UNIVERSITAT DE BARCELONA

A Mathematical Introduction to Neural Networks

Aisha Lichtner-Bajjaoui

Adviser: Josep Vives

Advanced Mathematics Master Program of the Universitat de Barcelona
Barcelona, January 12, 2020

Contents

Chapter 1

Introduction

In this work, we are going to introduce Neural Networks. First, we are going to give a mathematical formulation of the concept of Neural Networks. Later on, we will examine some important properties of Neural Networks and make a connection to common statistical methods such as Principal Component Analysis and Singular Value Decomposition. In the last chapter, we will give a practical application of a neural network for a regression problem. The concept of a Neural Network is inspired by the activities of a human brain. Neurons receive information, if the information is relevant to the neuron, a signal is sent to other neurons via synapses. The main difference between Neural Networks and rule-based statistical methods, is the learning ability of Neural Networks. At the beginning of a training phase a network has no explicit information. During the training phase the inter-neural connections are changed in a way that the network solves the given problem best. Therefore Neural Networks can provide solutions to a wide spectrum of problems.

A Neural Network is an abstract model consisting of one or more layers, that are connected in a certain way. The weighted connection between the layers plays the role of synapses. Each layer consists of units modelling neurons in the human brain. The units carry activation functions, modelling the impulses, that the real neurons send when being triggered. Just like the brain, the network will be trained to learn a specific task and later should perform a similar task in an unknown situation, using the experience that it gained before. For that matter, during the training phase already-observed information is passed to the model and the model produces an output. The output is evaluated on its ability to approximate the observed information. Depending on the result, the model is then changed to improve performance.

1.1 History of Neural Networks

The idea of an artificial Neural Networks goes back to 1940 when Walter Pitts and Warren McCulloch discovered that neurons in the human brain essentially perform combinations of logical operations and have binary outputs that depend on a specific threshold: active or not active. Based on that fact, mathematical models were built and created great interest in the Artificial Intelligence community. However, at that

time, computers had just been invented and were not close to being able to handle algorithms of such complexity. In the following years, scientist lost interest in Neural Networks due to lack of progress in the field and other, at the time, more promising methods. In 1970 Seppo Linnainmaa discovered Backpropagation, that later should revolutionize the performance of Neural Networks. Still, Neural Networks were not in focus of the Artificial Intelligence Community until 2012, when students won the ImageNet Large Scale Visual Recognition Competition with remarkable results. Since then Neural Networks are an uprising topic in Artificial Intelligence.

1.2 Basic Results of Linear Algebra and Analysis

Definition 1.2.1. A quadratic $(n \times n)$-matrix A $\in \mathbb{R}^{n \times n}$ is called

- positive definite if $\langle x, Ax \rangle = x^T Ax > 0$ for all $x \in \mathbb{R}^n$ with $x \neq 0$
- positive semidefinite if $\langle x, Ax \rangle = x^T Ax \geq 0$ for all $x \in \mathbb{R}^n$ with $x \neq 0$

Remark 1.2.2. Note that for every $(n \times p)$- matrix A the matrices $A^T A \in \mathbb{R}^{p \times p}$ and $AA^T \in \mathbb{R}^{n \times n}$ are symmetric and positive semidefinite by bilinearity of the scalar product

$$\langle x, A^T Ax \rangle = \langle Ax, Ax \rangle = ||Ax||^2 \geq 0$$

and

$$\langle x, AA^T x \rangle = \langle A^T x, A^T x \rangle \geq 0$$

Lemma 1.2.3. *Let $A \in \mathbb{R}^{n \times p}$ matrix. A map f defined by*

$$f(x) = Mx$$

is

1. *injective, if and only if A has full column rank p.*
2. *surjective if and only if A has full row rank n.*

Definition 1.2.4. Let A be an $(m \times n)$- matrix with column vectors $a_1, ..., a_n$ and B a $p \times q$ matrix, then the **Kronecker product**

$$A \otimes B$$

is a $mp \times nq$ matrix M defined by

$$m_{ij} = a_{ij} B$$

Definition 1.2.5. Let A be an $(m \times n)$- matrix with column vectors $a_1, ..., a_n$. The transformation of matrix A into a column vector by

$$P := (a_1^T, ..., a_n^T)^T$$

is denoted by

$$vec(A) := P$$

Definition 1.2.6. Let M be a quadratic $(n \times n)$-matrix, then the **trace of M** is defined by

$$tr(M) = \sum_{i=1}^{n} a_{ii}$$

Lemma 1.2.7. *Let M be a $(n \times m)$-matrix , N be a $(m \times n)$-matrix, Q be a quadratic $(n \times n)$- matrix and U be an orthogonal $(n \times n)$-matrix, then it states that,*

1. $$tr(MN) = tr((MN)^T)$$

2. $$tr(UQU^T) = tr(Q)$$

Lemma 1.2.8. *Let M be a symmetric $(n \times m)$ -dimensional positive semidefinite matrix of full column rank $m \leq n$. Then M is positive definite.*

Theorem 1.2.9. *Spectral Theorem*
Let $M \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then there exist a diagonal matrix $D \in \mathbb{R}^{n \times n}$ and an orthogonal matrix $U \in \mathbb{R}^{n \times n}$ such that $M = UDU^T$. The diagonal entries of D are the eigenvalues of M and the columns of U are the corresponding eigenvectors.

Theorem 1.2.10. *Let M be a $n \times m$ matrix, c any n-dimensional vector and $F : \mathbb{R}^m \to \mathbb{R}$ a quadratic function given by $F(z) = ||c - Mz||^2$. It states that:*

1. *F is convex*
2. *∇ F(z)= 0 if and only if z corresponds to a global minimum*
3. *If in addition $M^T M$ is positive definite, then F is strictly convex and has a unique minimum in $z = (M^T M)^{-1} M^T c$*

Remark 1.2.11. Observe that

$$\begin{aligned} ||c - Mz||^2 &=< c - MZ, c - Mz >=< c, c > -2 < c, MZ > + < Mz, Mz > \\ &= c^T c - 2c^T Mz + z^T M^T Mz \end{aligned}$$

and therefore

$$F'(z) = 2M^T Mz - 2M^T c$$

so with the previous Theorem we can conclude

Corollary 1.2.12. *A minimum is reached when M from previous Theorem satisfies*

$$M^T Mz = M^T c \tag{1.2.1}$$

Lemma 1.2.13. *For any matrices P, Q and R (of dimensions that allow multiplications) it states:*

1. $tr(PQ^T) = (vecP)^T vec(Q)$
2. $vec(PQR^T) = (R \otimes P)vec(Q)$
3. $(P \otimes Q)(R \otimes S) = PR \otimes QS$
4. $(P \otimes Q)^{-1} = P^{-1} \otimes Q^{-1}$
5. $(P \otimes Q)^T = P^T \otimes Q^T$
6. *If additionally P and Q are symmetric and positive semidefinite, then $(P \otimes Q)$ is symmetric and positive semidefinite.*
7. *If additionally P and Q are symmetric and positive definite, then $(P \otimes Q)$ is symmetric and positive definite.*

Proof. see [1], page 167-190. □

Chapter 2

Mathematical definition of a Neural Network

In this chapter, we will introduce Neural Networks in the words of Statistics. The general idea of a Neural Network is to find a model that, based on a set of N samples

$$D = \{[x_1, ..x_N], [y_1, .., y_N]\} \tag{2.0.1}$$

will approximate a unknown function f, with

$$f(x_i) = y_i$$

as well as possible. The model always consists of one input layer, an output layer and at least one hidden layer. The layers are connected with weighted transitions, that can be modelled by a product of a weight matrix with a vector, that stands for the output of the previous layer. Each layer consists of several units, the neurons. The amount of units in each layer defines the dimension of the layer. The activation functions in the neurons are essentially modelling a threshold that decides whether the information that a neuron got will be relevant for the further calculations or not. If the information is relevant it will be passed on to the next layer until the output layer is reached. During the so-called training phase, the model is given a sample data set D where the matrix $[x_1, ..., x_N]$ is the input and $[y_1, ...y_N]$ the output matrix. After processing each sample $(x_i, y_i) \in D$, the distance of the output that the network produces to the actual desired output y_i, is measured by some loss function, for example, the $\mathcal{L}^2$-norm or the euclidean norm. According to the results, the weights of the transitions are modified to achieve a better result in the next iteration. The dimension of the input layer is determined by the dimension of the input data, while the dimension of the output corresponds to the problem we want to solve using the model. For example, if we are trying to use the model for regression, the output layer will be of dimension one, because we want the model to assign one value to each input. For a classification problem, we use an output layer of dimension k, where k describes the k different categories we want to sort the data in.

For a sufficiently big sample, the model is expected to produce a good approximation of the desired output. In the following, we will try to find the best estimator $\widehat{F}$ of f that satisfies

$$\widehat{F}(x_i) = y_i.$$

That means the best approximation of f for given observation D. Then we will apply $\widehat{F}$ to unknown values z to make a prediction or classification.

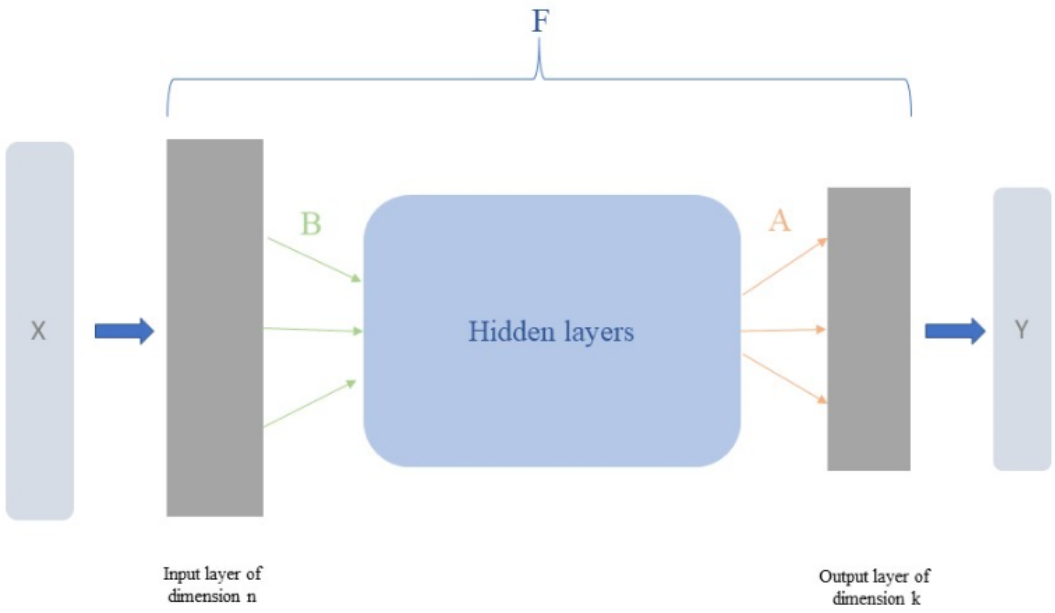


Figure 2.1: Neural Network with several hidden layers

Definition 2.0.1. Let $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^k$ for all $i \leq N \in \mathbb{N}$. Define

$$D := \{[x_1, ..x_N], [y_1, .., y_N]\}$$

the **sample** of size N. Let

$$X := [x_1, ..x_N]$$

denote an $n \times N$ matrix that consists of the **training input vectors** x_i in a sample D of size N. In the following we will call D the **training set**.

$$Y := [y_1, .., y_N]$$

will model the corresponding observed **training output vectors** y_i and form a $k \times N$-matrix.

