\begin{aligned}
\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \leq \sum_{t=1}^{T}\Bigg[&\frac{1}{2\alpha_t\left(1-\beta_{1t}\right)}\left[\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2 - \left\|\widehat{V}_t^{1/4}\left(\theta_{t+1}-\theta^*\right)\right\|^2\right] \\
&+ \frac{\alpha_t}{1-\beta_1}\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 + \frac{\beta_{1t}}{2\alpha_t\left(1-\beta_{1t}\right)}\left\|\widehat{V}_t^{1/4}\left(\theta_t-\theta^*\right)\right\|^2\Bigg]. \qquad (1.21)
\end{aligned}$$

We now proceed by bounding the second term, separately.
In order to do so, we first use the definition of $\widehat{v}_T$, which is the maximum of all v_t until the current time step T. This gives us

$$\begin{aligned}
\sum_{t=1}^{T}\alpha_t\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 &= \sum_{t=1}^{T-1}\alpha_t\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 + \alpha_T\sum_{i=1}^{d}\frac{m_{T,i}^2}{\sqrt{\widehat{v}_{T,i}}}, \\
&\leq \sum_{t=1}^{T-1}\alpha_t\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 + \alpha_T\sum_{i=1}^{d}\frac{m_{T,i}^2}{\sqrt{v_{T,i}}}, \\
&\leq \sum_{t=1}^{T-1}\alpha_t\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 + \alpha\sum_{i=1}^{d}\frac{(\sum_{t=1}^{T}(1-\beta_{1t})\prod_{k=1}^{T-t}\beta_{1(T-k+1)}g_{t,i})^2}{\sqrt{T((1-\beta_2)\sum_{t=1}^{T}\beta_2^{T-t}g_{t,i}^2)}},
\end{aligned}$$

where the last inequality follows from the update rules of m_t and v_t in algorithm 3, respectively.
If we now apply the Cauchy-Schwarz inequality, then we receive

$$\begin{aligned}
\sum_{t=1}^{T}\alpha_t\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 \leq \sum_{t=1}^{T-1}\alpha_t\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 \\
+ \alpha\sum_{i=1}^{d}\frac{(\sum_{t=1}^{T}\prod_{k=1}^{T-t}\beta_{1(T-k+1)})(\sum_{t=1}^{T}\prod_{k=1}^{T-t}\beta_{1(T-k+1)}g_{t,i}^2)}{\sqrt{T((1-\beta_2)\sum_{t=1}^{T}\beta_2^{T-t}g_{t,i}^2)}}.
\end{aligned}$$

Now, considering the fact that $\beta_{1t} \leq \beta_1$ for all $t = 1, \dots, T$, gives us

$$\sum_{t=1}^{T}\alpha_t\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 \leq \sum_{t=1}^{T-1}\alpha_t\left\|\widehat{V}_t^{-1/4} m_t\right\|^2 + \alpha\sum_{i=1}^{d}\frac{(\sum_{t=1}^{T}\beta_1^{T-t})(\sum_{t=1}^{T}\beta_1^{T-t}g_{t,i}^2)}{\sqrt{T((1-\beta_2)\sum_{t=1}^{T}\beta_2^{T-t}g_{t,i}^2)}}.$$

The next two steps are considering the inequality $\sum_{t=1}^{T} \beta_1^{T-t} \leq 1/(1-\beta_1)$ and afterwards using the linearity of the square root. This gives us

$$\begin{aligned}
\sum_{t=1}^{T} \alpha_t \left\| \widehat{V}_t^{-1/4} m_t \right\|^2 &\leq \sum_{t=1}^{T-1} \alpha_t \left\| \widehat{V}_t^{-1/4} m_t \right\|^2 + \frac{\alpha}{1-\beta_1} \sum_{i=1}^{d} \frac{\sum_{t=1}^{T} \beta_1^{T-t} g_{t,i}^2}{\sqrt{T((1-\beta_2)\sum_{t=1}^{T} \beta_2^{T-t} g_{t,i}^2)}}, \\
&\leq \sum_{t=1}^{T-1} \alpha_t \left\| \widehat{V}_t^{-1/4} m_t \right\|^2 + \frac{\alpha}{(1-\beta_1)\sqrt{T(1-\beta_2)}} \sum_{i=1}^{d} \sum_{t=1}^{T} \frac{\beta_1^{T-t} g_{t,i}^2}{\sqrt{\beta_2^{T-t} g_{t,i}^2)}}, \\
&\leq \sum_{t=1}^{T-1} \alpha_t \left\| \widehat{V}_t^{-1/4} m_t \right\|^2 + \frac{\alpha}{(1-\beta_1)\sqrt{T(1-\beta_2)}} \sum_{i=1}^{d} \sum_{t=1}^{T} \gamma^{T-t} \left| g_{t,i} \right|,
\end{aligned}$$

where we used the definition of $\gamma = \beta_1/\sqrt{\beta_2}$ in the last step.
If we consider similar upper bounds for all time steps $t = 1, \ldots, T-1$ as we did for the last time step, we receive

$$\begin{aligned}
\sum_{t=1}^{T} \alpha_t \left\| \widehat{V}_t^{-1/4} m_t \right\|^2 &\leq \sum_{t=1}^{T} \frac{\alpha}{(1-\beta_1)\sqrt{t(1-\beta_2)}} \sum_{i=1}^{d} \sum_{j=1}^{t} \gamma^{t-j} \left| g_{j,i} \right|, \\
&= \frac{\alpha}{(1-\beta_1)\sqrt{(1-\beta_2)}} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{1}{\sqrt{t}} \sum_{j=1}^{t} \gamma^{t-j} \left| g_{j,i} \right|, \\
&= \frac{\alpha}{(1-\beta_1)\sqrt{(1-\beta_2)}} \sum_{t=1}^{T} \sum_{i=1}^{d} \left| g_{t,i} \right| \sum_{j=t}^{T} \frac{\gamma^{j-t}}{\sqrt{j}}, \\
&\leq \frac{\alpha}{(1-\beta_1)\sqrt{(1-\beta_2)}} \sum_{t=1}^{T} \sum_{i=1}^{d} \left| g_{t,i} \right| \sum_{j=t}^{T} \frac{\gamma^{j-t}}{\sqrt{t}}.
\end{aligned}$$

Since $\gamma < 1$, we can apply the fact that for geometric series $\sum_{k \geq 1} q^k$, where $q \in (0,1)$ holds $\sum_{k \geq 1} q^k = 1/(1-q)$. This gives us

$$\begin{aligned}
\sum_{t=1}^{T} \alpha_t \left\| \widehat{V}_t^{-1/4} m_t \right\|^2 &\leq \frac{\alpha}{(1-\beta_1)\sqrt{(1-\beta_2)}} \sum_{t=1}^{T} \sum_{i=1}^{d} \left| g_{t,i} \right| \frac{1}{(1-\gamma)\sqrt{t}}, \\
&\leq \frac{\alpha}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 \sqrt{\sum_{t=1}^{T} \frac{1}{t}},
\end{aligned}$$

where we used the definition of the euclidean norm $\| \cdot \|_2$ and the fact that $\sum_{t \geq 1} 1/\sqrt{t} \leq \sqrt{\sum_{t \geq 1} 1/t}$ in the last step.
If we now apply the bound on harmonic sums $\sum_{t=1}^{T} 1/t \leq (1 + \log T)$, we receive

$$\begin{aligned}
\sum_{t=1}^{T} \alpha_t \left\| \widehat{V}_t^{-1/4} m_t \right\|^2 &\leq \frac{\alpha}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 \sqrt{1 + \log T}, \\
&= \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)\,(1-\gamma)\,\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 . \qquad (1.22)
\end{aligned}$$

Now, plugging inequality (1.22) into inequality (1.21) that swe were interested in originally, gives us

$$\begin{aligned}
&\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \\
&\quad \leq \sum_{t=1}^{T} \left[\frac{1}{2\alpha_t \left(1-\beta_{1t}\right)} \left[\left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2 - \left\| \widehat{V}_t^{1/4} \left(\theta_{t+1} - \theta^*\right) \right\|^2 \right] \right. \\
&\qquad \left. + \frac{\beta_{1t}}{2\alpha_t \left(1-\beta_{1t}\right)} \left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2 \right] + \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 .
\end{aligned}$$

Pulling out the first summand of the sum and shifting the index leads to

$$\begin{aligned}
&\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \\
&\quad \leq \frac{1}{2\alpha_1(1-\beta_1)} \left\| \widehat{V}_1^{1/4} \left(\theta_1 - \theta^*\right) \right\|^2 + \sum_{t=2}^{T} \left[\frac{\left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2}{2\alpha_t(1-\beta_{1t})} - \frac{\left\| \widehat{V}_{t-1}^{1/4} \left(\theta_t - \theta^*\right) \right\|^2}{2\alpha_{t-1}(1-\beta_{1(t-1)})} \right] \\
&\qquad + \sum_{t=1}^{T} \left[\frac{\beta_{1t}}{2\alpha_t(1-\beta_{1t})} \left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2 \right] + \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 .
\end{aligned}$$

Now we expand the first sum and pull out the factor $1/2$

$$\begin{aligned}
&\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \\
&\quad \leq \frac{1}{2\alpha_1(1-\beta_1)} \left\| \widehat{V}_1^{1/4} \left(\theta_1 - \theta^*\right) \right\|^2 \\
&\qquad + \frac{1}{2} \sum_{t=2}^{T} \left[\frac{\left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2}{\alpha_t(1-\beta_{1(t-1)})} - \frac{\left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2}{\alpha_t(1-\beta_{1(t-1)})} + \frac{\left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2}{\alpha_t(1-\beta_{1t})} \right. \\
&\qquad \left. - \frac{\left\| \widehat{V}_{t-1}^{1/4} \left(\theta_t - \theta^*\right) \right\|^2}{\alpha_{t-1}(1-\beta_{1(t-1)})} \right] + \sum_{t=1}^{T} \left[\frac{\beta_{1t}}{2\alpha_t(1-\beta_{1t})} \left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2 \right] \\
&\qquad + \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 .
\end{aligned}$$

If we now consider the fact, that $\beta_{1t} \leq \beta_1$ for all $t = 1, \ldots, T$ and the following observation

$$\frac{\left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2}{\alpha_t(1-\beta_{1t})} - \frac{\left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2}{\alpha_t(1-\beta_{1(t-1)})} \leq \frac{\beta_{1t}}{\alpha_t \left(1-\beta_1\right)^2} \left\| \widehat{V}_t^{1/4} \left(\theta_t - \theta^*\right) \right\|^2 ,$$

then, this leads to

$$
\begin{aligned}
&\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \\
&\quad \leq \frac{1}{2\alpha_1(1-\beta_1)} \left\| \widehat{V}_1^{1/4} (\theta_1 - \theta^*) \right\|^2 \\
&\qquad + \frac{1}{2} \sum_{t=2}^{T} \left[\frac{\left\| \widehat{V}_t^{1/4} (\theta_t - \theta^*) \right\|^2}{\alpha_t (1-\beta_{1(t-1)})} - \frac{\left\| \widehat{V}_{t-1}^{1/4} (\theta_t - \theta^*) \right\|^2}{\alpha_{t-1}(1-\beta_{1(t-1)})} \right] \\
&\qquad + \sum_{t=1}^{T} \left[\frac{\beta_{1t}}{2\alpha_t(1-\beta_{1t})} \left\| \widehat{V}_t^{1/4} (\theta_t - \theta^*) \right\|^2 \right] + \sum_{t=2}^{T} \left[\frac{\beta_{1t}}{\alpha_t (1-\beta_1)^2} \left\| \widehat{V}_t^{1/4} (\theta_t - \theta^*) \right\|^2 \right] \\
&\qquad + \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 , \\
&\quad \leq \frac{1}{2\alpha_1(1-\beta_1)} \left\| \widehat{V}_1^{1/4} (\theta_1 - \theta^*) \right\|^2 \\
&\qquad + \frac{1}{2(1-\beta_1)} \sum_{t=2}^{T} \left[\frac{\left\| \widehat{V}_t^{1/4} (\theta_t - \theta^*) \right\|^2}{\alpha_t} - \frac{\left\| \widehat{V}_{t-1}^{1/4} (\theta_t - \theta^*) \right\|^2}{\alpha_{t-1}} \right] \\
&\qquad + \sum_{t=1}^{T} \left[\frac{\beta_{1t}}{\alpha_t (1-\beta_1)^2} \left\| \widehat{V}_t^{1/4} (\theta_t - \theta^*) \right\|^2 \right] + \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 ,
\end{aligned}
$$

where we simply extended the third sum with the non-negative summand for $t = 1$, in order to join it with the second sum.
Furthermore, using the definition of the operator $\widehat{V}_t$ gives us

$$
\begin{aligned}
&\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \\
&\quad \leq \frac{1}{2\alpha_1(1-\beta_1)} \sum_{i=1}^{d} \widehat{v}_{1,i} (\theta_{1,i} - \theta_i^*)^2 + \frac{1}{2(1-\beta_1)} \sum_{t=2}^{T} \sum_{i=1}^{d} \left[\left(\frac{\widehat{v}_{t,i}^{1/2}}{\alpha_t} - \frac{\widehat{v}_{t-1,i}^{1/2}}{\alpha_{t-1}} \right) (\theta_{t,i} - \theta_i^*)^2 \right] \\
&\qquad + \frac{1}{(1-\beta_1)^2} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{\beta_{1t} \widehat{v}_{t,i}^{1/2} (\theta_{t,i} - \theta_i^*)^2}{\alpha_t} + \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 ,
\end{aligned}
$$