Definition 2.0.2. By $Z := [z_1, ..z_M]$ we denote the **test set** with **test vectors** $z_i \in \mathbb{R}^n$.

Definition 2.0.3. Let $\tilde{X}$ be a n- dimensional random vector that models the random choice of training input vectors x_i. The random vector Z describes the choice of test data. By $\tilde{Y}$ we will denote a k-dimensional random vector that models the output of the network for all possible inputs.

Definition 2.0.4. A function $F : \mathbb{R}^n \to \mathbb{R}^k$ represents the **network activity** and maps input vectors to output vectors.

The function F satisfies

$$F(x_i) + \epsilon_i = y_i,$$

where ϵ_i is a random variable with unknown distribution P_ϵ that describes the **noise** that is produced by the unknown values Z.

Our goal is to train the network, by changing the function F so it approximates f as good as possible by minimizing a **risk function** R that depends on the choice of a model F. It is calculated by integrating over a **loss function** L, that measures the approximation error

$$R(F) = \int L(\tilde{Y}, F(\tilde{X}))dP(\tilde{X}, \tilde{Y}), \tag{2.0.2}$$

where P denotes the unknown joint distribution of $(\tilde{X}, \tilde{Y})$.

Since we cannot calculate the integral because we are missing knowledge about the measure P, we will instead try to minimize the average loss in the sample D, for given observations (x_i, y_i)

$$R_E(F) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, F(x_i)).$$

We can understand, however, that an optimal solution to (2.0.2) must satisfy

$$F(x_i) = E(\tilde{Y}|x_i) \tag{2.0.3}$$

That is the expected value of $\tilde{Y}$, if we already know the output of x_i.

To make that connection more understandable we will look at the following example.

Example 2.0.5. Let the loss function L be the quadratic error that is

$$L(y, F(x)) = (y - F(x))^2.$$

Calculating the risk for this loss function we obtain

$$\begin{aligned} R(F) &= \int (y - F(x))^2 dP(x,y) \\ &= E[(\tilde{Y} - F(\tilde{X}))^2] \\ &= E[(\tilde{Y} + E(\tilde{Y}|\tilde{X}) - E(\tilde{Y}|\tilde{X}) - F(\tilde{X}))^2] \\ &= E[(\tilde{Y} - E(\tilde{Y}|\tilde{X}))^2] + E[(E(\tilde{Y}|\tilde{X}) - F(\tilde{X}))^2] \\ &+ 2E[(\tilde{Y} - E(\tilde{Y}|\tilde{X}))(E(\tilde{Y}|\tilde{X}) - F(\tilde{X}))]. \end{aligned}$$

Notice that since $E(\tilde{Y}|\tilde{X})$ is a projection of $\tilde{Y}$ on the sub-σ-field generated by $\tilde{X}$ it follows that $(\tilde{Y} - E(\tilde{Y}|\tilde{X}))$ is orthogonal to $E(\tilde{Y}|\tilde{X})$. We can conclude therefore, by

$$R(F) = E[(\tilde{Y} - E(\tilde{Y}|\tilde{X}))^2] + E[(E(\tilde{Y}|\tilde{X}) - F(\tilde{X}))^2],$$

that the best choice of F to minimize R is the conditional expectation

$$F(x) = E(\tilde{Y}|x)$$

Definition 2.0.6. $\widehat{F} : \mathbb{R}^n \to \mathbb{R}^k$ denotes a estimator that satisfies

$$\widehat{F}_D = \arg\min_F R(F) \tag{2.0.4}$$

for a given sample D.

Observe that we do not always know if the optimal estimator $E(\tilde{Y}|x)$ is contained in the space of functions that the network can implement or that satisfy given conditions on the approximating function, such as, for example, being continuous. We call the difference between $E(\tilde{Y}|x)$ and $\widehat{F}$ the **error**.

2.1 Bias and Variance

We are considering the estimator $\widehat{F}_D$ for a given sample D and the quadratic loss function. Taking in account the previous observation (2.0.3), the expected quadratic error for an observation x, given D, is

$$E[(E(Y|x) - \widehat{F}_D(x))^2|D]. \tag{2.1.1}$$

In order not to have a result that measures the accuracy of the approximation only for a specific sample D, we consider the expectation E_D of $(E(\tilde{Y}|x) - \widehat{F}_D(x))^2$, that is the expectation with respect to all possible sample sets D, hence the average over

all possible samples of size N. We observe

$$\begin{aligned}
&E_D[(E(\tilde{Y}|x) - \widehat{F}_D(x))^2] \\
&= E_D[((\widehat{F}_D(x) - E_D[\widehat{F}_D(x)]) + (E_D(\widehat{F}_D(x) - E(\tilde{Y}|x))^2] \\
&= E_D[(\widehat{F}_D(x) - E_D(\widehat{F}_D(x))^2] + E_D[(E_D[\widehat{F}_D(x)] - E(\tilde{Y}|x))^2] \\
&+ 2E_D[(\widehat{F}_D(x) - E_D(\widehat{F}_D(x))](E_D[\widehat{F}_D(x)] - E(\tilde{Y}|x)) \\
&= E_D[(\widehat{F}_D(x) - E_D(\widehat{F}_D(x))^2] + (E_D[\widehat{F}_D(x)] - E(\tilde{Y}|x))^2 \\
&+ 2E_D[(\widehat{F}_D(x) - E_D(\widehat{F}_D(x))](E_D[\widehat{F}_D(x)] - E(\tilde{Y}|x)) \\
&= \underbrace{(E_D[\widehat{F}_D(x)] - E(\tilde{Y}|x))^2}_{\text{bias}} + \underbrace{E_D[(E_D[\widehat{F}_D(x)] - \widehat{F}_D(x))^2]}_{\textbf{variance}}
\end{aligned}$$

If $\widehat{F}_D$ on average differs from $E(\tilde{Y}|\cdot)$, that means $(E_D[\widehat{F}_D(x)] - E(\tilde{Y}|x))^2$ is big, we call $\widehat{F}_D$ **biased**. If we found a non-biased $\widehat{F}_D$, the variance can still be a cause of poor performance. For example in the optimal case , $\widehat{F}_D(x) = E(\tilde{Y}|x)$, we obtain the variance

$$E_D[(E_D[E(\tilde{Y}|x)] - E(\tilde{Y}|x))^2]$$

that can have a big impact, depending on how much $E(\tilde{Y}|\cdot)$ varies for the given data.

By that, we can see that,on one hand, a function $\widehat{F}$ can approximate f very well for a specific training sample, but on the other hand, applying the approximation to other samples the result might not be good. This problem is called **underfitting**, the network has, due to a bad choice of training samples, not learned enough about the properties of the function f it approximates. On the other hand, F might be an unbiased approximation that is very sensitive to the given data, thus on average approximates $E(\tilde{Y}|\cdot)$ well, but still causes poor performance. We call this behaviour of a network **overfitting**, that means, the network models the noise in the training sample.

We can observe that there is a dilemma between reducing the bias and reducing the variance. By increasing the dimension of the training samples, that can be done by adding a layer of a higher dimension, that has the effect of giving the training samples more features and therefore more information, the bias can be reduced. But by giving more information the variance will be in general increased. This creates a trade-off between variance and bias.

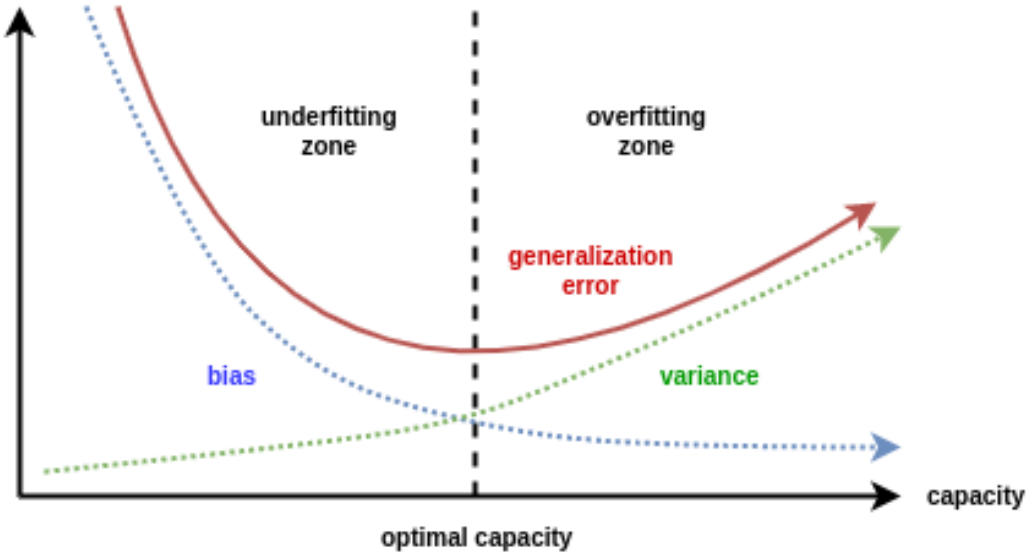


Figure 2.2: Bias and Variance

2.2 The Neurons

In this section, we will take a closer look at the activities in the neurons. Each neuron receives a signal $s = (s_1, ..., s_m)$ that consist of, depending on the architecture of the network, $m \in \mathbb{N}$ scalar inputs $s_1, ..., s_m$ and a vector that models the weights w of the transition from the previous neuron towards the neuron. A function h is then acting on the inputs, producing one output $h(s_1, ..., s_m)$ that is passed on to an activation function g and generates the one-dimensional output $g(h(s_1, ..., s_m))$.

There are different ways in which the neurons process the inputs. Essentially we can distinguish between two types of neurons:

1. **Neurons that perform a scalar product**
   A way to handle the inputs is to build the scalar product of the inputs and the vector w of weights. Hence the activity of the neuron can be described by

   $$h(s) = \langle w, s \rangle$$

   An example of common choices for the function g is a sigmoid function $g(x) = \frac{1}{1+e^{-t}}$, the identity or a sign function.

2. **Neurons that calculate a distance**
   Alternatively, the Neurons can also calculate the distance between an input signal s and w. Hence

   $$h(s) = ||w - s||$$

   for some norm $|| \cdot ||$.

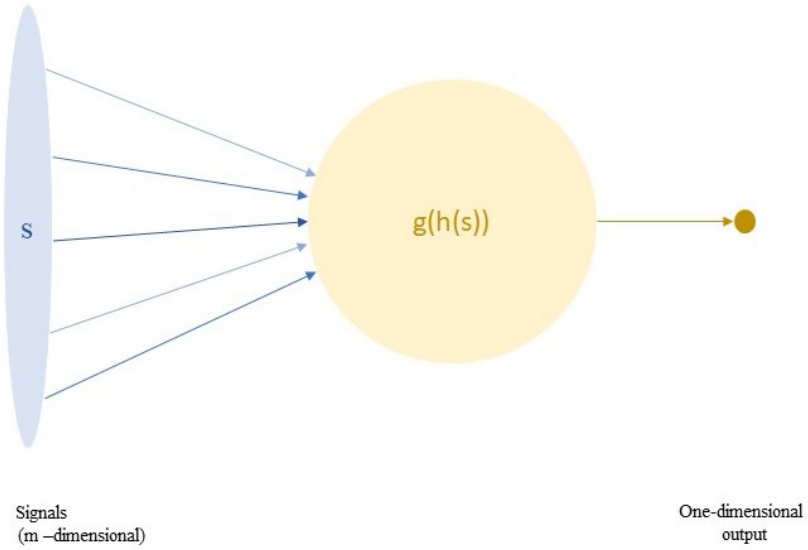


Figure 2.3: Neuron activity

2.3 Multilayer Perceptron

The activity of a network is mainly characterised by the choice of activation function and weights connecting the layers, but also on the choice of a specific architecture. A **network architecture** is a choice of the number of hidden layers, the size of the layers, meaning the number of neurons and the way the neurons are connected. In this work, we will only examine Multilayer Perceptrons.

Definition 2.3.1. A **Multilayer Perceptron** is a network that consists of one input layer, one output layer and one or more hidden layers. The neurons in a layer are only connected to neurons in different layers, in a way that they do not build any cycles, hence, are not connected to neurons of the previous layer. The Neurons that are used only perform a scalar product operation and the network activity can be described by a function

$$y = F(x; W)$$

for any input x and output y, where W describes the chosen weights.

Let us take now a closer look at how the Multilayer Perceptron learns.

2.3.1 Training a Multilayer Perceptron

In the **training phase** the Multilayer Perceptron is given a sample D, defined like in (2.0.1). Based on the given information, the network is trying to find a F that is a solution to (2.0.4). Since for a fixed choice of network architecture with quadratic

error function, the results depend only on the choice of weights, that are calculated during the training phase, the problem (2.0.4) can be reduced to

$$W^\star = \arg\min_W \frac{1}{N} \sum_{i=1}^{N} (y_i - F(x_i; W))^2 \tag{2.3.1}$$

W is the matrix, that models all weights of the whole network. The problem (2.3.1) is solved by a gradient descent method, like Backpropagation. The gradient descent methods are based on the idea of minimizing a function by iteratively moving in the direction of the steepest descent, that is the direction in which the gradient of the function points. So for (2.3.1) the t-th iteration step is modelled by

$$W_{k,j}(t) = W_{k,j}(t-1) + \Delta W_{k,j}(t)$$

where the gradient $\Delta W_{k,j}(t)$ is calculated as follows

$$\Delta W_{k,j}(t) = -\epsilon(t) \frac{\partial (y_i - F(x_i; W))}{\partial W_{k,j}}$$

Remark 2.3.2. Notice that another approach would be to build the gradient of the whole expression $\frac{1}{N} \sum_{i=1}^{N} (y_i - F(x_i; W))$. However that would slow down the procedure, since for each iteration step, all samples are considered.

2.3.2 Autoassociative Multilayer Perceptron

An important

of a Multilayer Perceptron is the Autoassociative Multilayer Perceptron, that can be interpreted as learning the identity map $F(x) = x$.
A significant application of that architecture is to understand how to compress data efficiently, hence without losing important information. Since the network is first lowering the dimension of the input and then tries to reproduce it as well as possible, one can understand that in the optimal case, that is in the minimum of R, the data is compressed in the hidden layer in a way that preserves as much information as possible.

Definition 2.3.3. A Multilayer Perceptron where the output of the network equals the given input is called **Autoassociative Multilayer Perceptron**.

Chapter 3

Properties and Connection to Statistical Methods

In this chapter, we will discuss important properties of Multilayer Perceptrons and their connection to already well knows statistical methods. To begin with we will introduce some notations and definitions that will be used in this chapter.

Definition 3.0.1. For an $(n \times m)$-dimensional matrix M. We define the **average vector** by

$$\mu_M := \frac{1}{n} Mv$$

where v denotes a m-dimensional vector of ones.

Definition 3.0.2. For a sample matrix X with column vectors x_i and a sample matrix Y with column vectors y_i for $i \in \{1, .., N\}$, let the $(n \times n)$-dimensional matrices

$$\Sigma_{XX} := \sum_{i=1}^{N} (x_i - \mu_{x_i}) \cdot (x_i - \mu_{x_i})^T$$

$$\Sigma_{XY} := \sum_{i=1}^{N} (x_i - \mu_{x_i}) \cdot (y_i - \mu_{y_i})^T$$

$$\Sigma_{YX} := \sum_{i=1}^{N} (y_i - \mu_{y_i}) \cdot (x_i - \mu_{x_i})^T$$

$$\Sigma_{YY} := \sum_{i=1}^{N} (y_i - \mu_{y_i}) \cdot (y_i - \mu_{y_i})^T$$

define the **sample covariance matrices**. Finally define

$$\Sigma := \Sigma_{YX} {\Sigma_{XX}}^{-1} \Sigma_{XY}$$

Lemma 3.0.3. *Σ_{XX} is positive semidefinite and symmetric.*

Remark 3.0.4. Notice that, for centered samples X and Y, that means $\mu_X = 0$, we can observe that $\Sigma_{XX} = XX^T$, $\Sigma_{YY} = YY^T$, $\Sigma_{XY} = XY^T$ and $\Sigma_{YX} = YX^T$.

Remark 3.0.5. Additionally, we can expect the sample covariance matrix to have only positive eigenvalues, which implies full rank. A rank deficient covariance matrix would indicate a poor choice of training samples since linear dependencies describe correlations in the training set.

Definition 3.0.6. For eigenvalues $\lambda_1 > \lambda_2 > ... > \lambda_n$ of $\sum$ and an index set $I = \{i_1, ..i_p\}$ with $i_1 < i_2 < ... < i_p \leq n$. Define the **matrix of the orthonormal eigenvectors** of $\sum$ associated to the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_p$ by

$$U_I := [u_{i_1}, ..., u_{i_p}].$$

Furthermore, define

$$U_p := [u_1, ..., u_p]$$

as the **matrix of the first p eigenvectors** corresponding to the p biggest eigenvalues $\lambda_1 > ... > \lambda_p$.

Definition 3.0.7. For an $n \times p$ matrix M with $p < n$ we denote the **orthogonal projection** on the subspace spanned by the column vectors of M by P_M.
If M is of full rank p, then

$$P_M = M(M^T M)^{-1} M^T.$$

Let us first see that Multilayer Perceptrons are universal approximators.

Theorem 3.0.8. *If f is a measurable function, N the size of a sample and $\phi(W)$ denotes the space of functions that a Multilayer Perceptron, with one hidden nonlinear layer and weights W, can implement. Then, for any fixed precision value ϵ, there exists an optimal weight matrix W such that, the probability that $F(\cdot, W) \in \phi(W)$ approximates f with precision ϵ, tends to one for $N \to \infty$*

Proof. See [2], page 359-366. □

3.1 Global minimum and Principal Component Analysis

In this section we will take a closer look at the learning capacity of Multilayer Perceptrons using gradient descent methods, discussing, in particular, the problem of local minima. As a result, we will observe that there is a unique global minimum and no local minima. Moreover, we will find a connection to Ordinary Least Square Regression and Principal Component Analysis in the autoassociative case. To begin with, let us introduce Principal Component Analysis and Ordinary Least Square Regression.

3.1.1 Principal Component Analysis

Principal Component Analysis is a method in statistics to represent high dimensional data in simplified terms, meaning in a lower dimension. To do that, losing as little information as possible, we are looking for the **principal components** of the data, that is the components of the specific data set that carry the most information, hence varies the most. The data is then being represented with respect to the principal components. That means the dimension and therefore the complexity is reduced with losing as little information as possible.

Let us put this concept into mathematical terms.

Definition 3.1.1. Let X be a $(n \times n)$-dimensional matrix, with covariance matrix C of rank n. The orthonormal eigenvectors of C are denoted by $u_i = (u_{1i}, ..., u_{ni})$ and the eigenvalues by λ_i and arranged in decreasing order. That is

$$\lambda_1 > \lambda_2 ... > \lambda_n$$

The vectors defined by

$$P_i = u_{1i}X_1 + ... + u_{ni}X_n$$

are called the **principal components**.

Notice that

$$Var(P_1) = \lambda_1 > ... > Var(P_n) = \lambda_n$$

Therefore, the first components carry the most information.

The principal components can be obtained by diagonalizing the covariance matrix in the following way

$$C = U\Lambda U^T$$

with $\Lambda = diag(\lambda_1, ..., \lambda_n)$ and U is an orthogonal matrix with orthonormal column vectors. The matrix X can now be represented by its principal components, by using the linear transformation

$$P = XU$$

3.1.2 Ordinary Least Square Regression

Let us remember briefly another common method in Statistics, the Ordinary Least Square Regression. That is, finding a solution to a problem of the form

$$Mx_i = y_i$$

for M being a $(n \times n)$-matrix and x_i, y_i fixed n -dimensional vectors, that minimizes $\sum_{i \in \{1,..,N\}} ||y_i - Mx_i||^2$.

Theorem 3.1.2. *If W is of rank at most p and Σ_{XX} is invertible. Then the problem stated above has a unique solution*

$$M = \Sigma_{YX}{\Sigma_{XX}}^{-1}$$

where M is a $(n \times n)$-matrix.

3.1.3 Results about Local and Global Minima

We are observing a Multilayer Perceptron with one linear input layer of n units, one linear output layer of n units and one linear hidden layer of $p < n$ units. N training vectors x_i, $i \in \{1, .., N\}$ of dimension n are being used, that represent different patterns, that we want the network to recognize and map to N output patterns y_i, $i \in \{1, .., N\}$, of dimension n. We will denote X as the input matrix and Y is the output matrix. In this section, we will consider centred training data, that satisfies $\mu_x = 0$. The occurring error, for chosen network architecture F will be measured by the quadratic error function

$$R_E := \sum_{i \in \{1,..,N\}} ||y_i - F(x_i)||^2$$

During the training phase, we want to minimize the error by adjusting the weights in each layer. That means by changing the function F. A minimum can be found by using a numerical algorithm, such as an implementation of the gradient descent method. Therefore, the network suffers from a very common problem in numerical analysis: It is hard to tell whether the gradient descent method can find the global minimum or if it will end in a local minimum. Since we are considering a network that consists of only one hidden layer and all units are linear, the function F must be of the form $F(x_i) = Wx_i = ABx_i$, where B is a real $p \times n$ matrix that represents the transition from the input layer to the hidden layer and A a real $n \times p$ matrix representing the transition from the hidden layer to the output layer. As a network architecture that does not use all units of a hidden layer is not practicable, we can expect A and B to have full rank p, that means to represent surjective, respectively injective maps. Although using linear layers restricts the network capabilities strongly, since only linear functions can be approximated, it still has an important purpose: Understanding the activities in the hidden layers. Notice that the results that are found for one hidden layer can be easily extended to a deeper architecture with more hidden layers, since by linearity of the transitions they can be interpreted as one hidden layer.