Since we assumed that the parameter space Θ has bounded diameter D_∞, we can write as well

$$
\begin{aligned}
&\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \\
&\quad \leq \frac{1}{2\alpha_1(1-\beta_1)} \sum_{i=1}^{d} \widehat{v}_{1,i} D_\infty^2 + \frac{1}{2(1-\beta_1)} \sum_{t=2}^{T} \sum_{i=1}^{d} \left[\left(\frac{\widehat{v}_{t,i}^{1/2}}{\alpha_t} - \frac{\widehat{v}_{t-1,i}^{1/2}}{\alpha_{t-1}} \right) D_\infty^2 \right] \\
&\qquad + \frac{1}{(1-\beta_1)^2} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{\beta_{1t} \widehat{v}_{t,i}^{1/2} D_\infty^2}{\alpha_t} + \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 ,
\end{aligned}
$$

Lastly, we realize that the first double sum is of telescopic nature. Hence, we can reduce it to

$$\sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta^*) \leq \frac{D_\infty^2 \sqrt{T}}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} \widehat{v}_{T,i} + \frac{D_\infty^2}{(1-\beta_1)^2} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{\beta_{1t} \widehat{v}_{t,i}^{1/2}}{\alpha_t} + \frac{\alpha\sqrt{1+\log T}}{(1-\beta_1)(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 ,$$

where we additionally included the definition of α_T. □

1.3 Neural networks in computer vision

Lastly in this chapter, we want to apply the theory of neural networks to the setting we actually are interested in. This setting is usually called computer vision - basically, machine learning that is applied to images and videos. Since we want to apply neural networks to a problem that deals with images, we need to know how to view images from a mathematical perspective. In order to do so, we need to introduce some quantities first. The first important quantity is a pixel. Basically, this is a single point in the image.

Definition 1.3.1. Let $d \in \mathbb{N}$ and $\Psi = \{0, \dots, 255\}$. Then we define a **pixel with d channels** as

$$\psi = (\psi_1, \psi_2, \dots, \psi_d) ,$$

where for all $i = 1, \dots, d$ holds $\psi_i \in \Psi$. Hence, for each pixel holds $\psi \in \Psi^d$

Remark 1.3.2. We want to distinguish mainly two kinds of pixels. If $d = 1$, we speak of a **black and white pixel**.
If $d = 3$, we speak of an **RGB pixel**. Here, RGB stands for the red, green and blue color channels.

As one may already know, images consist of multiple pixels that are aligned in a grid. We will refer to this grid as a pixel domain.

Definition 1.3.3. Let $M \in \mathbb{N}$ be the **horizontal amount of pixels** and $N \in \mathbb{N}$ be the **vertical amount of pixels**. Then we call the grid Ω defined by

$$\Omega \coloneqq \{1, \dots, M\} \times \{1, \dots, N\} \subset \mathbb{N}^2,$$

the **pixel domain** with the tuple (M, N) being called the **resolution**.

If we now combine the definitions of a pixel and a pixel domain, we can define a mathematical representation of an image - a so called digital image.

Definition 1.3.4. Let $d \in \mathbb{N}$ be the number of channels and Ω a pixel domain with the resolution (M, N). Then we define a **digital image** by

$$\psi = \left(\psi_{ij}\right)_{i,j} = \left(\psi_{ij,1}, \dots, \psi_{ij,d}\right) , \qquad (i,j) \in \Omega,$$

where each ψ_{ij} is a pixel with d channels.
Since for each pixel ψ_{ij} holds $\psi_{ij} \in \Psi^d$, we define the **image domain** as $\Psi^{d \times M \times N}$. We will write $\psi \in \Psi_{d,\Omega} \coloneqq \Psi^{d \times M \times N}$ from now on. If the context is clear, we will reduce the notation up to Ψ.

However, since we usually consider floats and not integers in numerical mathematics, we need to consider a way to represent pixels as floats. In order to do that, we introduce the normed image domain.

Definition 1.3.5. Let Ω be a pixel domain and $d \in \mathbb{N}$ a number of channels. Then we call the Ψ', defined as

$$\Psi' \coloneqq \frac{1}{255}\Psi = \left\{ \frac{0}{255}, \frac{1}{255}, \dots, \frac{255}{255} \right\},$$

the **normed image domain**. Obviously, it holds $\Psi' \subset [0,1]$.

We realize that we can easily transform images from an ordinary image domain to a normed image domain by dividing each pixel value by 255. Equally, we can transform images the other way around by multiplying each pixel value by 255.
Since neural networks rarely produce a value that is a fraction with denominator 255, we need to find a way to process such values. This we will do in the following lemma.

Lemma 1.3.6. *Let $d \in \mathbb{N}$ be a number of channels and let $p \in [0,1]^d$. Then we can consider p as a pixel by transforming it through*

$$\psi_i = \lceil 255 \cdot p_i - 0.5 \rceil, \qquad i = 1, \dots, d.$$

Proof. Since $p \in [0,1]^d$, we can denote p as

$$p = (p_1, \dots, p_d),$$

where for all $i = 1, \dots, d$ holds $p_i \in [0,1]$.
Hence, if we multiply each p_i by 255, it follows that

$$255 \cdot p_i \in [0, 255].$$

Therefore, it holds that

$$255 \cdot p \in [0, 255]^d.$$

If we now round each entry p_i to an integer, we yield exactly the representation defined in definition 1.3.1. □

Now having formally defined what an image is, we can consider how a neural network operating on images looks like. In order to do this, it is sufficient to consider a single neural layer, since neural networks consist of multiple layers. But first, we need some technical assertions to understand how we can actually feed images into neural networks, since images are somewhat matrices and neural networks often operate on arrays.

Lemma 1.3.7. *Let Ω be a pixel domain with resolution (M, N). Then the image ψ with $d \in \mathbb{N}$ channels defined on $\Psi_{d,\Omega}$ can be represented as an MN-dimensional array instead of an $M \times N$-matrix.*

Proof. Let ψ be a matrix with entries $\psi_{ij} \in \Psi_d$, what follows from the definition of an image 1.3.4. Hence, ψ is an $M \times N$- matrix.

Define the rows of ψ by $\widehat{\psi}_i \coloneqq (\psi_{i1}, \dots, \psi_{iN})$ for all $i \in \{1, \dots, M\}$. Then the matrix representation of the picture ψ can be written as

$$\begin{pmatrix} \psi_{11} & \cdots & \psi_{1N} \\ \vdots & \ddots & \vdots \\ \psi_{M1} & \cdots & \psi_{MN} \end{pmatrix} = \begin{pmatrix} \widehat{\psi}_1 \\ \vdots \\ \widehat{\psi}_M \end{pmatrix}.$$

If we now transpose each $\widehat{\psi}_i$ and keep the same representation, we transform the $M \times N$ matrix into an MN-dimensional array

$$\begin{pmatrix} \widehat{\psi}_1^{\mathsf{T}} \\ \vdots \\ \widehat{\psi}_M^{\mathsf{T}} \end{pmatrix} \in \Psi^{MN}.$$

□

Lemma 1.3.7 allows us to feed images into a neural network by considering them as one large array. However, it still is unclear how the images are processed throughout the neural network. In order to formulate this, we need to define a function operating on images.

Definition 1.3.8. Let Ω_0 and Ω_1 be pixel domains with resolutions (M_0, N_0) and (M_1, N_1) respectively. Furthermore, let $d \in \mathbb{N}$ be an arbitrary number of channels.
Then we define an **image operator** T as the continuous mapping

$$\begin{aligned} T : \Psi_{d,\Omega_0} &\to \Psi_{d,\Omega_1} \\ \psi_0 &\mapsto \psi_1 \coloneqq T\,\psi_0, \end{aligned}$$

where T does not change the number of channels d. Thus, we shorten Ψ_{d,Ω_0} to Ψ_{Ω_0} in the following.

The definition 1.3.8 is quite general, since we do not demand any specific properties from the image operator T whatsoever. Indeed, there are some image operators that are commonly used in computer vision. We will take a look at those in the course of this section.
These technicalities helps us introduce neural networks operating on pictures.

Definition 1.3.9. Let φ be an arbitrary activation function and $\widehat{\varphi}$ the component-wise mapping of φ as in 1.1.4 and Ω_0 be an arbitrary pixel domain with resolution $(M_0, N_0) \in \mathbb{N}^2$. Let ψ be an image with number of channels $d \in \mathbb{N}$.
Then a neural layer that operates on images looks as follows

$$\begin{aligned} H : \Psi_{\Omega_0} &\to \Psi_{\Omega_1} \\ \psi_0 &\mapsto \widehat{\varphi}\left(T\,\psi_0 + b\right), \end{aligned}$$

where T is an image operator as in definition 1.3.8, b is a bias and Ω_1 a pixel domain with resolution (M_1, N_1).

With the help of definition 1.3.9 we realise, that each layer H_i of a neural network in a computer vision setting represents an own image space, denoted by Ψ_{Ω_i}. Meaning, that all elements fed to the corresponding layer are images with resolution (M_i, N_i).
Now we want to consider some useful examples of image operators. First, we will take a look at multidimensional convolutions, hence convolutions on images. For more details please look at [1, chapter 9].

Definition 1.3.10. Let Ω_0, Ω_1 be pixel spaces with arbitrary but fixed number of channels and resolutions (M_0, N_0) and (M_1, N_1) respectively. Then the **image convolution operator** $T_{\text{conv}} : \Psi_{\Omega_0} \to \Psi_{\Omega_1}$ is defined by

$$\begin{aligned} T_{\text{conv}} : \Psi_{\Omega_0} &\to \Psi_{\Omega_1} \\ \psi_0 &\mapsto \psi_1 \coloneqq T_{\text{conv}}\, \psi_0 \,. \end{aligned}$$

This operator is defined component-wise by

$$(\psi_1)_{ij} \coloneqq (\psi_0 * k)_{ij} = \sum_{m,n=1}^{s} (\psi_0)_{m+i,n+j}\, k_{mn}, \tag{1.23}$$

where $(i,j) \in \Omega_1 \coloneqq \{1, \dots, M_0 - s + 1\} \times \{1, \dots, N_0 - s + 1\}$. Hence, $M_1 = M_0 - s + 1$ and $N_1 = N_0 - s + 1$.
Furthermore, k is called an s-**convolution kernel**, an image with resolution (s, s) with $s \in \mathbb{N}$.

Another very useful example is the pooling operator. We will distinguish between average and min- and max-pooling.

Definition 1.3.11. Let Ω_0, Ω_1 be pixel spaces with arbitrary but fixed number of channels and resolutions (M_0, N_0) and (M_1, N_1) respectively. Furthermore, $s \in \mathbb{N}$ is called **stride** and $I \coloneqq \{1, \dots, p_1\} \times \{1, \dots, p_2\} \subset \mathbb{N}^2$ with $p_1, p_2 \in \mathbb{N}$ is called **pooling**.
Then the **average-pooling operator** $T_{\text{avg}} : \Psi_{\Omega_0} \to \Psi_{\Omega_1}$ with stride s and pooling I is defined by

$$\begin{aligned} T_{\text{avg}} : \Psi_{\Omega_0} &\to \Psi_{\Omega_1} \\ \psi_0 &\mapsto \psi_1 \coloneqq T_{\text{avg}}\, \psi_0 \,. \end{aligned}$$

This operator is defined component-wise by

$$(\psi_1)_{ij} \coloneqq \frac{1}{p_1 p_2} \sum_{(m,n) \in I} (\psi_0)_{m+i,n+j}\,, \tag{1.24}$$

where $(i,j) \in \Omega_1 \coloneqq \{1, \dots, M_0 - p_1 + 1\} \times \{1, \dots, N_0 - p_2 + 1\}$. Hence, $M_1 = M_0 - p_1 + 1$ and $N_1 = N_0 - p_2 + 1$.

Definition 1.3.12. Let Ω_0, Ω_1 be pixel spaces with arbitrary but fixed number of channels and resolutions (M_0, N_0) and (M_1, N_1) respectively. Furthermore, $s \in \mathbb{N}$ is called **stride** and $I \coloneqq \{1, \dots, p_1\} \times \{1, \dots, p_2\} \subset \mathbb{N}^2$ with $p_1, p_2 \in \mathbb{N}$ is called **pooling**.
Then the **min-pooling operator** $T_{\text{min}} : \Psi_{\Omega_0} \to \Psi_{\Omega_1}$ with stride s and pooling I is defined by

$$\begin{aligned} T_{\text{min}} : \Psi_{\Omega_0} &\to \Psi_{\Omega_1} \\ \psi_0 &\mapsto \psi_1 \coloneqq T_{\text{min}}\, \psi_0 \,. \end{aligned}$$

This operator is defined component-wise by

$$(\psi_1)_{ij} \coloneqq \min_{(k,l) \in I} (\psi_0)_{kl}\,, \tag{1.25}$$

where $(i,j) \in \Omega_1 \coloneqq \{1, \dots, M_0 - p_1 + 1\} \times \{1, \dots, N_0 - p_2 + 1\}$. Hence, $M_1 = M_0 - p_1 + 1$ and $N_1 = N_0 - p_2 + 1$.

Definition 1.3.13. Let Ω_0, Ω_1 be pixel spaces with arbitrary but fixed number of channels and resolutions (M_0, N_0) and (M_1, N_1) respectively. Furthermore, $s \in \mathbb{N}$ is called **stride** and $I \coloneqq \{1, \dots, p_1\} \times \{1, \dots, p_2\} \subset \mathbb{N}^2$ with $p_1, p_2 \in \mathbb{N}$ is called **pooling**.
Then the **max-pooling operator** $T_{\max} : \Psi_{\Omega_0} \to \Psi_{\Omega_1}$ with stride s and pooling I is defined by

$$\begin{aligned} T_{\max} : \Psi_{\Omega_0} &\to \Psi_{\Omega_1} \\ \psi_0 &\mapsto \psi_1 \coloneqq T_{\max}\, \psi_0 \,. \end{aligned}$$

This operator is defined component-wise by

$$(\psi_1)_{ij} \coloneqq \max_{(k,l)\in I} (\psi_0)_{kl} \,, \tag{1.26}$$

where $(i, j) \in \Omega_1 \coloneqq \{1, \dots, M_0 - p_1 + 1\} \times \{1, \dots, N_0 - p_2 + 1\}$. Hence, $M_1 = M_0 - p_1 + 1$ and $N_1 = N_0 - p_2 + 1$.

At this point, we should mention that each of the recently introduced operators can be used to connect two neural layers. This we will put down in writing in the following proposition.

Proposition 1.3.14. *Let T_{conv} be an image convolution operator with s-convolution kernel k. Furthermore, let Ψ_0 and Ψ_1 be arbitrary image domains and φ be an arbitrary activation function.*
Then

$$\begin{aligned} H_{conv} : \Psi_0 &\to \Psi_1, \\ \psi_0 &\mapsto H_{conv}(\psi_0) = \widehat{\varphi}\,(T_{conv}\, \psi_0 + b) \,, \end{aligned}$$

*defines a neural layer, where the parameters θ are the s-convolution kernel k. We will call such a layer a **convolutional neural layer** and denote the layers' parameters $\theta = T_{conv}$, since the kernel defines the convolution operator uniquely.*

1.4 Probability and Statistics

Another important topic for this thesis is probability theory and statistics. Especially in chapter 3 this will be crucial to analyse variational autoencoding neural networks in more detail.
We want to begin by introducing the basic quantities in probability theory, merely to define a notation throughout the thesis.

Definition 1.4.1. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a **probability space** and $(\mathcal{X}, \mathcal{A})$ be a **measurable space**. Furthermore, let X be a **random variable** defined over $\mathcal{X}$. Then we define the measurable function $p : (\Omega, \mathcal{A}) \to (\mathcal{X}, \mathcal{A})$ that satisfies

$$\int_\Omega p^+ d\,\mathrm{P} < \infty \qquad\qquad \text{or} \int_\Omega p^- d\,\mathrm{P} < \infty,$$

where $p^+ = \max\{p, 0\}$ and $p^- = -\min\{p, 0\}$ describe the positive and negative part of p, as **probability density function** of P.
Moreover, let Θ be a **parameter space**. We will denote the probability density function as $p(x; \theta)$, where θ describes the parameters of the probability distribution of X.

Here we should note, that we want to consider the defining parameters of a probability distribution in the arguments of the underlying probability density function. This means that for example for a normally distributed random variable X with mean μ and variance σ^2 we would denote its probability density function as $p(x; \mu, \sigma^2)$.
Another important quantity we need to consider beforehand is the joint distribution, as well as marginal distributions.

Definition 1.4.2. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space and $(\mathcal{X}, \mathcal{A})$, $(\mathcal{Y}, \mathcal{A})$ be measurable spaces. Furthermore, let X and Y be random variables defined on $\mathcal{X}$ and $\mathcal{Y}$, respectively. Then the distribution $p(x, y)$of the random variable (X, Y) defined on $\mathcal{X} \times \mathcal{Y}$ is called the **joint probability distribution**.

By marginalizing over one of both random variables we receive the marginal distribution.

Definition 1.4.3. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space, $(\mathcal{X}, \mathcal{A})$, $(\mathcal{Y}, \mathcal{A})$ be measurable spaces and X, Y random variables defined on the corresponding measurable spaces. Then we define the **marginal probability distribution of** X as

$$p(x) = \int p(x, z)dz,$$

as well as the **marginal probability distribution of** Y as

$$p(x) = \int p(x, z)dx.$$

Another important quantity is the conditional probability. Since if there is partial information on the outcome of a random experiment, the probabilities for the possible events may change. Thus, we introduce this quantity analogous to [9, Chapter 8].

Definition 1.4.4. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space and $A \in \mathcal{A}$. We define the **conditional probability given** A for any $B \in \mathcal{A}$ by

$$\mathrm{P}(B|A) = \begin{cases} \frac{P(A \cap B)}{P(A)}, & \text{if } \mathrm{P}(A) > 0, \\ 0, & \text{else.} \end{cases}$$

At this point we should note, that the conditional probability, introduced in definition 1.4.4 is indeed a probability measure, see [9, Theorem 8.4]. Lastly, we want to introduce the famous Bayes' formula, see e.g [9, Theorem 8.7]. This will be fundamental for our Bayesian learning setting in chapter 3.

Theorem 1.4.5. *Let $(\Omega, \mathcal{A}, P)$ be a probability space and I be a countable set. Furthermore, let $(B_i)_{i \in I}$ be a sequence of pairwise disjoint sets with $P(\bigcup_{i \in I} B_i) = 1$.*
Then for any $A \in \mathcal{A}$ with $P(A) > 0$ and any $k \in I$ holds

$$P(B_k|A) = \frac{P(A|B_k)\, P(B_k)}{\sum_{i \in I} P(A|B_i)\, P(B_i)}.$$

Proof. Using the definition 1.4.4 of the conditional probability twice, we receive

$$\mathrm{P}(B_k|A) = \frac{\mathrm{P}(B_k \cap A)}{\mathrm{P}(A)} = \frac{\mathrm{P}(A|B_k)\, \mathrm{P}(B_k)}{\mathrm{P}(A)}. \tag{1.27}$$

Now, due to the σ-additivity of P, the following holds

$$\mathrm{P}(A) = \mathrm{P}\left(\bigcup_{i\in I}(A\cap B_i)\right) = \sum_{i\in I}\mathrm{P}\left(A\cap B_i\right) = \sum_{i\in I}\mathrm{P}(A|B_i)\,\mathrm{P}(B_i).$$

Plugging this result right into equation (1.27) yields

$$\mathrm{P}(B_k|A) = \frac{\mathrm{P}(A|B_k)\,\mathrm{P}(B_k)}{\sum_{i\in I}\mathrm{P}(A|B_i)\,\mathrm{P}(B_i)},$$

which is exactly the proposed assertion. □

Since we will be interested in modelling probability distributions to approximate observed data as well as possible, we now want to consider how to construct a family of distributions by tweaking the parameters of the probability distribution. One example of such a family are parametric models. These are families of distributions that can be indexed by a finite number of parameters.

Definition 1.4.6. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space and X a random variable defined over Ω. Furthermore, let Θ be an arbitrary parameter space.
Then we denote the **parametric model** as the family of distributions

$$\mathcal{P}_\Theta = \{p(x;\theta) : \theta\in\Theta\}.$$

Example 1.4.7. Let's consider some elementary examples of parametric models.

1. Let $U \subset \mathbb{N}^2$ be a finite subset with tuples $(a,b) \in U$ and $X \sim \mathcal{U}(a,b)$ a uniformly distributed random variable. Then the parametric model of X with regard to U looks as follows

   $$\mathcal{P}_U = \{p(x;a,b) : (a,b)\in U\},$$

   where for each probability density function $p(x;a,b)$ holds

   $$p(x;a,b) = \begin{cases} 1/(b-a), & \text{if } x\in[a,b],\\ 0, & \text{otherwise.}\end{cases}$$

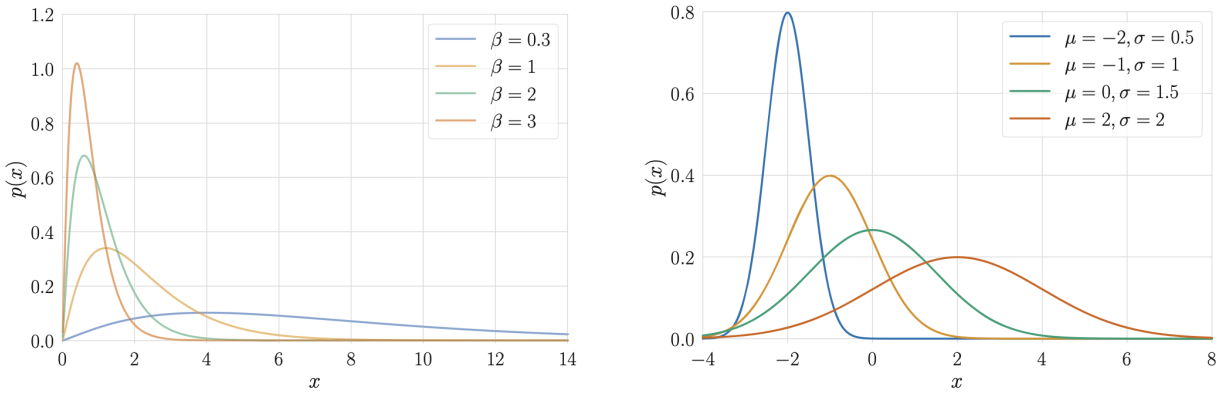
2. Let $N \subset \mathbb{Q}\times\mathbb{Q}_+$ be a finite subset with tuples $(\mu,\sigma^2) \in N$ and $X \sim \mathcal{N}(\mu,\sigma^2)$ a normally distributed random variable. Then the parametric model of X with regard to N looks as follows

   $$\mathcal{P}_N = \{p(x;\mu,\sigma^2) : (\mu,\sigma^2)\in N\},$$

   where for each probability density function $p(x;\mu,\sigma^2)$ holds

   $$p(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\,\sigma^2}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Another way to generate a family of distributions is to alter an underlying base probability density function, hence named standard probability density function. Possible alterations might be shifting or scaling (or both) the standard probability density function. Therefore, we cite a theorem from [10, Theorem 2.1], which proposes exactly such a construction.