We can now make some statements about optimal weights, that means the matrices A and B for that the quadratic error function attains a minimum. These optimal weights are never unique since they can always be multiplied by an invertible matrix C and result in the same global map

$$(AC)(C^{-1}B) = AB = W.$$

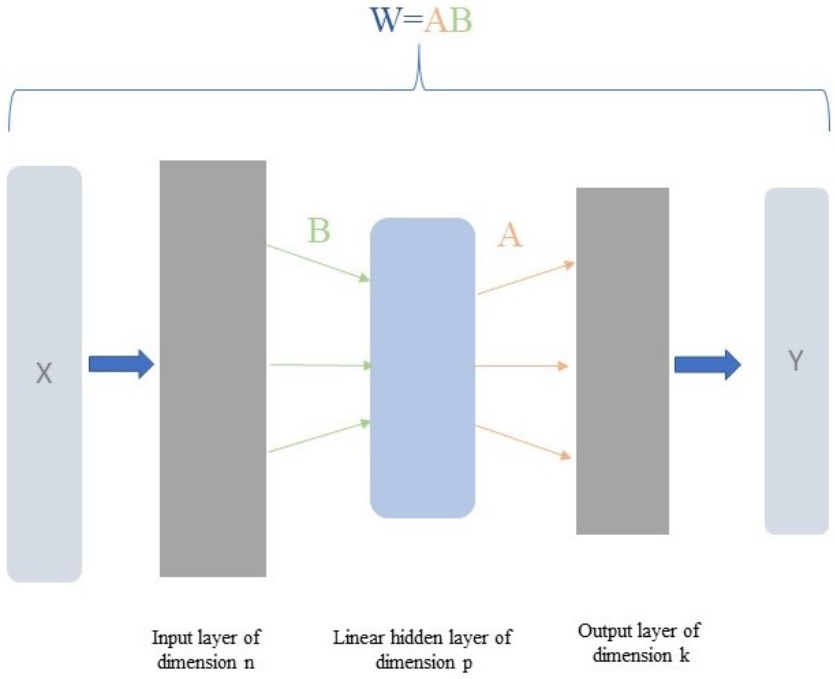


Figure 3.1: Multilayer Perceptron with one linear hidden layer

Theorem 3.1.3. *For a fixed $n \times p$ matrix A the function $R_E(A, B)$ is convex in the coefficients of B. It also states that:*

1. *E attains its minimum for any B that satisfies*

$$A^T AB\Sigma_{XX} = A^T \Sigma_{YX}. \tag{3.1.1}$$

2. *If Σ_{XX} is invertible and A has full rank p, then E is strictly convex in B and a unique minimum is reached in*

$$B = \widehat{B}(A) = (A^T A)^{-1} A^T \Sigma_{YX} {\Sigma_{XX}}^{-1}. \tag{3.1.2}$$

3. *In the autoassociative case it follows:*

$$B = \widehat{B}(A) = (A^T A)^{-1} A^T. \tag{3.1.3}$$

Proof.

1. Let A be any fixed $(n \times p)$-matrix and B a $(p \times n)$-matrix, for that R_E attains a minimum. First observe that, by definition, the square of the euclidean norm of a vector is just the sum of its squared elements. Therefore we can conclude, that the sum of the squared norm of two vectors v, w is equal to the squared norm of the vector $p := (v^T, w^T) = vec([v, w])^T$. That means

$$||v||^2 + ||w||^2 = ||p||^2 = ||(v^T, w^T)||^2$$

So we obtain that

$$\begin{aligned} R_E(A,B) &= \sum_{i \in \{1,..,N\}} ||y_i - ABx_i||^2 \\ &= ||vec(Y - ABX)||^2 \\ &= vec(Y - ABX)^T vec(Y - ABX) \\ &\overset{(1.2.13(1))}{=} tr(vec(Y - ABX)vec(Y - ABX)^T). \end{aligned} \tag{3.1.4}$$

By Lemma 1.2.13(2) we know

$$||vec(Y - ABX)||^2 = ||vec(Y) - (X^T \otimes A)vecB||^2$$

and therefore by Theorem 1.2.10 and (1.2.12) it follows that

$$R_E(A,B) = ||vec(Y) - (X^T \otimes A)vec(B)||^2$$

is convex in the elements of B and a global minimum is reached if and only if

$$(X^T \otimes A)^T (X^T \otimes A)vec(B) = (X^T \otimes A)^T vec(Y).$$

Looking at the left hand side we obtain

$$\begin{aligned} (X^T \otimes A)^T (X^T \otimes A)vec(B) &\overset{(1.2.13(5))}{=} (X \otimes A^T)(X^T \otimes A)vec(B) \\ &\overset{(1.2.13(3))}{=} (XX^T \otimes A^T A)vec(B) \\ &= (\Sigma_{XX} \otimes A^T A)vec(B) \\ &\overset{1.2.13(2)}{=} vec(A^T AB\Sigma_{XX}) \end{aligned}$$

While at the right hand we observe

$$\begin{aligned} (X^T \otimes A)vec(y) &\overset{1.2.13(5)}{=} (X \otimes A^T)vec(Y) \\ &\overset{1.2.13(2)}{=} vec(A^T YX^T) \\ &= vec(A^T \Sigma_{YX}) \end{aligned}$$

So it follows

$$vec(A^T AB\Sigma_{XX}) = vec(A^T\Sigma_{YX}) \tag{3.1.5}$$

and therefore R_E attains a minimum in B that satisfies

$$A^T AB\Sigma_{XX} = A^T\Sigma_{YX}$$

2. To prove the second statement, we assume that Σ_{XX} is invertible and A has full rank p. By (1.2.2) and Lemma 1.2.8 we can conclude that $A^T A$ a symmetric and positive definite. Lemma 3.0.3 states and Σ_{XX} is symmetric and positive semidefinite. Since by assumption Σ_{XX} has full rank it follows that Σ_{XX} is positive definite. We can deduce then with Lemma 1.2.13(7) that

$$(X^T \otimes A)^T(X^T \otimes A) = A^T A \otimes \Sigma_{XX}$$

is positive definite and symmetric. We know by Theorem 1.2.10(3) that E has a unique minimum in

$$\widehat{B}(A) = (A^T A)^{-1}A^T\Sigma_{YX}{\Sigma_{XX}}^{-1}$$

that we obtain by (3.1.1).

3. Since in the autoassociative case $x_i = y_i$ for $i \in \{1, .., N\}$, it results that $\Sigma_{XX} = \Sigma_{XY} = \Sigma_{YX} = \Sigma_{YY}$, so

$$\widehat{B}(A) = (A^T A)^{-1}A^T.$$

□

A similar result can be formulated for a fixed matrix B of weights connecting the input layer to the hidden layer.

Theorem 3.1.4. *For a fixed $p \times n$ matrix B the function $R_E(A, B)$ is convex in the coefficients of A. It also states that:*

1. *R_E attains its minimum for any A that satisfies*

$$AB\Sigma_{XX}B^T = \Sigma_{YX}B^T. \tag{3.1.6}$$

2. *If Σ_{XX} is invertible and B has full rank p, then R_E is strictly convex in A and a unique minimum is reached in*

$$A = \widehat{A}(B) = \Sigma_{YX}B^T(B\Sigma_{XX}B^T)^{-1}. \tag{3.1.7}$$

3. *In the autoassociative case it follows*

$$A = \widehat{A}(B) = \Sigma_{XX}B^T(B\Sigma_{XX}B^T)^{-1}. \tag{3.1.8}$$

Proof.

1. Like in the previous proof, we use

$$R_E(A,B) = ||vec(Y - ABX)||^2.$$

Now in order to find an expression dependent on the vector $vec(A)$ we can write

$$\begin{aligned} R_E(A,B) = ||vec(Y - ABX)||^2 &= ||vec(Y) - vec(IdA(BX)||^2 \\ &\overset{(1.2.13(2))}{=} ||vec(Y) - ((BX)^T \otimes Id)vec(A)||^2. \end{aligned}$$

We can now conclude with Theorem 1.2.10(1) that E is convex in A and has a global minimum if A satisfies

$$(X^TB^T \otimes Id)^T(X^TB^T \otimes Id)vec(A) = (X^TB^T \otimes Id)^Tvec(Y).$$

Looking at the left-hand side, we obtain:

$$\begin{aligned} (X^TB^T \otimes Id)^T(X^TB^T \otimes Id)vec(A) &\overset{(1.2.13(5))}{=} (BX \otimes Id)(X^TB^T \otimes Id)vec(A) \\ &= (BXX^TB^T \otimes Id)vec(A) \\ &= (B\Sigma_{XX}B^T \otimes Id)vec(A) \\ &\overset{(1.2.13(2))}{=} vec(IdA(B\Sigma_{XX}B^T)^T) \\ &= vec(AB\Sigma_{XX}B^T). \end{aligned}$$

And the right hand side provides

$$\begin{aligned} (X^TB^T \otimes Id)^Tvec(Y) &\overset{(1.2.13(5))}{=} (BX \otimes Id)vec(Y) \\ &\overset{(1.2.13(2))}{=} vec(YX^TB^T) \\ &= vec(\Sigma_{YX}B^T). \end{aligned}$$

So it follows that R_E has a global minimum if A satisfies

$$AB\Sigma_{XX}B^T = \Sigma_{YX}B^T.$$

2. Let B be of full rank and Σ_{XX} be invertible, therefore full rank. With Remark 1.2.2 and Σ_{XX} being positive semidefinite and symmetric we obtain that by

$$\langle x, B\Sigma_{XX}B^Tx\rangle = \langle B^Tx, \Sigma_{XX}B^Tx\rangle \geq 0,$$

the matrix $B\Sigma_{XX}B^T$ is positive semidefinite and by symmetry of Σ_{XX}, symmetric. Since it is also full rank p we can conclude by Lemma 1.2.8 that it is positive definite.

By Lemma 1.2.13(7) we can conclude that

$$(B\Sigma_{XX}B^T \otimes Id) = (X^TB^T \otimes Id)^T(X^TB^T \otimes Id)$$

is positive definite. From Theorem 1.2.10(3) we can deduce that R_E has a unique minimum in

$$\widehat{A}(B) = \Sigma_{YX}B^T(B\Sigma_{XX}B^T)^{-1},$$

that we obtain from (3.1.6) and the fact that $(B\Sigma_{XX}B^T)$ is invertible as a positive definite quadratic matrix.

3. As seen in previous proof, for the autoassociative case we know $\Sigma_{XX} = \Sigma_{YX}$ and therefore the previous formula can be simplified to

$$\widehat{A}(B) = \Sigma_{XX}B^T(B\Sigma_{XX}B^T)^{-1}.$$

□

Using the previous results, we can now make some statements about the global map that corresponds to a critical point.

Theorem 3.1.5. *Let Σ_{XX} be invertible and A of full rank. A and B define a critical point of R_E, that means $\frac{\delta R_E}{\delta a_{ij}} = \frac{\delta R_E}{\delta b_{ij}} = 0$, if and only if*

1. *In the general case:*
   The global map $W = AB$ is of the form

$$W = P_A\Sigma_{YX}{\Sigma_{XX}}^{-1} \tag{3.1.9}$$

   and A satisfies

$$P_A\Sigma = P_A\Sigma P_A = \Sigma P_A. \tag{3.1.10}$$

   Or equivalently, $B = \widehat{B}$, as defined in (3.1.2) and A satisfies (3.1.10).

2. *In the autoassociative case with $\Sigma = \Sigma_{XX}$:*
   The global map is of the form

$$W = P_A \tag{3.1.11}$$

   with $B = \widehat{B}$ and A satisfies

$$P_A\Sigma_{XX} = P_A\Sigma_{XX}P_A = \Sigma_{XX}P_A. \tag{3.1.12}$$

Proof.