(a) **(b)**

Figure 1.2: Illustration of location-scale families for the Gamma distribution and the Gaussian distribution, respectively. In the parametrization of the the Gamma function with the rate parameter β, the scale parameter as defined in theorem 1.4.8 is actually $\sigma = 1/\beta$. Note that as the scale σ increases the distribution becomes less concentrated around the location parameter μ. In particular, $\lim_{\sigma\to\infty} p(x;\mu,\sigma) = \delta(x-\mu)$. **(a)** Members of the same scale family of Gamma distributions with shape parameter $\alpha = 2.2$. **(b)** Members of the same location-scale family of Gaussian distributions.

Theorem 1.4.8. *Let $p(x)$ be a probability density function and μ and $\sigma > 0$ constants. Then the following functions are also probability density functions*

$$g(x;\mu,\sigma) = \frac{1}{\sigma} p\left(\frac{x-\mu}{\sigma}\right).$$

*We refer to the parameters μ as **location parameter** and σ as **scale parameter**. Moreover, we call the family $\mathcal{P}_{\mu,\sigma} = \{g(x;\mu,\sigma) : \mu \text{ and } \sigma > 0\}$ a **location-scale family**.*

Now, let's consider some examples of location-scale families. Moreover, we borrow two graphics from [10] that serve to illustrate the assertion from theorem 1.4.8. These graphics are shown in figure 1.2 and show the generation of a Gamma and a Gaussian distributed probability density function family, which we consider in example 1.4.9.

Example 1.4.9. The following examples are all location-scale families.

1. Let $\alpha, \beta > 0$ be constants. Then the Gamma distribution $\mathrm{Ga}(\alpha,\beta)$ is a scale family for each value of the shape parameter α

$$p(x;\alpha,\beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}.$$

2. Let $\mu \in \mathbb{R}$ and $\sigma > 0$ be constants. Then the Gaussian distribution $\mathcal{N}(\mu,\sigma^2)$ is a location-scale family for both, the location parameter μ and the scale parameter σ, respectively. This leads to

$$p(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\,\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2},$$

similarly to example 1.4.7.

2 Autoencoders

Having introduced the basics of neural networks in chapter 1 we can consider a specific architecture of a neural network, a so called autoencoding neural network, or short: autoencoder. The idea of autoencoders is to take a given input, compress the input to a smaller size (usually called encoding) and afterwards, expand it as accurately as possible to the original size again (usually called decoding). Such an architecture is widely used in different applications. For example on social media platforms, where users send images to one another. Instead of sending the original image, which size might very well be a couple of megabytes, the image is firstly being encoded and sent in the compressed representation. Afterwards, the recipient decodes the image to its original representation. This way one has only to transmit the encoded representation, which usually is smaller by magnitudes. Another very important application of autoencoders is in the machine learning field. Most state of the art machine learning models are using autoencoding structures, since it is way more efficient to first encode the data and then run a model on the encoded data. This is quite straight-forward, considering the same argument as in the previous use case - the encoded data being smaller by magnitudes. This way processing the samples can happen much faster compared to the non-encoded data samples and secondly, it makes storing data (on the drive and in memory) much more efficient.
In this chapter we want to consider how to formulate autoencoding neural networks from a mathematical point of view, take a look at some important results and lastly, analyse the theory in multiple applications using Python.

2.1 Conceptional ideas

As already mentioned, an autoencoding neural network first compresses/encodes the input data to a smaller representation. The size of this smaller representation is usually referred to as bottleneck of the autoencoder. Afterwards, the autoencoding neural network expands/decodes the data to its original size. Hence, we can divide these two steps into separate architectures - the encoding and the decoding part of the neural network, which we will formulate separately. In figure 2.1 we can take a look at a visual example of an autoencoding architecture.

If we divide the autoencoding structure as described above, we firstly obtain the encoder as we can see in figure 2.2 or formally defined as follows.

Definition 2.1.1. Let Θ be a parameter space and $\theta \in \Theta$ a set of parameters, $L \in \mathbb{N}$ and $d_1, \ldots, d_L \in \mathbb{N}$. Let further φ be an activation function and $f_{\varphi,L,\theta}$ a neural network.
If the neural network $f_{\varphi,L,\theta}$ fulfils the condition $n_i = d_1 \geq \ldots \geq d_L = n_o$ with $n_i, n_o \in \mathbb{N}$ being the input and output dimensions respectively, then we speak of an **encoding neural network** (or short: **encoder**) and denote it as f_e.

For the second part of the divided autoencoding structure, we obtain the decoder as we can see in figure 2.3. We can define this architecture analogously to the encoder in definition 2.1.1.

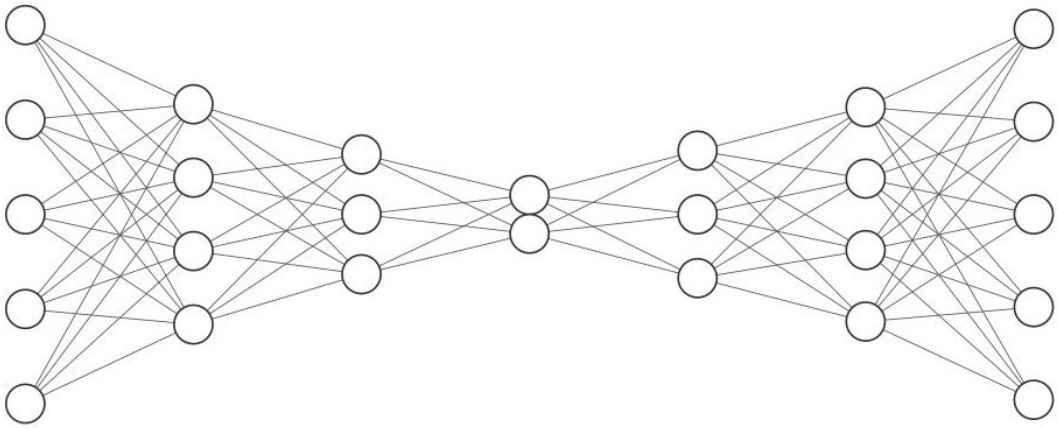


Figure 2.1: An autoencoding neural network with input and output $x, y \in \mathbb{R}^5$. The five hidden layers have dimensions 4, 3, 2, 3 and 4 respectively. Hence, the bottleneck dimension is 2 in this example. The graphic was generated with http://alexlenail.me/NN-SVG/index.html

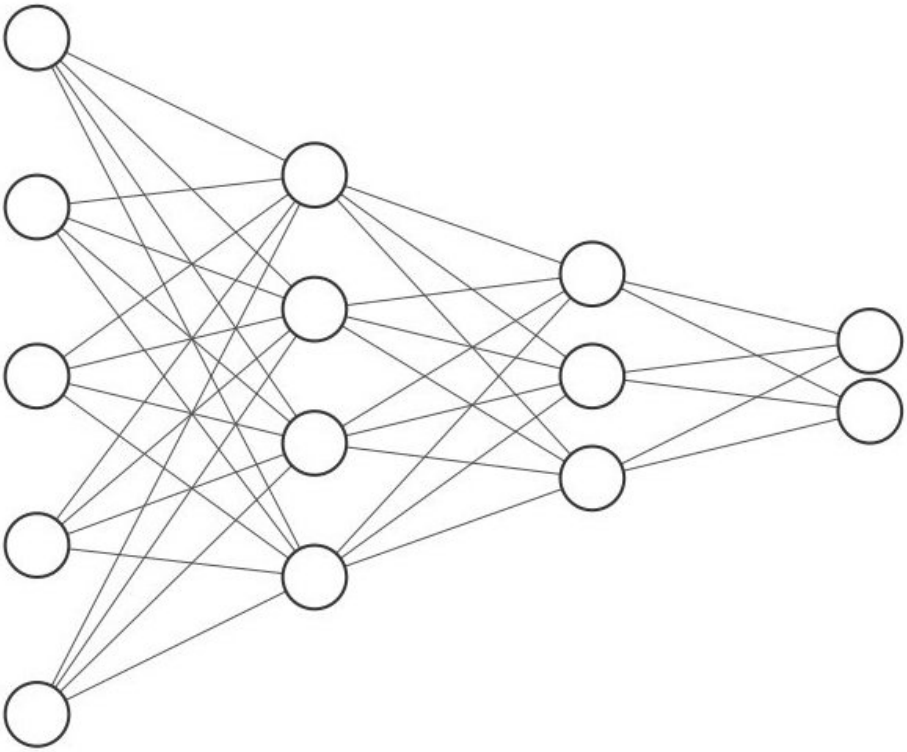


Figure 2.2: An encoding neural network with input $x \in \mathbb{R}^5$ and output $y \in \mathbb{R}^2$. The two hidden layers have dimensions 4 and 3. Hence, the encoder reduces the data dimensionality from 5 to 2 dimension. The graphic was generated with http://alexlenail.me/NN-SVG/index.html

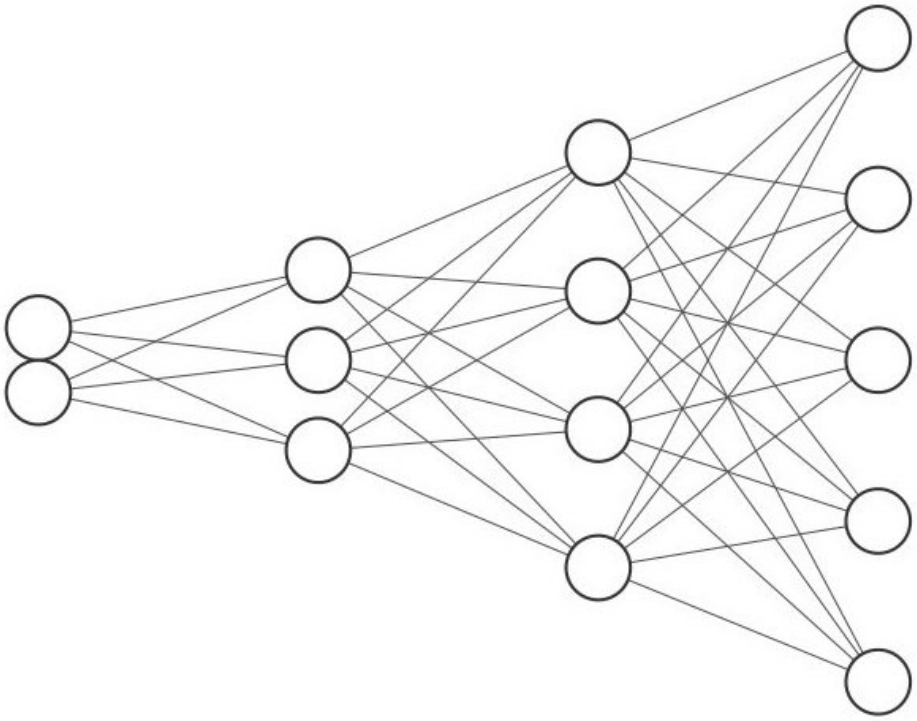


Figure 2.3: A decoding neural network with input $x \in \mathbb{R}^2$ and output $y \in \mathbb{R}^5$. The two hidden layers have dimensions 3 and 4. Hence, the decoder expands the data dimensionality from 2 to 5 dimensions. The graphic was generated with http://alexlenail.me/NN-SVG/index.html

Definition 2.1.2. Let Θ be a parameter space and $\theta \in \Theta$ a parameter, $L \in \mathbb{N}$ and $d_1, \ldots, d_L \in \mathbb{N}$. Let further φ be an activation function and $f_{\varphi,L,\theta}$ a neural network.
If the neural network $f_{\varphi,L,\theta}$ fulfils the condition $n_i = d_1 \leq \ldots \leq d_L = n_o$ with $n_i, n_o \in \mathbb{N}$ being the input and output dimensions respectively, then we speak of an **decoding neural network** (or short: **decoder**).

Before combining the encoding and the decoding structure to obtain the autoencoding neural network, we need to consider the following technicality first.

Lemma 2.1.3. *Let f_1, f_2 be two neural networks of depths $L_1, L_2 \in \mathbb{N}$ with parameters $\theta_1, \theta_2 \in \Theta$, where Θ is an arbitrary parameter space. Furthermore, let the dimensions of each layer be $d_1, \ldots, d_{L_1} \in \mathbb{N}$ of f_1 and $\widetilde{d}_1, \ldots, \widetilde{d}_{L_2} \in \mathbb{N}$ of f_2. Additionally, let $d_{L_1} = \widetilde{d}_1$.*
Then their composition $f_2 \circ f_1$ is a neural network of depth $L_1 + L_2$ with parameters (θ_1, θ_2).

Proof. Since f_1 is a neural network of depth L_1 with parameters θ_1, its architecture looks like

$$f_1(x) = H_{L_1} \circ H_{L_1-1} \circ \ldots H_2 \circ H_1(x), \qquad x \in \mathbb{R}^{d_1}\,. \tag{2.1}$$

Analogously, we can write f_2 as

$$f_2(y) = \widetilde{H}_{L_2} \circ \widetilde{H}_{L_2-1} \circ \ldots \widetilde{H}_2 \circ \widetilde{H}_1(y), \qquad y \in \mathbb{R}^{\widetilde{d}_1}\,. \tag{2.2}$$

Since we assumed that the output dimension d_{L_1} of the neural network f_1 is equal to the input

dimension $\widetilde{d}_1$ of the neural network f_2, we can consider the result of (2.1) as input for (2.2)

$$y \coloneqq H_{L_1} \circ H_{L_1-1} \circ \dots H_2 \circ H_1(x), \qquad x \in \mathbb{R}^{d_1}.$$

Hence, we obtain

$$\begin{aligned} f_2(y) &= \widetilde{H}_{L_2} \circ \widetilde{H}_{L_2-1} \circ \dots \widetilde{H}_2 \circ \widetilde{H}_1(y), \\ f_2\left(f_1\left(x\right)\right) &= \widetilde{H}_{L_2} \circ \widetilde{H}_{L_2-1} \circ \dots \widetilde{H}_2 \circ \widetilde{H}_1\left(H_{L_1} \circ H_{L_1-1} \circ \dots H_2 \circ H_1(x)\right), \\ f_2\left(f_1\left(x\right)\right) &= \widetilde{H}_{L_2} \circ \widetilde{H}_{L_2-1} \circ \dots \widetilde{H}_2 \circ \widetilde{H}_1 \circ H_{L_1} \circ H_{L_1-1} \circ \dots H_2 \circ H_1(x), \qquad x \in \mathbb{R}^{d_1}. \end{aligned} \tag{2.3}$$

Therefore, from (2.3) follows that the composition $f_2 \circ f_1$ is a neural network of depth $L_1 + L_2$. Lastly, we consider the parameters θ of the neural network $f_2 \circ f_1$. Since the parameters of a neural network were defined as $\theta = (\theta_1, \dots, \theta_L)$, where each entry is defined as $\theta_i = (W_i, b_i)$ and denotes the weight and bias of each layer H_i or $\widetilde{H}_i$, respectively, we can write the parameters of both neural networks as

$$\begin{aligned} \theta_1 &\coloneqq (\theta_1, \dots, \theta_{L_1}) = \left((W_1, b_1), \dots, (W_{L_1}, b_{L_1})\right), \\ \theta_2 &\coloneqq (\widetilde{\theta}_1, \dots, \widetilde{\theta}_{L_2}) = \left((\widetilde{W}_1, \widetilde{b}_1), \dots, (\widetilde{W}_{L_2}, \widetilde{b}_{L_2})\right). \end{aligned}$$

Hence, the composition $f_2 \circ f_1$ has the parameters

$$\begin{aligned} \theta &= \left((W_1, b_1), \dots, (W_{L_1}, b_{L_1}), (\widetilde{W}_1, \widetilde{b}_1), \dots, (\widetilde{W}_{L_2}, \widetilde{b}_{L_2})\right) \\ &= \left(\theta_1, \dots \theta_{L_1}, \widetilde{\theta}_1, \dots, \widetilde{\theta}_{L_2}\right) =: (\theta_1, \theta_2). \end{aligned}$$

□

Lemma 2.1.3 allows us to consider a modern approach to neural networks, a so called modular approach. Essentially, we consider entire structures like the encoding and the decoding neural network as a self-contained module. These modules can now easily be put together by considering them in series. This is very useful in practice, since modern neural networks consist of thousands of layers and thus billions of parameters. Considering a modular approach one can therefore divide the whole neural network and tune each module separately.

Theorem 2.1.4. *Let Θ be a parameter space, $N \in \mathbb{N}$ and $L_1, \dots, L_N \in \mathbb{N}$. Furthermore, let $f_1, \dots, f_N$ be neural networks with parameters $\theta_1, \dots, \theta_N \in \Theta$ and depths $L_1, \dots, L_N$, respectively. Lastly, let the output dimension of f_i match the input dimension of f_{i+1} for all $i \in \{1, \dots, N-1\}$.*
Then the composition

$$f \coloneqq f_N \circ f_{N-1} \circ \dots \circ f_1,$$

is a neural network with parameters $\theta = (\theta_1, \dots \theta_N)$ of depth $L = L_1 + \dots + L_N$.

Proof. Applying lemma 2.1.3 to f_1 and f_2 yields the composed neural network $f^{(1)} \coloneqq f_2 \circ f_1$ with parameters $\theta^{(1)} \coloneqq (\theta_1, \theta_2)$ and depth $L^{(1)} \coloneqq L_1 + L_2$.
If we now apply lemma 2.1.3 once again to $f^{(1)}$ and f_3, we receive $f^{(2)} \coloneqq f_3 \circ f^{(1)}$ with parameters

$\theta^{(2)} \coloneqq (\theta^{(1)}, \theta_3) = (\theta_1, \theta_2, \theta_3)$ and depth $L^{(2)} \coloneqq L^{(1)} + L_3 = L_1 + L_2 + L_3$.
We realize, that iteratively applying lemma 2.1.3 yields after $N-1$ applications

$$\begin{aligned} f^{(N-1)} &= f_N \circ f_{N-1} \circ \ldots \circ f_1, \\ \theta^{(N-1)} &= (\theta_1, \ldots, \theta_N), \\ L^{(N-1)} &= \sum_{i=1}^{N-1} L_i. \end{aligned}$$

Therefore the assertion is proven. □

Definition 2.1.5. Let f_e and f_d be an encoding and a decoding neural network with input dimension n_i in $\mathbb{N}$ and output of the encoding neural network n_b in $\mathbb{N}$, that we will refer to as **bottleneck** of the autoencoding neural network. Then we define an **autoencoding neural network** f_a as the composition

$$\begin{aligned} f_a : \mathbb{R}^{n_i} &\to \mathbb{R}^{n_i}, \\ x &\mapsto (f_d \circ f_e)(x). \end{aligned}$$

Remark 2.1.6. Let f_e and f_d be an encoding and a decoding neural network as in definition 2.1.5. Then the composition $f_d \circ f_e$ is indeed a neural network, following from lemma 2.1.3.

2.2 Training of autoencoders

When tackling the question of how to train an autoencoding neural network, we realize that in contrast to regular neural networks, where we compare the output of the neural network to a label, in the current setting we can compare the input data to the computed output, since the goal of an autoencoding neural network ultimately is to alter and reconstruct images. In other words, we approach this optimization problem in an unsupervised learning setting. This forces us to consider slightly different loss functions than in the supervised learning setting.

Definition 2.2.1. Let X, Y be Banach spaces representing the input and output space, respectively.
Then a continuous function defined as

$$\begin{aligned} L : X \times Y &\to \mathbb{R}, \\ (x, p(x)) &\mapsto L(x, p(x)), \end{aligned}$$

is called **unsupervised loss function**.

There are multiple important loss functions in computer vision. We will consider a couple of those in the following example. For further reading please take a look at [11]

Example 2.2.2. Let Ω be a pixel domain with resolution (M, N) and d the number of channels. Furthermore, let f be a neural network with arbitrary but fixed architecture.
Then the following functions are loss functions operating on images.

Mean Squared Error (MSE):

$$\mathrm{L}_{\mathrm{MSE}}(\psi, f(\psi)) = \sum_{i=1}^{M} \sum_{j=1}^{N} \left\| \psi_{ij} - f(\psi)_{ij} \right\|^2,$$

Binary Cross-Entropy (BCE):

$$\mathrm{L}_{\mathrm{BCE}}(\psi, f(\psi)) = -\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\Big(\psi_{ij}\log\Big(f(\psi)_{ij}\Big) + \big(1-\psi_{ij}\big)\log\Big(1-f(\psi)_{ij}\Big)\Big),$$

where ψ denotes an image defined on Ψ_Ω.

Remark 2.2.3. The Binary Cross-Entropy loss function is usually used for binary classification problems. However, it still works in computer vision.

2.3 Applications

In this section we want to train a couple of own autoencoding neural networks on the MNIST dataset - a dataset consisting of handwritten digits. Lastly, we will consider a composition of MNIST images in order to simulate a scenario that will allow the Duckeneers GmbH to generate new data. We will consider some various architectures of neural networks and visualise the results in a comprehensible manner.
First, we want to consider the most simple architecture, a fully connected linear neural network.

Definition 2.3.1. Let Θ be an arbitrary parameter space, $L \in \mathbb{N}$ and $d_1, \ldots, d_L \in \mathbb{N}$. Furthermore, let φ be an arbitrary activation function.
Then an encoding neural network, where each layer $H_1, \ldots, H_L$ is defined as

$$H_i(x) = \widehat{\varphi}(W_i x + b_i), \qquad x \in \mathbb{R}^{d_i}, i \in \{1, \ldots, L\},$$

where $\theta_i = (W_i, b_i) \in \Theta$ are the parameters of the i-th layer, see (1.2). Such an encoding neural network is called a **linear encoder**.

Analogously, we define a linear decoding neural network.

Definition 2.3.2. Let Θ be an arbitrary parameter space, $L \in \mathbb{N}$ and $d_1, \ldots, d_L \in \mathbb{N}$. Furthermore, let φ be an arbitrary activation function.
Then a decoding neural network, where each layer $H_1, \ldots, H_L$ is defined as

$$H_i(x) = \widehat{\varphi}(W_i x + b_i), \qquad x \in \mathbb{R}^{d_i}, i \in \{1, \ldots, L\},$$

where $\theta_i = (W_i, b_i) \in \Theta$ are the parameters of the i-th layer H_i, see (1.2). Such a decoding neural network is called a **linear decoder**.

With the definitions of the linear encoder and linear decoder, we now can define a linear autoencoder.

Definition 2.3.3. Let f_e and f_d be a linear encoder and a linear decoder. Then a **linear autoencoder** f_{lin} is defined as the composition

$$f_{\mathrm{lin}} \coloneqq f_d \circ f_e.$$

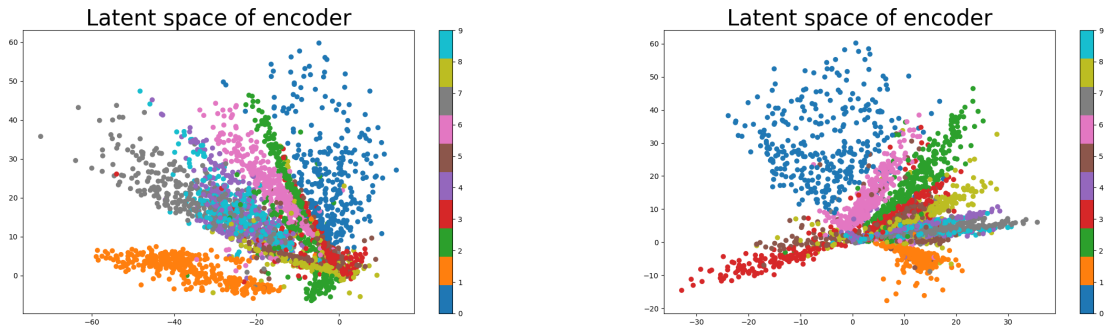


Figure 2.4: The figure illustrates the latent space of the linear autoencoder with bottleneck $n_b = 2$ optimized with an ADAM optimizer (left) and an AMSGrad optimizer (right). Each dot is one encoded image of a digit. The color and the corresponding color map represent the digit that was encoded.