1. "$\Longrightarrow$" :
   Let Σ_{XX} be invertible and A, B matrices that define a critical point of R_E. From Theorem 3.1.3 we can deduce that B must be of the form

$$\widehat{B}(A) = (A^T A)^{-1} A^T \Sigma_{YX} (\Sigma_{XX})^{-1}.$$

   So

$$W = AB = A\widehat{B}(A) = \underbrace{(A^T A)^{-1} A^T}_{=P_A} \Sigma_{YX} (\Sigma_{XX})^{-1} = P_A \Sigma_{YX} (\Sigma_{XX})^{-1}$$

   and it follows (3.1.9). Furthermore, A must satisfy, that by multiplying (3.1.6) by A^T on the right hand side, we get

$$\underbrace{AB}_{=W} \Sigma_{XX} \underbrace{B^T A^T}_{=W^T} = \Sigma_{YX} \underbrace{B^T A^T}_{=W^T}$$

   and therefore

$$W \Sigma_{XX} W^T = \Sigma_{YX} W^T$$

   which equals by (3.1.9),

$$\begin{aligned} &P_A \Sigma_{YX} {\Sigma_{XX}}^{-1} \Sigma_{XX} (P_A \Sigma_{YX} {\Sigma_{XX}}^{-1})^T = \Sigma_{YX} (P_A \Sigma_{YX} {\Sigma_{XX}}^{-1})^T \\ &\iff P_A \Sigma_{YX} {\Sigma_{XX}}^{-1} \Sigma_{XX} ({\Sigma_{XX}}^{-1})^T {\Sigma_{YX}}^T P_A^T = \Sigma_{YX} {\Sigma_{XX}}^{-1} \Sigma_{YX} P_A. \end{aligned}$$

   Since the covariance matrix Σ_{XX} as well as the orthogonal projection are symmetric, the inverse of a symmetric matrix is symmetric and by definition $(\Sigma_{YX})^T = \Sigma_{XY}$ we obtain

$$P_A \Sigma_{YX} {\Sigma_{XX}}^{-1} \Sigma_{XY} P_A = \Sigma_{YX} {\Sigma_{XX}}^{-1} \Sigma_{XY} P_A$$

   which equals by definition of Σ to

$$P_A \sum P_A = \sum P_A.$$

   Since by

$$(\Sigma)^T = (\Sigma_{YX} {\Sigma_{XX}}^{-1} \Sigma_{XY})^T = {\Sigma_{XY}}^T ({\Sigma_{XX}}^{-1})^T {\Sigma_{YX}}^T = \Sigma_{YX} {\Sigma_{XX}}^{-1} \Sigma_{XY},$$

   we know that Σ is symmetric. We can conclude

$$P_A \Sigma P_A = (P_A \Sigma P_A)^T = (\Sigma P_A)^T = P_A \Sigma = \Sigma P_A.$$

"$\Longleftarrow$":
Assume A satisfies (3.1.10) and the global map W satisfy (3.1.9). By multiplying (3.1.9) by $(A^TA)^{-1}A^T$ on the left we get:

$$\begin{aligned}&(A^TA)^{-1}A^TP_A\Sigma_{YX}{\Sigma_{XX}}^{-1} = (A^TA)^{-1}A^TW = (A^TA)^{-1}A^TAB = B\\ \Longleftrightarrow\quad &(A^TA)^{-1}A^TA(A^TA)^{-1}A^T\Sigma_{YX}{\Sigma_{XX}}^{-1} = B\\ \Longleftrightarrow\ &B = (A^TA)^{-1}A^T\Sigma_{YX}{\Sigma_{XX}}^{-1} = \widehat{B}(A).\end{aligned}$$

We observe that B satisfies (3.1.2).
Now we can see that A satisfies (3.1.7), by

$$\begin{aligned}&\sum P_A = P_A\sum P_A\\ \Longleftrightarrow\ &\Sigma_{YX}\underbrace{{\Sigma_{XX}}^{-1}\Sigma_{XY}P_A}_{=W^T} = \underbrace{P_A\Sigma_{YX}{\Sigma_{XX}}^{-1}}_{=W}\Sigma_{XY}P_A\end{aligned}$$

$$\Longleftrightarrow \Sigma_{YX}W^T = \Sigma_{YX}B^TA^T = W\underbrace{\Sigma_{XY}P_A}_{=\Sigma_{XX}W^T} = W\Sigma_{XX}B^TA^T$$

$$\Longleftrightarrow \Sigma_{YX}B^TA^T = AB\Sigma_{XX}B^TA^T.$$

Multiplying that by $A(A^TA)^{-1}$ on the right side, we obtain

$$\Sigma_{YX}B^TA^TA(A^TA)^{-1} = AB\Sigma_{XX}B^TA^TA(A^TA)^{-1}$$

$$\Longleftrightarrow \Sigma_{YX}B^T = AB\Sigma_{XX}B^T.$$

So we can conclude, that A and B satisfy, respectively (3.1.6) and (3.1.2) (thus (3.1.1)). Therefore, by Theorem 3.1.3 and Theorem 3.1.4 they define a critical point of R_E.

2. "$\Longrightarrow$"
In the autoassociative case, we consequently obtain by $\Sigma = \Sigma_{XX}$ and (3.1.3) and with the same reasoning as in the general case:
$W = P_A$ and

$$P_A\Sigma_{XX} = P_A\Sigma_{XX}P_A = \Sigma_{XX}P_A.$$

"$\Longleftarrow$":
Let A satisfy (3.1.12) and the global map W satisfy (3.1.11). Multiplying (3.1.11) on the left side by A^{-1} gives us directly $B = \widehat{B} = (A^TA)^{-1}A^T$.
Using (3.1.12) and (3.1.11) we get

$$\begin{aligned}\Sigma_{XX}P_A = P_A\Sigma_{XX}P_A &\Longleftrightarrow \Sigma_{XX}P_A^T = P_A\Sigma_{XX}P_A^T\\ &\Longleftrightarrow \Sigma_{XX}B^TA^T = AB\Sigma_{XX}B^TA^T\\ &\Longleftrightarrow \Sigma_{XX}B^T(A^TA) = AB\Sigma_{XX}B^T(A^TA)\\ &\Longleftrightarrow \Sigma_{XX}B^T = AB\Sigma_{XX}B^T.\end{aligned}$$

A satisfies (3.1.6). Therefore, by Theorem 3.1.3 and Theorem 3.1.4 they define a critical point of R_E.

□

Theorem 3.1.6. *Let $\sum \in \mathbb{R}^{n\times n}$ be full rank with $n \in \mathbb{N}$ distinct ordered eigenvalues $\lambda_1 > \lambda_2 > ... > \lambda_n$ and $I = \{i_1, ..i_p\}$ with $i_1 < i_2 < ... < i_p$ for $p \leq n$ the ordered index set such that , $U_I := [u_{i_1}, ..., u_{i_p}]$ is the matrix formed by the orthonormal eigenvectors of $\sum$ associated to the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$.*
Two full rank matrices $A \in \mathbb{R}^{n\times p}$ and $B \in \mathbb{R}^{p\times n}$ define a critical point of E if and only if there exists a ordered set I of p indices and an invertible $n \times n$ matrix C such that

1. *In the general case, A and B satisfy*

$$A = U_I C \tag{3.1.13}$$

and

$$B = C^{-1} U_I^T \Sigma_{YX} {\Sigma_{XX}}^{-1}. \tag{3.1.14}$$

Additionally, if a such critical point exists, then it states

$$W = P_{U_I} \Sigma_{YX} {\Sigma_{XX}}^{-1} \tag{3.1.15}$$

and

$$R_E(A, B) = tr\Sigma_{YY} - \sum_{i\in I} \lambda_i. \tag{3.1.16}$$

2. *In the auto associative case, A and B satisfy*

$$A = U_I CB = C^{-1} U_I^T \tag{3.1.17}$$

and

$$W = P_{U_I}. \tag{3.1.18}$$

Proof.
Since Σ is a real symmetric matrix it follows by the Spectral Theorem 1.2.9 that it is diagonalizable:

$$\Sigma = U\Lambda U^T \tag{3.1.19}$$

where U is an orthogonal matrix of eigenvectors of Σ and $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_n)$, a diagonal Matrix with the decreasing eigenvalues of Σ on the diagonal. Since Σ is full rank we know that $\Sigma_{XX}, \Sigma_{YX}, \Sigma_{XY}$ are full rank.
"$\Longleftarrow$:

Let $A = U_I C$ and $B = C^{-1} U_I^T \Sigma_{YX} {\Sigma_{XX}}^{-1}$.
By

$$\begin{aligned}(A^T A)^{-1} A^T \Sigma_{YX} {\Sigma_{XX}}^{-1} &= ((U_I C)^T (U_I C))^{-1} (U_I C)^T \Sigma_{YX} {\Sigma_{XX}}^{-1} \\ &= (C^T {U_I}^T U_I C)^{-1} C^T U_I^T \Sigma_{YX} {\Sigma_{XX}}^{-1} \\ &= C^{-1} ({U_I}^T U_I)^{-1} (C^T)^{-1} C^T {U_I}^T \Sigma_{YX} {\Sigma_{XX}}^{-1} \\ &= C^{-1} {U_I}^T \Sigma_{YX} {\Sigma_{XX}}^{-1} \\ &= B\end{aligned}$$

and

$$\begin{aligned}\Sigma_{YX} B^T (B \Sigma_{XX} B^T)^{-1} &= \Sigma_{YX} (C^{-1} {U_I}^T \Sigma_{YX} {\Sigma_{XX}}^{-1})^T ((C^{-1} {U_I}^T \Sigma_{YX} {\Sigma_{XX}}^{-1}) \\ &\quad \times \Sigma_{XX} (C^{-1} {U_I}^T \Sigma_{YX} {\Sigma_{XX}}^{-1})^T)^{-1} \\ &= \Sigma_{YX} (C^{-1} {U_I}^T \Sigma_{YX} {\Sigma_{XX}}^{-1})^T ((C^{-1} {U_I}^T \Sigma_{YX} {\Sigma_{XX}}^{-1})^T)^{-1} \\ &\quad \times {\Sigma_{XX}}^{-1} ((C^{-1} {U_I}^T \Sigma_{YX} {\Sigma_{XX}}^{-1}))^{-1} \\ &= \Sigma_{YX} {\Sigma_{XX}}^{-1} (C^{-1} {U_I}^T \Sigma_{YX} {\Sigma_{XX}}^{-1})^{-1} \\ &= \Sigma_{YX} {\Sigma_{YX}}^{-1} {\Sigma_{XX}}^{-1} \Sigma_{XX} {U_I^T}^{-1} C = U_I C \\ &= A,\end{aligned}$$

it follows that A and B satisfy (3.1.7) and respectively (3.1.2). Therefore A and B define a critical point.
"$\Longrightarrow$:"
Let A and B define critical points of E.