Before considering specific examples on the MNIST dataset, we need to consider some properties of the said dataset first.

Remark 2.3.4. The MNIST dataset $\mathcal{D}$ consists of greyscale images with a resolution of $(28, 28)$. Hence, the images are defined on the pixel domain Ω of $\mathcal{D}$ with $\Omega = \{1, \dots, 28\} \times \{1, \dots, 28\}$ with only one channel.

Now, let us take a look at a specific example of a linear autoencoder defined on the MNIST dataset.

Algorithm 4 Linear Autoencoder

Let the input and output dimensions be $n_i, n_o \in \mathbb{N}$. Furthermore, let the linear encoder and the linear decoder have k hidden linear layers with dimensions $n_1, n_2, \dots, n_k \in \mathbb{N}$ with bottleneck $n_b \in \mathbb{N}$.
Let the chosen optimizer be ADAM first and afterwards be AMSGrad with a learning rate $\gamma = 3 \times 10^{-4}$ and the MSE loss function. We will perform the training on 10.000 epochs with a batch size of 512.

Require: $\gamma \leftarrow 3 \times 10^{-4}$

```
for epoch in {1, ..., 10.000} do
    for image in batch do
        image = image.reshape(784)          ▷ Convert the image from matrix to array.
        encoded = encoder(image)            ▷ Encode the image onto latent space.
        reconstructed = decoder(encoded)    ▷ Decode the encoded image.
        loss = MSE(reconstructed, image)    ▷ Compare the output to the input.
        optimization(loss, γ)               ▷ Perform an optimization step.
    end for
end for
```

Let's take a look at the trained autoencoders using the ADAM and the AMSGrad optimizer. In figure 2.4 we can see the latent space of the models. Furthermore, in figure 2.5 we can see ten reconstructions for each of the ten digits. The training progresses are illustrated in figure 2.6 with the ADAM optimizer and in figure 2.7 with an AMSGrad optimizer, respectively.

Now, let's consider another latent dimension, resulting in 3 dimensions. This allows the neural network to save more information upon encoding the data and hence, it should be able to per-

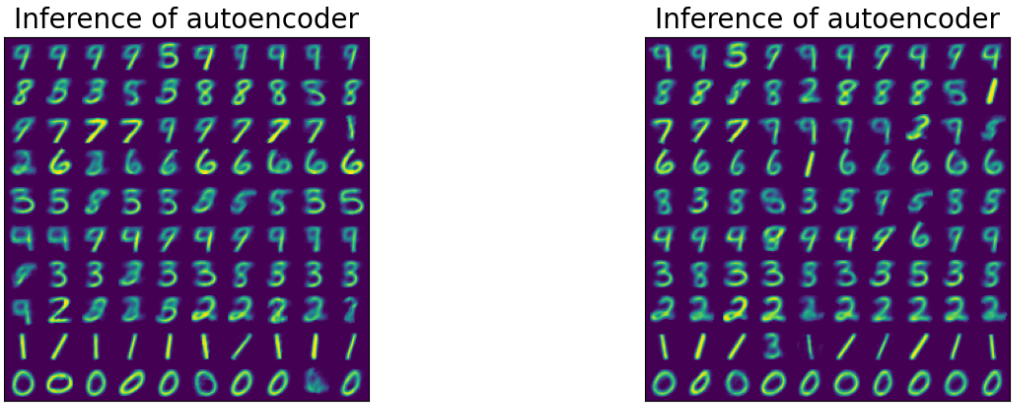


Figure 2.5: The figure illustrates the inference of the linear autoencoder with bottleneck $n_b = 2$ optimized with an ADAM optimizer (left) and an AMSGrad optimizer (right). The inference are ten reconstructed images for each of the ten digits.

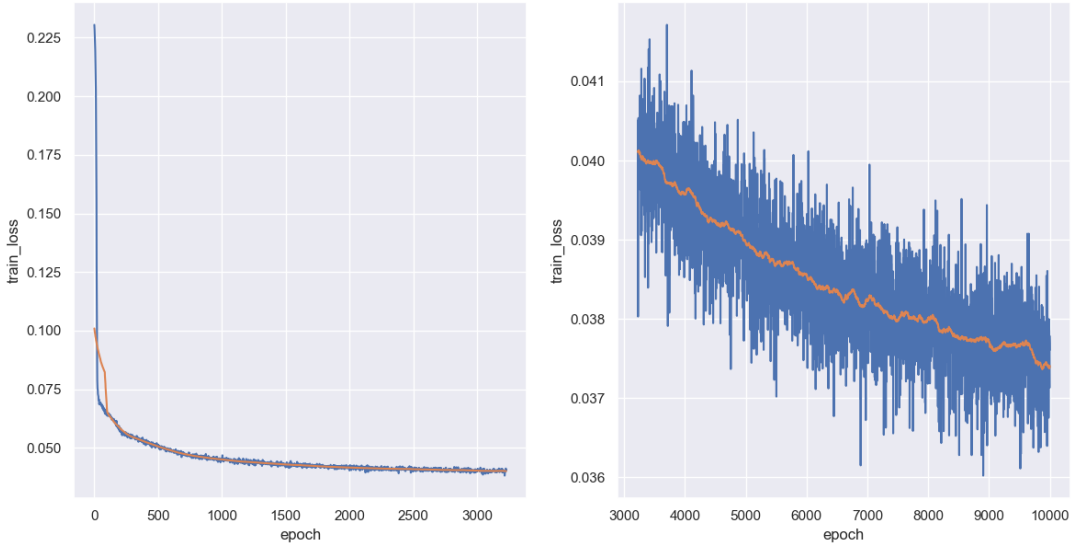


Figure 2.6: The figure illustrates the training progresses of the linear autoencoder with bottleneck $n_b = 2$ optimized with an ADAM optimizer with epochs on one axis and corresponding training loss on the other axis.On the left side we see the first 3.500 epochs and on the right side the following epochs until 10.000. The blue line represents the loss in each epoch and the orange line represents the moving average over 100 epochs to point out the trend of the training progress.

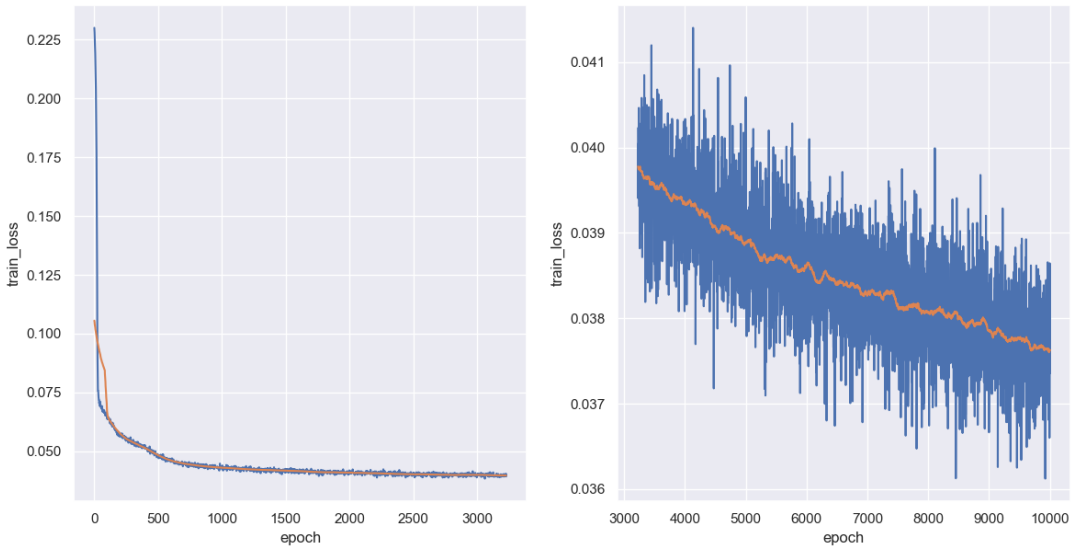


Figure 2.7: The figure illustrates the training progresses of the linear autoencoder with bottleneck $n_b = 2$ optimized with an AMSGrad optimizer with epochs on one axis and corresponding training loss on the other axis.On the left side we see the first 3.500 epochs and on the right side the following epochs until 10.000. The blue line represents the loss in each epoch and the orange line represents the moving average over 100 epochs to point out the trend of the training progress.

form better reconstructions. However, it makes visualising the latent space a bit more difficult. In figure 2.8 we can see the latent space of the autoencoder from two different perspectives trained with the ADAM optimizer and in figure 2.9 with the AMSGrad optimizer. Furthermore, in figure 2.10 we can see ten reconstructions for each of the ten digits. The training progress for the ADAM optimizer is illustrated in figure 2.11 and for the AMSGrad optimizer in figure 2.12, respectively.

As we saw in figure 2.10, the linear autoencoder does produce recognisable digits in its reconstructions, but it still performs quite poorly. To address this issue, we propose another architecture of an autoencoding neural network. In this setting, we now consider convolutional layers instead of linear layers, this means that the connections between each layer are no longer

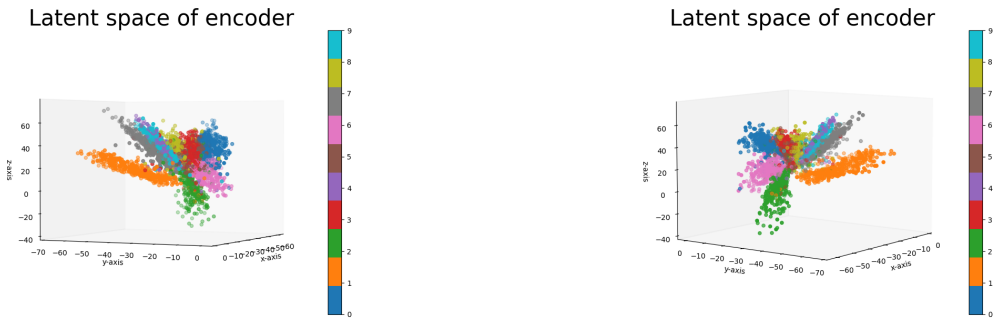


Figure 2.8: The figure illustrates the latent space of the linear autoencoder with bottleneck $n_b = 3$ optimized with an ADAM optimizer from two different perspectives. Each dot is one encoded image of a digit. The color and the corresponding color map represent the digit that was encoded.

Figure 2.9: The figure illustrates the latent space of the linear autoencoder with bottleneck $n_b = 3$ optimized with an AMSGrad optimizer from two different perspectives. Each dot is one encoded image of a digit. The color and the corresponding color map represent the digit that was encoded.

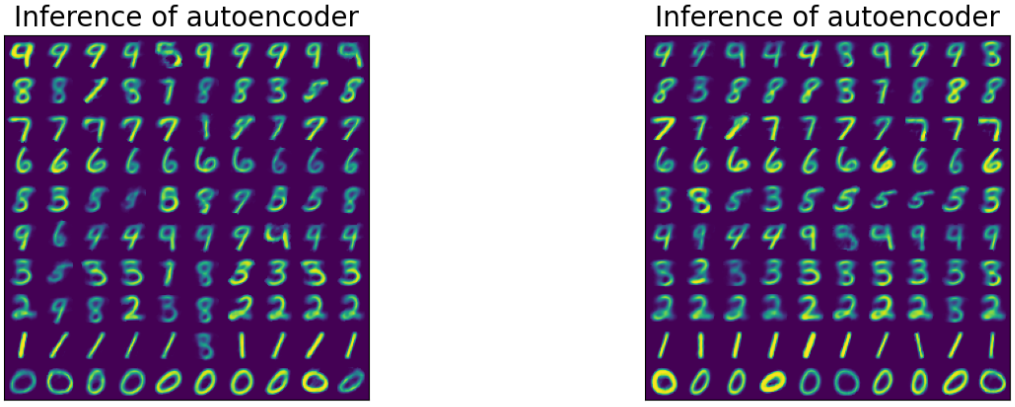


Figure 2.10: The figure illustrates the inference of the linear autoencoder with bottleneck $n_b = 3$ optimized with an ADAM optimizer (left) and an AMSGrad optimizer (right). The inference are ten reconstructed images for each of the ten digits

.

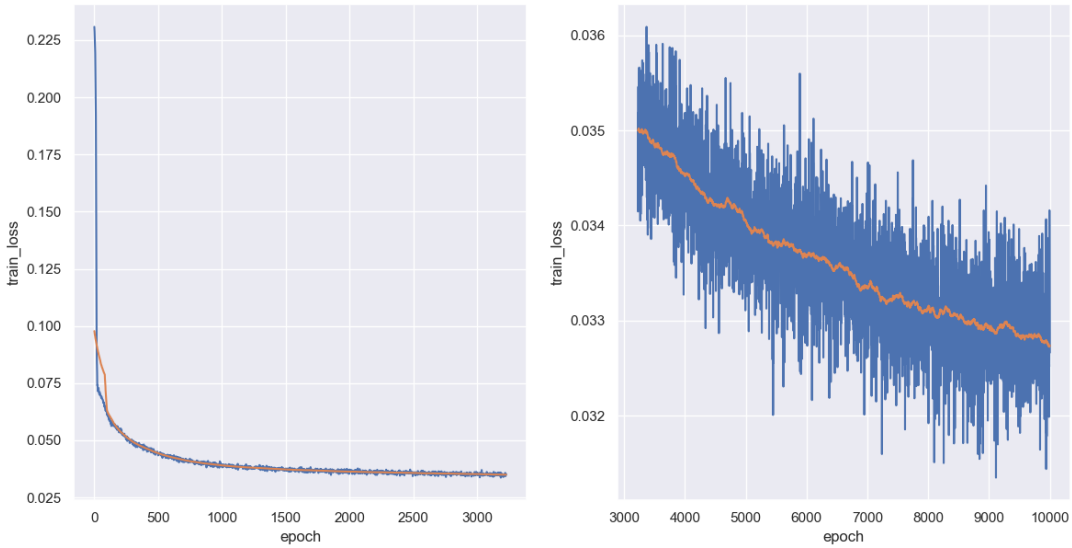


Figure 2.11: The figure illustrates the training progresses of the linear autoencoder with bottleneck $n_b = 3$ optimized with an ADAM optimizer with epochs on one axis and corresponding training loss on the other axis.On the left side we see the first 3.500 epochs and on the right side the following epochs until 10.000. The blue line represents the loss in each epoch and the orange line represents the moving average over 100 epochs to point out the trend of the training progress.

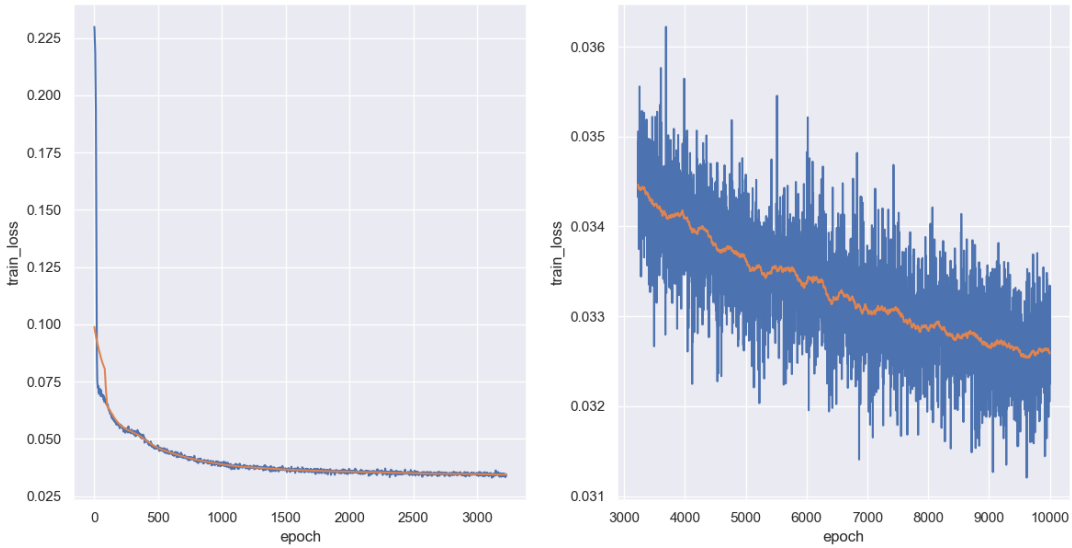


Figure 2.12: The figure illustrates the training progresses of the linear autoencoder with bottleneck $n_b = 3$ optimized with an AMSGrad optimizer with epochs on one axis and corresponding training loss on the other axis.On the left side we see the first 3.500 epochs and on the right side the following epochs until 10.000. The blue line represents the loss in each epoch and the orange line represents the moving average over 100 epochs to point out the trend of the training progress.

matrix multiplications, but much more convolutions. Hence, we introduce the convolutional encoder and the convolutional decoder.

Definition 2.3.5. Let Θ be an arbitrary parameter space, $L \in \mathbb{N}$ and $d_1, \dots, d_L \in \mathbb{N}$. Furthermore, let φ be an arbitrary activation function.
Then an encoding neural network, where each layer $H_1, \dots, H_L$ is defined as

$$H_i(x) = \widehat{\varphi}(T_{\text{conv},i}x + b_i), \qquad x \in \mathbb{R}^{d_i}, i \in \{1, \dots, L\},$$

where $\theta_i = (T_{\text{conv},i}, b_i) \in \Theta$ are the parameters of the i-th layer, see proposition 1.3.14. Such an encoding neural network is called a **convolutional encoder**.

Analogously, we define a convolutional decoding neural network.

Definition 2.3.6. Let Θ be an arbitrary parameter space, $L \in \mathbb{N}$ and $d_1, \dots, d_L \in \mathbb{N}$. Furthermore, let φ be an arbitrary activation function.
Then a decoding neural network, where each layer $H_1, \dots, H_L$ is defined as

$$H_i(x) = \widehat{\varphi}(T_{\text{conv},i}x + b_i), \qquad x \in \mathbb{R}^{d_i}, i \in \{1, \dots, L\},$$

where $\theta_i = (T_{\text{conv},i}, b_i) \in \Theta$ are the parameters of the i-th layer, see proposition 1.3.14 Such a decoding neural network is called a **convolutional decoder**.

Analogously to the linear autoencoder, we can define the convolutional autoencoder with the convolutional encoder and decoder architecture.

Definition 2.3.7. Let f_e and f_d be a convolutional encoder and a convolutional decoder. Then a **convolutional autoencoder** f_{conv} is defined as the composition

$$f_{\text{conv}} \coloneqq f_d \circ f_e.$$

Now, lets define a specific example of a convolutional autoencoder and see how it performs.

Algorithm 5 Convolutional Autoencoder

Let the input and output dimensions be $n_i, n_o \in \mathbb{N}^{2\times 1}$. Furthermore, let the convolutional encoder and the convolutional decoder have k hidden linear layers with dimensions $n_1, n_2, \dots, n_k \in \mathbb{N}^{2\times 1}$ with bottleneck $n_b \in \mathbb{N}^{2\times 1}$. Here, $n_{b;1,2} \in \mathbb{N}^2$ represents the resolution and $n_{b,3} \in \mathbb{N}$ represents the number of channels in the corresponding layer, respectively.
Let the chosen optimizer be AMSGrad with a learning rate $\gamma = 3 \times 10^{-4}$ and the MSE loss function. We will perform the training on 10.000 epochs.

Require: $\gamma \leftarrow 3 \times 10^{-4}$

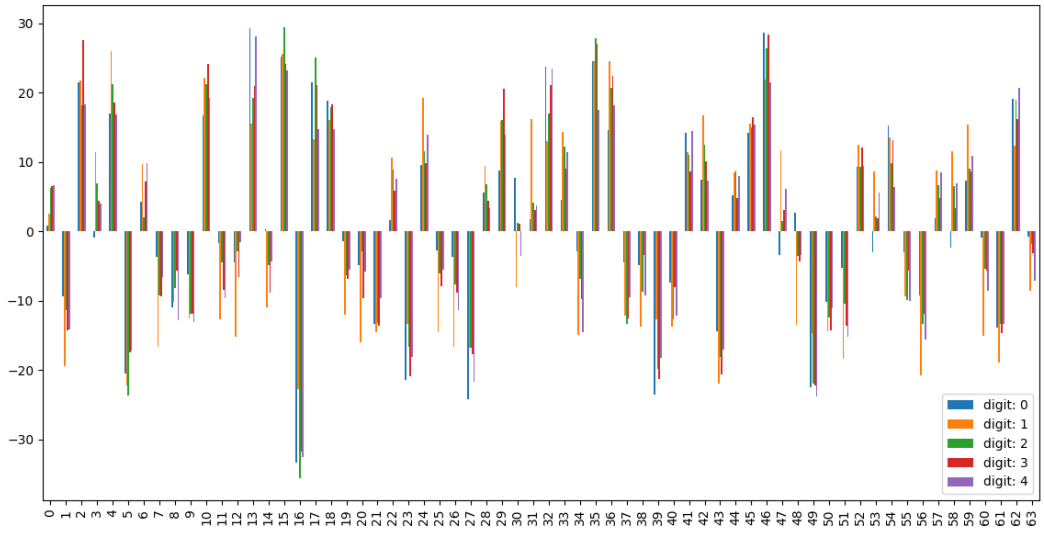
for epoch in $\{1, \dots, 10.000\}$ **do**
  for image in batch **do**
    encoded = encoder(image) ▷ Encode the image onto latent space.
    reconstructed = decoder(encoded) ▷ Decode the encoded image.
    loss = MSE(reconstructed, image) ▷ Compare the output to the input.
    optimization(loss, γ) ▷ Perform an optimization step.
  end for
end for

However, in contrast to the linear autoencoder, the convolutional autoencoder architecture does not allow us to visualise the latent space as easily. The reason is that we designed our two

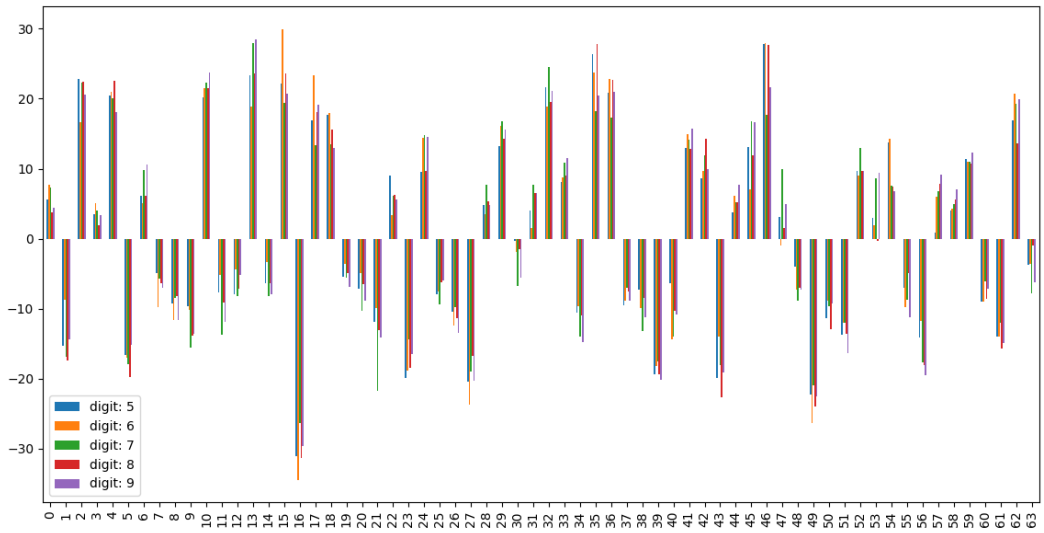
linear autoencoders, each with distinct bottleneck dimensions of 2 and 3, respectively. This choice allowed us straightforward visualization of the latent space, given the single channel nature of our data, which was not altered in the course of computation. Unfortunately, the convolutional neural network does alter the amount of channels in the course of computation, such that we encode the data onto a resolution of one single pixel that comprises of 64 channels and therefore, we would have to visualise 64 channels.
We still came up with an idea of how to visualise the encoded representation, where we plot each channel in a separate bar in a bar plot, see figure 2.13. It is to be read in the following way: each digit receives an own representation in the latent space, that can be uniquely described by the composition of activations in each channel. Here, by activation we mean the value that each channel has. We see that most of the channels have quite different average values and so can be distinguished in that way.
However, what is highly interesting to highlight here is that although the magnitude of the values might differ quite significantly, their signs match most of the time.