First, observe that

$$P_{U^T A} = U^T P_A U \tag{3.1.20}$$

since by definition of $P_{U^T A}$:

$$P_{U^T A} = U^T A((A^T U)(U^T A))^{-1} A^T U = U^T \underbrace{A(A^T A)^{-1} A^T}_{=P_A} U = U^T P_A U.$$

We can conclude by

$$\underbrace{U P_{U^T A} U^T}_{=P_A} \underbrace{U \Lambda U^T}_{=\sum} \overset{(3.1.19)}{=} P_A \sum \overset{(3.1.10)}{=} \sum P_A = U \Lambda U^T U P_{U^T A} U^T$$

that

$$U P_{U^T A} \Lambda U^T = U \Lambda P_{U^T A} U^T$$

and consequently,

$$P_{U^T A} \Lambda = \Lambda P_{U^T A}.$$

Now we can observe

$$
\begin{aligned}
P_{U^TA} \cdot \Lambda &= \begin{pmatrix} p_{1,1} & \cdots & p_{1,n} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ p_{n,1} & \cdots & p_{n,n} \end{pmatrix} \begin{pmatrix} \lambda_{1,1} & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \lambda_{n,n} \end{pmatrix} \\
&= \begin{pmatrix} \lambda_1 \cdot p_{1,1} & \cdots & \lambda_n \cdot p_{1,n} \\ \vdots & & \vdots \\ \lambda_1 \cdot p_{n,1} & \cdots & \lambda_n \cdot p_{n,n} \end{pmatrix} \\
&= \begin{pmatrix} \lambda_{1,1} & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \lambda_{n,n} \end{pmatrix} \begin{pmatrix} p_{1,1} & \cdots & p_{1,n} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ p_{n,1} & \cdots & p_{n,n} \end{pmatrix} \\
&= \begin{pmatrix} \lambda_1 \cdot p_{1,1} & \cdots & \lambda p_{1,n} \\ \vdots & & \vdots \\ \lambda_n p_{n,1} & \cdots & \lambda_n \cdot \lambda p_{n,n} \end{pmatrix} \\
&= \Lambda \cdot P_{U^TA} \\
&\iff \lambda_k \cdot p_{i,k} = \lambda_i \cdot p_{i,k} \text{ for all } i,k \in \{1,..,n\}.
\end{aligned}
$$

Knowing that $\lambda_1 > ...\lambda_n > 0$, it follows, that $\lambda_i \neq 0$ for $i \neq k$, thus P_{U^TA} is a diagonal matrix of rank p. Since orthogonal projectors only have eigenvalues 1 and 0, we can define a unique index ordered set $I = \{i_1, ..., i_p\}$, with $1 \leq i_1 < ... < i_p \leq n$, that represents the indices of the positions where P_{U^TA} has an entry that is one. That means $i_k \in I$ if $p_{i_k,i_k} = 1$. By that, we can define a diagonal matrix

$$
M_I = \begin{pmatrix} j_1 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & j_n \end{pmatrix}
$$

such that $j_k = 1$ if $j_k \in I$, for $k \in \{1, ..., n\}$ and therefore $P_{U^TA} = M_I$. Now define $U_I := [u_{i_1}, ..., u_{i_p}]$ as the matrix of the i_k -th eigenvector of $\sum$. See that, without loss of generality, if $j_2 = 0$ and $j_1 = j_3 = ... = j_n = 1$, we can observe $U_I = [u_1, u_3, .., u_n]$

and clearly

$$
\begin{aligned}
UM_IU^T &= \begin{pmatrix} u_{1,1} & \cdots & u_{n,1} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ u_{1,n} & \cdots & u_{n,n} \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & & & \vdots \\ 0 & & 1 & & \vdots \\ \vdots & & & \ddots & \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} \begin{pmatrix} u_{1,1} & \cdots & u_{n,1} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ u_{1,n} & \cdots & u_{n,n} \end{pmatrix} \\
&= \begin{pmatrix} u_{1,1} & 0 & u_{3,1} & \cdots & u_{n,1} \\ \vdots & & \vdots & & \\ \vdots & & \vdots & & \\ u_{1,n} & 0 & u_{3,n} & \cdots & u_{n,n} \end{pmatrix} \begin{pmatrix} u_{1,1} & \cdots & u_{1,n} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ u_{n,1} & \cdots & u_{n,n} \end{pmatrix} \\
&= \begin{pmatrix} u_{1,1} & 0 & u_{3,1} & \cdots & u_{n,1} \\ \vdots & & \vdots & & \\ \vdots & & \vdots & & \\ u_{1,n} & 0 & u_{3,n} & \cdots & u_{n,n} \end{pmatrix} \begin{pmatrix} u_{1,1} & \cdots & u_{1,n} \\ 0 & & 0 \\ u_{3,1} & \cdots & u_{3,n} \\ \vdots & & \vdots \\ u_{n,1} & \cdots & u_{n,n} \end{pmatrix} \\
&= U_I U_I^T.
\end{aligned}
$$

So, it yields $P_A = UP_{U^TA}U^T = U_IU_I^T = P_{U_I}$. In other words: P_A is the orthogonal projection on the subspace spanned by the column vectors of U_I. Therefore, we can deduce that the column vectors of A span the same space as the column vectors of U_I. That means there exists a bijection from A to U_I, thus there exists an invertible $(p \times p)$-dimensional matrix C such that $A = U_IC$. Since by assumption A and B define critical points of R_E and A is invertible, by (3.1.2) B must be of the form

$$B = \widehat{B}(A) = C^{-1}U_I^T\Sigma_{YX}{\Sigma_{XX}}^{-1}.$$

From previous Theorem and $P_A = P_{U_I}$ follows directly (3.1.15).
It is left to prove (3.1.16). As in the proof of (3.1.3) we can write

$$
\begin{aligned}
R_E(A,B) &= (vec(Y - ABX))^T(vec(Y - ABX)) \\
&= vec(Y)^Tvec(Y) - 2(vec(ABX)^Tvec(Y)) + vec(ABX)^Tvec(ABX)
\end{aligned}
$$

and by using (1.2.13(1)), we obtain

$$
\begin{aligned}
R_E(A,B) &= tr(YY^T) - 2tr(ABXY^T) + tr(ABXX^TB^TA^T) \\
&= tr(\Sigma_{YY}) - 2tr(W\Sigma_{XY}) + tr(W\Sigma_{XX}W^T).
\end{aligned}
$$

If A is of full rank and $B = \widehat{B}(A)$, then

$$W(A) = AB = U_ICC^{-1}U_I^T\Sigma_{YX}{\Sigma_{XX}}^T = P_{U_I}\Sigma_{YX}{\Sigma_{XX}}^{-1} = P_A\Sigma_{YX}{\Sigma_{XX}}^{-1}.$$

So,

$$tr(W\Sigma_{XX}W^T) = tr(P_A \sum P_A) \overset{(3.1.10)}{=} tr(P_A \sum) = tr(UP_{U^TA}U^TU\Lambda U^T)$$
$$\overset{(1.2.7)}{=} tr(P_{U^TA}U^TU\Lambda) = tr(P_{U^TA}\Lambda)$$

and

$$tr(W\Sigma_{XY}) = tr(P_A \sum) = tr(UP_{U^TA}U^TU\Lambda U^T) = tr(P_{U^TA}\Lambda).$$

Consequently, for any A of rank p we get a explicit formula for $R_E(A)$,

$$R_E(A) = tr(\Sigma_{YY}) - tr(P_{U^TA}\Lambda)$$

Since $A = U_I C$, we conclude that $P_{U^TA} = M_I$ and therefore $tr(P_{U^TA}\Lambda) = \sum_{i\in I}$. So it follows (3.1.16).
The autoassiciative case follows directly from applying $\Sigma_{XX} = \Sigma_{YX} = \Sigma_{XY} = \Sigma_{YY}$. □

That means a critical point W of rank p is always the product of the Ordinary Least Squares Regression matrix and an orthogonal projection onto the subspace spanned by p eigenvectors of $\sum$.

Finally, we can state our main result, that is, that R_E has no local minima and all critical points, other than the global minima, correspond to saddle points.

Theorem 3.1.7. *The critical point W of R_E associated with the index set $\{1, ..., p\}$, in other words, associated with the biggest p-eigenvalues, is the unique local and global minimum of R_E. The other $\binom{n}{p} - 1$ index sets correspond to saddle points.*

Proof. Let A and B matrices such that (3.1.13) and (3.1.14) are satisfied, and therefore define a critical point of R_E. Furthermore, let I be a p-index set with $I \neq \{1, .., p\}$. Our aim is now to show that A and B do not define a local or global minimum, but a saddle point, that means there exist $\tilde{A}$ and $\tilde{B}$ in a neighbourhood of A and B such that

$$R_E(\tilde{A}, \tilde{B}) < R_E(A, B)$$

Let U_p the matrix of the first p eigenvectors of $\sum$ corresponding to the p-biggest eigenvalues and U_I of the i-th eigenvectors, for $i \in I$. Moreover, let $j \in I$ and $k \in \{1, .., p\}\backslash I$. Now let us construct a matrix $\tilde{U}_I$ by replacing the j-th vector of U_I by

$$\tilde{u}_j := (1 + \epsilon^2)^{\frac{1}{2}}(u_j + \epsilon u_k)$$

That can be interpreted as moving from u_j a little in the direction of the vector u_k

of U_p. Notice that,

$$\tilde{U}_I^T\tilde{U}_I = \begin{pmatrix} \tilde{u_1} \\ \vdots \\ (1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k) \\ \tilde{u}_{j+1} \\ \vdots \\ \tilde{u_p} \end{pmatrix} \begin{pmatrix} \tilde{u_1} & \cdots & (1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k) & \tilde{u}_{j+1} & \cdots & \tilde{u_p} \end{pmatrix}$$

$$= \begin{pmatrix} <\tilde{u}_1,\tilde{u}_1> & & <(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k),\tilde{u}_1> & & <\tilde{u_p},\tilde{u}_1> \\ <\tilde{u}_1,\tilde{u}_2> & \cdots & <(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k),\tilde{u}_2> & \cdots & \\ & & \vdots & & \\ \vdots & & <(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k),(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k)> & & \vdots \\ & & \vdots & & \\ <\tilde{u}_1,\tilde{u}_p> & & & e & <\tilde{u_p},\tilde{u}_p> \end{pmatrix}$$

Since $u_k \notin I$ is not a vector of $\tilde{U}_I$ and all vectors of $\tilde{U}_I$ are orthonormal and

$$\langle (1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k), (1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k)\rangle = (1+\epsilon^2)^{-1}||uj+\epsilon u_k||^2 = 1$$

it follows that $\tilde{U}_I^T\tilde{U}_I = Id$. Define now $\tilde{A} := \tilde{U}_I C$ and $\tilde{B} := \widehat{B}(\tilde{A})$ for some invertible $p \times p$ matrix. Consequently,

$$P_{\tilde{A}} = \tilde{U}_I C(C^T\tilde{U}_I^T\tilde{U}_I C)^{-1}C^T\tilde{U}_I^T = \tilde{U}_I\tilde{U}_I^T.$$

and therefore,

$$P_{U^T\tilde{A}} = U^T\tilde{U}_I\tilde{U}_I^T U = \left(\begin{array}{c} \underline{u_1} \\ \vdots \\ \underline{u_k} \\ \vdots \\ u_n \end{array}\right) \left(\tilde{u_1} \,\middle|\, \cdots \,\middle|\, (1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k) \,\middle|\, \cdots \,\middle|\, \tilde{u_p} \right)$$

$$\times \left(\begin{array}{c} \underline{\tilde{u_1}} \\ \vdots \\ \underline{(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k)} \\ \underline{\tilde{u}_{j+1}} \\ \vdots \\ \tilde{u_p} \end{array}\right) \left(\begin{array}{c|c|c|c} u_1 & \cdots & \cdots & u_n \end{array}\right)$$

$$= \begin{pmatrix} <u_1,\tilde{u}_1> & & \cdots<u_1,(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k)>\cdots & <u_1,\tilde{u_p}> \\ \vdots & & & \\ & <u_k,(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k)> & & \\ \vdots & & <u_j,(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k)> & \vdots \\ <u_p,\tilde{u}_1> & & & <u_p,\tilde{u_p}> \\ \vdots & & & \vdots \\ <u_n,\tilde{u}_1> & & & <u_n,\tilde{u_p}> \end{pmatrix}$$

$$\times \begin{pmatrix} <\tilde{u_1},u_1> & & \cdots<(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k),u_1>\cdots & <\tilde{u}_p,u_p> & \cdots<\tilde{u}_p,u_n> \\ \vdots & & \vdots & \vdots & \vdots \\ & <(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k),u_k> & & & \\ & & <(1+\epsilon^2)^{-\frac{1}{2}}(u_j+\epsilon u_k),u_j> & & \\ & & \vdots & & \\ <\tilde{u}_p,u_n>\cdots & & & <\tilde{u}_p,u_p> & \cdots<\tilde{u}_n,u_p> \end{pmatrix}$$