Another interesting result that we want to highlight here is the inference of the convolutional autoencoder, which is displayed in figure 2.14. All reconstructions are sharp enough to be recognised as their original digit, which is a huge improvement compared to the inference of the linear autoencoder, see e.g. figure 2.10.

Lastly, we want to take a quick look at the training progress of the convolutional autoencoder, see figure 2.15.



This figure illustrates the average activations of each of the 64 channels in the latent space of the convolutional autoencoder for the digits $0, 1, 2, 3$ and 4. The average has been taken over 5.000 different images of the same digit.



This figure illustrates the average activations of each of the 64 channels in the latent space of the convolutional autoencoder for the digits $5, 6, 7, 8$ and 9. The average has been taken over 5.000 different images of the same digit.

Figure 2.13: The figure illustrates the activations of all 64 channels in the latent space of our trained convolutional autoencoder, which was optimized with an AMSGrad optimizer, see algorithm 5.

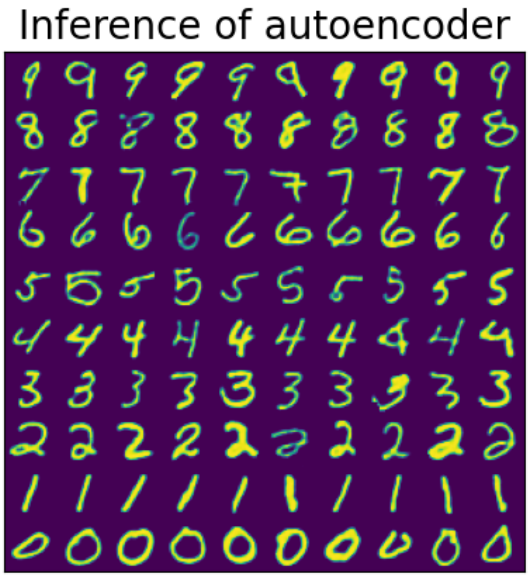


Figure 2.14: The figure illustrates the inference of the convolutional autoencoder, optimized with the AMSGrad optimizer. The inference consists of the reconstructed images for each of the ten digits.

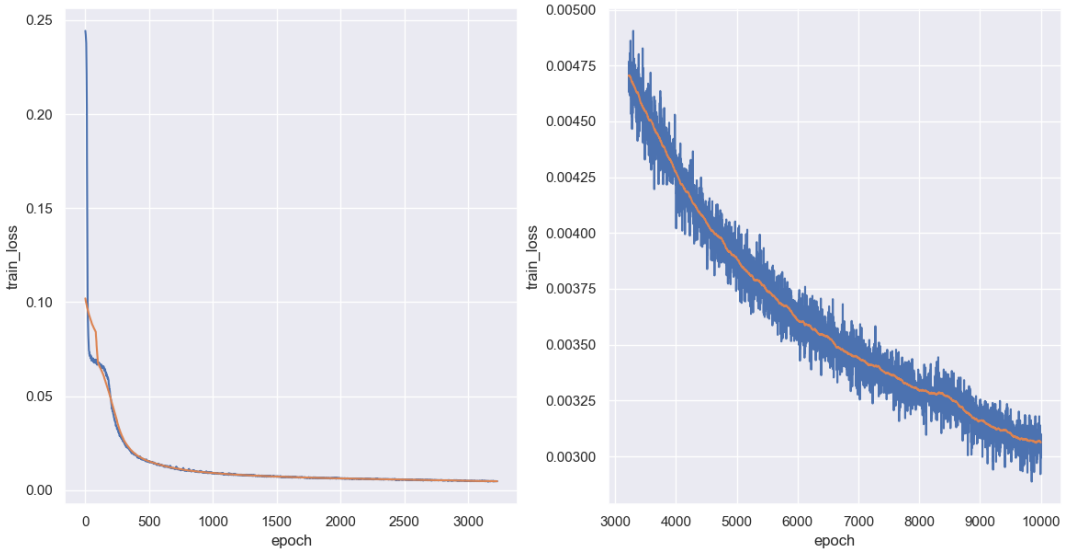


Figure 2.15: The figure illustrates the training progresses of the convolutional autoencoder optimized with an AMSGrad optimizer with epochs on one axis and corresponding training loss on the other axis. On the left side we see the first 3.500 epochs and on the right side the following epochs until 10.000. The blue line represents the loss in each epoch and the orange line every 300 epochs to point out the trend of the training progress.

3 Variational Autoencoders

Differently from ordinary autoencoding neural networks that we already mentioned to be discriminative models, variational autoencoding neural networks on the other hand are so called generative models, see [10, chapter 5]. This means that instead of trying to estimate the conditional distribution of $y|x$, where y is a predicted label to an observation x, variational autoencoders attempt to capture the entire probability distribution of Y. This is very interesting for multiple reasons, since this means that we can simulate and anticipate the evolution of the model output. Hence, we could generate new data based on the captured probability distribution. This will be our ultimate goal in this chapter, considering applications at last.

3.1 Probabilistic foundations

Before diving into the depths of variational autoencoders, we want to begin by laying the essential probabilistic foundations. As already mentioned, in contrast to ordinary autoencoding neural networks which we discussed in chapter 2 variational autoencoding neural networks attempt to capture the entire probability distribution of Y. So the emerging question is how can we infer the probability distribution of Y, if we only have the observations x given? We assume that the single data points x_i are not independent - what is no grave assumption, since the observations are assumed to have a reason to be shaped the way they are (e.g. having the same underlying probability distribution). This reason, since it is unknown or *latent* to us, we will try to consider in more detail. In order to model this hidden cause, we introduce another random variable Z, a so called *latent variable* and thus by marginalizing over z obtain the joint distribution $p(x, z)$

$$p(x) = \int p(x, z)dz = \int p(x|Z)p(z)dz. \tag{3.1}$$

This probability distribution we now want to introduce formally, since they will be of crucial importance in this chapter. However, we first need to introduce the latent space and the observation space.

Definition 3.1.1. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space and let $\mathcal{X}$ as well as $\mathcal{Z}$ be measurable spaces. Furthermore, let X be a random variable defined on $\mathcal{X}$ and Z be a random variable defined on $\mathcal{Z}$, respectively.
We will call $\mathcal{X}$ the **observation space**, X the **observation random variable** as well as $\mathcal{Z}$ the **latent space** and Z the **latent random variable** in the following.

Considering the marginal distributions of the joint distribution, we can introduce the model evidence and the model prior, where the first refers to the observable data and the second refers to our prior belief in what the latent space might look like.

Definition 3.1.2. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space and X, Z be observation and latent random variables, respectively. Therefore, $p(x, z)$ describes the joint distribution of X and Z. Then we call the marginal distribution

$$p(x) = \int p(x, z)dz = \int p(x|Z)p(z)dz,$$

the **model evidence** (sometimes referred to as **marginal likelihood**).
Moreover, we call the marginal distribution

$$p(z) = \int p(x, z)$$

the **model prior**.

However, since our ultimate goal is to find a probability distribution that describes our observations as good as possible, we need to somehow define an update rule, how we want to adapt our current model of the latent space with respect to newly observed data. In order to do this, we need to introduce two more quantities. The first one is the so called likelihood. Conceptionally, it is like a probability density function, where we fix the observation and let the parameter be a variable. The second quantity is called the posterior - as the name already suggests, this describes our understanding of the latent space after considering newly observed data.

Definition 3.1.3. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space, $(\mathcal{X}, \mathcal{A})$ be a measurable space on which we define a random variable X and let Θ be a parameter space. Furthermore, let X have a probability distribution with parameters $\theta \in \Theta$. Hence, we describe the probability density function of X as $p(x; \theta)$.
Then we define the **likelihood function** as

$$L(\theta|x) = p(x; \theta),$$

where $x \in \Omega$ is fixed and $\theta \in \Theta$ is variable.

Definition 3.1.4. Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space, $(\mathcal{X}, \mathcal{A})$ be a measurable space on which we define a random variable X and let Θ be a parameter space. Furthermore, let X have a probability distribution with parameters $\theta \in \Theta$. Hence, we describe the probability density function of X as $p(x; \theta)$.
Then we define the **model posterior** as $p(\theta|X)$.

Remark 3.1.5. Since it is common to make no distinction between random variables and parameters in a Bayesian approach, we can consider the model parameters $\theta \in \Theta$ as part of the latent space. Hence, we can describe the likelihood function as $p(x|Z)$.
Moreover, with the same argument we can consider the model posterior as $p(z|X)$.

FROM HERE ON THE TEXT IS NOT REWORKED YET

Firstly, we want to define the specific structure of variational autoencoding neural networks and their key differences to ordinary autoencoding neural networks. As already motivated,

variational autoencoders are generative models. This means, that they intend to capture the entire probability distribution of the observation space Y. Hence, they are statistical models and depend on certain probabilistic approaches.

In order to compute the gradient of a variational autoencoding neural network, we need to determine a way to quantify the distance between probability distributions. A popular measure for this case is the so called Kullback-Leibler divergence. It assesses the dissimilarity between two probability distributions over the same random variable X.

Definition 3.1.6. Let X be a random variable and p, q two probability density functions over X. Then, the **Kullback-Leibler divergence** is defined as

$$D_{\mathrm{KL}}(p \parallel q) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx. \tag{3.2}$$

If we consider the Kullback-Leibler divergence with regard to a dataset consisting of observed data samples, we can write equivalently.

Definition 3.1.7. Let X be a random variable and p, q two probability density functions over X. Furthermore, let $D = \{x_i\}_{i=i}^{L}$ be an unsupervised dataset of length $L \in \mathbb{N}$.
Then, the **Kullback-Leibler divergence** is defined as

$$D_{\mathrm{KL}}(p \parallel q) = \sum_{i=1}^{L} p(x_i) \log \left(\frac{p(x_i)}{q(x_i)} \right). \tag{3.3}$$

Definition 3.1.8. Let X be a random variable and p a probability density function over X. Then, the nonnegative measure of the expected information content of X under correct distribution p, defined as

$$\mathcal{H}(X) = -\int p(x) \log p(x) dx = \mathbb{E}_p \left[-\log p(x) \right], \tag{3.4}$$

is called **entropy** of p.

Definition 3.1.9. Let X be a random variable and p, q two probability density functions over X. Then, the nonnegative measure of the expected information content of X under incorrect distribution q, defined as

$$\mathcal{H}_q(X) = -\int p(x) \log q(x) dx = \mathbb{E}_p \left[-\log q(x) \right], \tag{3.5}$$

is called **cross-entropy** between p and q.

Remark 3.1.10. Let X be a random variable and p, q probability density functions over X. Then the Kullback-Leibler divergence between p and q can be written as follows

$$D_{\mathrm{KL}}(p \parallel q) = \mathcal{H}_q(p) - \mathcal{H}(p),$$

where $\mathcal{H}_q$ and $\mathcal{H}$ denote the cross-entropy and entropy, respectively.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[2] C. Lemaréchal, "Cauchy and the gradient method," *Doc Math Extra*, vol. 251, no. 254, p. 10, 2012.

[3] L. V. Kantorovich and G. P. Akilov, *Functional analysis*. Elsevier, 2016.

[4] Y. Lei, T. Hu, and K. Tang, "Stochastic gradient descent for nonconvex learning without bounded gradient assumptions," *CoRR*, vol. abs/1902.00908, 2019.

[5] F. Edition, A. Papoulis, and S. U. Pillai, "Probability, random variables, and stochastic processes," 2002.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[7] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019.

[8] H. B. McMahan and M. Streeter, "Adaptive bound optimization for online convex optimization," *arXiv preprint arXiv:1002.4908*, 2010.

[9] A. Klenke, *Probability theory: a comprehensive course*. Springer Science & Business Media, 2013.

[10] L. P. Cinelli, M. A. Marins, E. A. B. Da Silva, and S. L. Netto, *Variational methods for machine learning with applications to deep networks*. Springer, 2021.

[11] D. Foster, *Generative deep learning*. " O'Reilly Media, Inc.", 2022.

Stuttgart, October 11, 2023