So by orthonormality of eigenvectors, we obtain a diagonal matrix with diagonal entries of the form

$$d_i = \begin{cases} 0 \text{ for } i \notin I \cup \{k\} \\ 1 \text{ for } i \in I \text{ and } i \neq j, i \neq k \\ 1/(1+\epsilon^2) \text{ for } i = j \\ \epsilon^2/(1+\epsilon^2) \text{ for } i = k \end{cases} .$$

Then, we get

$$R_E(\tilde{A},\tilde{B}) = tr\Sigma_{YY} - trP_{U^T\tilde{A}}\Lambda$$

and since Λ carries the eigenvalues of $\sum$ we can conclude

$$R_E(\tilde{A},\tilde{B}) = tr\Sigma_{YY} - (\sum_{i\in I\setminus\{j\}} \lambda_i + \frac{1}{(1+\epsilon^2}\lambda_j + \frac{\epsilon^2\lambda_k}{1+\epsilon^2}) = tr\Sigma_{YY} - \sum_{i\in I}\lambda_i - \frac{\epsilon^2(\lambda_k-\lambda_j)}{1+\epsilon^2}$$

$$= R_E(A,B) - \frac{\epsilon^2(\lambda_k-\lambda_j)}{1+\epsilon^2}.$$

All eigenvalues are non-negative and of decreasing order, so by $k < j$ follows $\lambda_k > \lambda_j$. Therefore, it yields that, by having chosen ϵ arbitrary, that there is a neighbourhood

of A, B such that there exist $\tilde{A}, \tilde{B}$ in that neighbourhood that satisfies $R_E(\tilde{A}, \tilde{B}) < R_E(A, B)$. Hence, A, B do not define a minimum, and by convexity of R_E neither a maximum, so they define a saddle point. □

Remark 3.1.8. The critical points defined by rank deficient matrices A and B, correspond to saddle points as well, but will not be discussed in this work. (See [3], page 58)

3.1.4 The Autoassociative Case and its Connection to PCA

As a special case, we take a closer look at the autoassociative architecture.

Looking at the autoassociative case of our result (3.1.6) we find that in the global minimum

$$\begin{aligned} A &= U_p C \\ B &= C^{-1} U_p^T \\ W &= P_A. \end{aligned}$$

$U_p := U_{\{1,\dots,p\}}$ is the matrix of the first p-eigenvectors of Σ_{XX}.
Now choosing the identity map for C, we can observe that the optimal way of lowering the dimension of the input data is, up to equivalence, the projection of the data on the subspace spanned by the eigenvectors of Σ_{XX}. That coincides with the transformation to the principal components of X, that we have seen in (3.1.1). Furthermore, this result answers an important question: How well can a network recognize patterns that it has not seen in the training phase, that means that cannot be found in X? The answer can be given by

$$R_E(\widehat{A}, \widehat{B}) = ||U_p v - v||^2$$

where v describes an unknown input-pattern. Hence, the error can be measured by the distance of the unknown pattern to the space spanned by the principal components of the patterns, seen during the training phase.

3.1.5 Conclusion

We took a closer look at a linear Multilayer Perceptron with one hidden layer and could observe that, up to equivalence, the unique global minimum can be found and there are no local minima. All critical points, apart from the global minimum, refer to saddle points. By observing autoassociative architecture, we found a measure of how well a network can recognize unknown patterns. Moreover, we have seen that in some special cases the optimal results coincide with well-known results in Statistics, such as Ordinary Least Square Regression and Principal Component Analysis. That means instead of using a gradient descent method on the error function, in these special cases, we could use these already known methods directly. However, using the gradient descent method, with appropriate parameters, to avoid the algorithm ending in a saddle point, will also work on more general data and can be expected to produce the same results, as no local minima will disturb the gradient descent.

3.2 Autoassociation with Multilayer Perceptrons and Singular Value Decomposition

In this section, we will take a closer look at the autoassociative case. Having discussed the linear case in the previous section, we will now move onto a non-linear case, which consists of a network with one non-linear hidden layer. Hence, we will be looking at an auto-associative network consisting of one linear input and one linear output layer, both of n units and one non-linear hidden layer of $p < n$ units with non-linear activation function g. As in the previous chapter, let B denote a real $p \times n$ matrix that represents the transition from the input layer to the hidden layer and A a real $n \times p$ matrix representing the transition from the output of the hidden layer to the output layer. As before, $X := [x_1, ..., x_N]$ is the input matrix and $Y := [y_1, ..., y_N]$ the output matrix. Additionally, we will define

$$H_0 := BX + b_1 v^T$$

as the input of the hidden layer, with n-dimensional bias vector b_1 and v a N-dimensional vector of ones. Furthermore, define $H := [h_1, ..., h_N] = g(H_0)$ as the output matrix of the hidden layer, with

$$h_i = g(Bx_i + b_1) \quad i \in \{1, .., N\}$$

for some nonlinear function g. Since we are trying to approximate the identity map for linear input x_i, in the output layer the non-linearity has to be removed, that means it must be of the form

$$y_i = Ah_i + b_2$$

Where b_2 describes the bias of the output of the hidden layer. Consequently, the network activity is described by the function F with component-wise activity

$$F_i(x_i) = A(g(Bx_i + b_1 v^T) + b_2 v^T)$$

or equivalently,

$$F(X) = A(g(BX + b_1 v^T) + b_2 v^T) = Y$$

Since we want the network to reproduce the input the occurring error will be measured by the quadratic error function

$$R_E := ||X - F(X)||^2 \tag{3.2.1}$$

with respect to the matrix norm $||A|| = tr(AA^T)$. We want to find the optimal weights A, B and optimal biases b_1, b_2 that minimizes the error function.

Remark 3.2.1. Observe that the norm equals the norm used in the previous chapter (see (3.1.4)).

To begin with, let us recapitulate the Singular Value Decomposition.

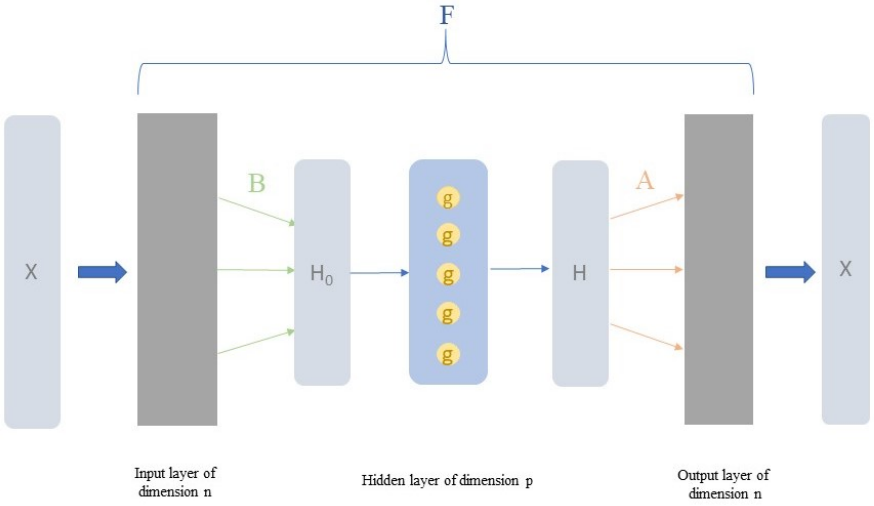


Figure 3.2: Autoassociative Multilayer Perceptron with one hidden layer

3.2.1 Singular Value Decomposition

Singular Value Decomposition is a tool from linear algebra to factorize real- or complex-valued matrices. Since in our case we are only interested in real values, we will just state the Theorem for the real case.

Theorem 3.2.2. *Let $M \in R^{n\times m}$ real valued matrix of rank $r \leq m, n$. There exist two real orthogonal matrices $U \in \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{m\times m}$ and a quasi-diagonal matrix $D \in \mathbb{R}^{n\times m}$ of the form*

$$D = \begin{pmatrix} s_1 & & & & \\ & \ddots & & & \\ & & s_r & & \\ & & & 0 & \\ & & & & \ddots \end{pmatrix}$$

such that

$$M = UDV$$

with positive, decreasing and uniquely defined entries $s_1 > ... > s_r > 0$ that are the singular values of M.

Proof. See [4], page 217-218. □

Remark 3.2.3. In the case that M is a quadratic matrix, the singular value decomposition equals the eigendecomposition that was used in the previous chapter in the proof of (3.1.6).

3.2.2 Explicit Results

In the following we will first examine the transition from the output of the hidden layer H to the linear output layer Y and then take a look at the non-linear activity

between input layer X and the output of the hidden layer H.

By deriving the error function with respect to b_2 we can find an **optimal bias vector** $\widehat{b}_2$.

Theorem 3.2.4. *Let R_E be the quadratic error function defined by*

$$R_E := ||X - AH - b_2 v^T||^2$$

with respect to the matrix norm $||A|| = tr(AA^T)$. Let X be a $n \times N$-matrix and v an n-dimensional vector of ones. Then, minimization of E with respect to b_2 yields the optimal bias vector $\hat{b}_2$ is given by

$$\hat{b}_2 = \frac{1}{N}(X - AH)v.$$

Observe that this bias vector ensures that the average input equals the average output, which is desired when trying to reproduce data in the most accurate way.

Proof.
First note that

$$\begin{aligned} R_E &= ||X - F(X)||^2 \\ &= ||X - AH - b_2 v^T||^2 \\ &= tr((X - AH - b_2 v^T)(X - AH - b_2 v^T)^T) \\ &= tr((X - AH) - b_2 v^T)(X - AH)^T - v b_2^T)) \\ &= tr((X - AH)(X - AH)^T) + tr((X - AH)(-v b_2^T)) \\ &\quad + tr((b_2 v^T)(v b_2^T) - tr(b_2 v^T (X - AH)^T) \end{aligned}$$

so we get that the derivative equals

$$\frac{\partial R_E}{\partial b_2} = \frac{\partial}{\partial b_2}(-tr(((X - AH)v)b_2^T) + Ntr(b_2 b_2^T) - tr(b_2 v^T (X - AH)^T)$$

Notice that $((X - AH)u) =: v$ and $v^T(X - AH)^T = v^T$ equals some n-dimensional vector v, so the derivative is

$$\frac{\partial}{\partial b_2}(-tr(((X-AH)v)b_2^T) = \frac{\partial}{\partial b_2}(-tr(v b_2^T)) = -\frac{\partial}{\partial b_2}\sum_{i \leq n} v_i b_{2_i} = -v = -((X-AH)v)$$

And equivalently

$$\frac{\partial}{\partial b_2}(-tr(b_2 v^T (X - AH)^T)) = -v = -(X - AH)v.$$

Clearly,

$$\frac{\partial}{\partial b_2} N tr(b_2 b_2^T) = 2N b_2.$$

Since vector v is a vector of ones we get $-v^T(X - AH)^T = -(X - AH)v$ and therefore

$$\frac{\partial R_E}{\partial b_2} = -2(((X - AH)v) - Nb_2).$$

So, using the previously seen result (1.2.10) we can deduce $\hat{b}_2 = \frac{1}{N}(X - AH)v$. □

Now to find a corresponding optimal matrix $\widehat{A}$ and $\widehat{H}$ we substitute $\widehat{b}_2$ in R_E and obtain

$$R_E = ||X - AH - \frac{1}{N}(X - AH)vv^T||^2 = ||X(Id - \frac{1}{v}v^T) - AH(Id - \frac{1}{N}vv^T)||^2.$$

So to simplify, we define $X' := X(Id - \frac{1}{N}vv^T)$ and $H' := H(Id - \frac{1}{N}vv^T)$ so it yields

$$R_E = ||X' - AH'||^2.$$

Observe, that the transition from the output of hidden-layer to the output layer is linear, so both A and H' are matrices with real entries that can be expected to have full rank and we are considering the same norm as in previous chapter (see Remark 3.2.1. Thereupon, we find that this problem can be reduced to the autoassociative case (3.1.4) from the previous chapter with optimal matrix

$$\widehat{A} = U'_p C^{-1} \tag{3.2.2}$$

for some arbitrary invertible $(p \times p)$-matrix C and U'_p the matrix of the first eigenvectors of sample covariance matrix of X'.

Remark 3.2.5. Building X' out of X is centering the data by removing the average vector μ_X.

Remark 3.2.6. We notice that the optimal weight matrix, as well as the optimal bias, is obtained independently from the way the output H, or rather H', of the hidden layer was produced. That means the choice of the nonlinear function g has no impact on the choice of the optimal biases and weights, connection the output layer and the output of the hidden layer.

3.2.3 Linear Function in the Hidden Units

Before discussing the non-linear case we will take a look at the linear case. Notice, that by linearity of matrix operations, a linear activation function can be interpreted as a part of the linear transition A from the input layer to the output H of the hidden layer that we have already examined in the previous section. Consequently, we can just observe $H_0 = H$. As before, this case can be reduced to the auto-associative case (3.1.4) of the previous chapter, that gives the optimal weight matrix

$$\widehat{B} = CU'^T_p$$

Now we can find an explicit representation of the optimal bias vector $\hat{b}_2$.

Corollary 3.2.7. *In the linear case the optimal bias vector is given by*

$$\hat{b}_2 = (Id - U'_p U'^T_p)\mu_x - U'_p C^{-1} b_1$$

Proof. Substituting the optimal weight matrix $\widehat{B}$ in the definition of H_0 we obtain

$$H_0 = H = CU'^T_p X + b_1 v^T \tag{3.2.3}$$

Now, taking in account (3.2.4) and inserting the optimal weight matrix $\widehat{A}$, we obtain

$$\begin{aligned}\hat{b}_2 &= \frac{1}{N}(X - U'_p C^{-1}(CU'^T_p X + b_1 v^T))v \\ &= \frac{1}{N}(Xv - U'_p U'^T_p Xv + U'_p C^{-1} b_1 N) \\ &= (Id - U'_p U'^T_p)\mu_x - U'_p C^{-1} b_1\end{aligned}$$

□

Remark 3.2.8. Since the eigenvalue decomposition, that was used to find the optimal weights in the previous chapter, is a special case of the singular value decomposition, the optimal weights obtained in this section have been calculated using Singular Value Decomposition.

3.2.4 Hidden layer with Non-Linear Units

Let us proceed now by discussing the transition from the input layer to the output of the hidden layer H. In particular, we want to observe the impact of the non-linear activation function in the hidden layer on the choice of the optimal weight matrices. We have seen already in the linear case, that there is an optimal weight matrix $\widehat{B}$ depending on X', that produces an optimal output of the hidden layer $\widehat{H}'$, for the centered input $\widehat{H}'_0$.
Now we want to see, if by making small changes on $\widehat{H}'_0$ and $\widehat{B}$, denoted by $\tilde{H}_0$, respectively $\tilde{B}$, we can obtain the result $g(\tilde{H}_0) = \widehat{H}'$. We have seen before (3.2.3) that $\widehat{H}$ is of the form

$$\widehat{H} = CU'^T_p X + b_1 v^T$$

Consequently,

$$\widehat{H}' = CU'^T_p X \tag{3.2.4}$$

We need to choose an appropriate candidate for the non-linear function g:
That is a function that can be linearly approximated somewhere, so the produced output H' can be linearly approximated and we can use the results of the linear case. Therefore, we will assume that g can be approximated linearly for values x in a neighbourhood of zero.

Theorem 3.2.9. *If g is a nonlinear function that can be approximated linearly for small values x, means for arbitrary small $\epsilon > 0$ and $\alpha_0, \alpha_1 \in \mathbb{R}$ with $\alpha_1 \neq 0$*

$$g(x) = \alpha_0 + \alpha_1 x + \epsilon$$

for small x. Then, changing the optimal matrices $\widehat{B}$ and $\widehat{H}_0$ to

$$\tilde{B} = \alpha_1^{-1} C U_p'^T \tag{3.2.5}$$

$$\tilde{H}_0 = \alpha_1^{-1} C U_p'^T X + b_1 v^T \tag{3.2.6}$$

satisfies

$$g(\tilde{H}_0) = \widehat{H}'.$$

Proof.
To begin with, notice that in order to use the linear approximation property of g we have to scale the matrix $\tilde{H}_0$, to make the coefficients small. That can be done by adjusting the bias vector b_1 and the arbitrary invertible matrix C. A good choice for the bias $\widehat{b_1}$ is forcing the average input $\frac{1}{N}\tilde{H}_0 v$ of the hidden layer to be zero by choosing

$$\widehat{b_1} = -\alpha_1^{-1} C U_p'^T \mu_X \tag{3.2.7}$$

and a small, but invertible scaling matrix C. That means with entries that must satisfy $|c_{ij}| < 1$.

Now by substituting (3.2.7) in (3.2.6) we obtain

$$\begin{aligned}
\tilde{H}_0 &= \alpha_1^{-1} C U_p'^T X + (-\alpha_1^{-1} C U_p'^T \mu_X) v^T \\
&= \alpha_1^{-1} C U_p'^T X + (-\alpha_1^{-1} C U_p'^T \frac{1}{N} X v) v^T \\
&= \alpha_1^{-1} C U_p'^T X (Id - \frac{1}{N} v v^T) \\
&= \alpha_1^{-1} C U_p'^T X'.
\end{aligned}$$

Using the singular value decomposition of matrix X' gives us $X' = U_n \Lambda_n V_n$ with U_n consisting of the normalized eigenvectors of $\Sigma_{X'X'}$, V_n the normalized eigenvectors of $\Sigma_{X'X'}^T$ and Λ_n a diagonal matrix with entries $\sqrt{\lambda_i}$ of the decreasingly ordered eigenvalues of $\Sigma_{X'X'}$. We obtain by using the Singular Value Decomposition

$$\begin{aligned}
\tilde{H}_0 &= \alpha_1^{-1} C U_p'^T U_n \Lambda_n V_n \\
&= \alpha_1^{-1} C \Lambda_p V_p
\end{aligned}$$

and now by applying g to $\tilde{H}_0$

$$g(\tilde{H}_0) = \alpha_0 v v^T + C \Lambda_p V_p = \widehat{H}.$$

By $H' = H(Id - vv^T)$ follows,

$$\widehat{H}' = C \Lambda_p V_p$$

with optimal bias

$$\hat{b}_2 := \mu_X - \alpha_0 U_p' C^{-1} v$$

which equals (3.2.3) and therefore prooves the Theorem. □

3.2.5 Conclusion

In this section we could observe that for linear input data the choice of a specific activation function for the units of the hidden layer does not affect the result of an autoassociative network, given that this function can be linearly approximated in a neighbourhood of zero. Hence, non-linear functions at the hidden layer do not improve a network that handles linear input. This result can be inductively extended to a network of several hidden layers. Therefore for linear input data, it is recommendable to use linear layers, since an explicit form of the solution to the optimization problem (3.2.1) can be obtained by using the results of the previous section. This avoids the problem of local minima that appears when we are trying to find a minimum using gradient descent for a network with non-linear layers. Moreover, we understand that the optimal bias vector is the one that removes the average from the data, that means transfers the data to centred data with average zero.

Chapter 4

An Application to Neural Networks

4.1 The Problem

We are looking at a data set of flights that were performed by an airline in the past three years. In order to plan efficiently, the airline wants to get an accurate prediction of block times for future flights.
The **block time** describes the time between the moment when the breaks of a specific airplane are being removed before departure and the time when the breaks are being set at the destination airport. It includes the *taxi out* time, that is the time between leaving the parking spot and take off, as well as the *take off* time, the *enroute* time, *landing* time and the *taxi in* time. Block times determine the departure and landing slots to be requested in each of the airports. Thus, since slots need to be assigned several months before the beginning of each season, having predicted the correct block times is key to achieve an optimal on-time performance for the whole season. We are trying to predict block times for a whole season, only depending on the weekday and time the flight is scheduled for, not on the month. Ideally, we want the block time prediction to be as close to reality as possible, taking as acceptable a maximum deviation of 10 minutes. Also, a time prediction than is longer than actual time is better than forecasting shorter, because arriving late results in higher chances of propagating the delay to the following flights, increasing the costs drastically.

4.2 The Dataset

The used data set consists of 546863 rows of flights that were generated automatically. Since for future predictions we will only have information about the schedule, we should use the same variables for calculating the predictions. The relevant columns are

- `Std UTC`, referring to the time the flight is scheduled
- `DP latacode` giving the code of the departure airport
- `AR latacode`, the arrival airport
- `BlkSched` the scheduled block time
- `ActualBlock` giving the measured actual block time

However, we will separate the scheduled time `Std UTC` into day of the week, time and season that the flight was performed.

	Acreg	Version	Std UTC	Sta Utc	Flightcode	AOC	DP Iatacode	AR Iatacode	Rm Off Block Utc	Rm Takeoff Utc	TaxiOut	Rm Touchdown Utc	Rm On Block Utc	TaxiIn	BlkSched	ActualBlock
561264	YRBMI	737-800	30.08.2019 18:15	30.08.2019 21:55	DY4214	WET	PMI	ARN	30.08.2019 18:15	30.08.2019 18:25	10	30.08.2019 21:46	30.08.2019 21:51	5.0	220	216
561265	YRBMI	737-800	31.08.2019 06:15	31.08.2019 08:25	DY4487	WET	ARN	BUD	31.08.2019 06:15	31.08.2019 06:24	9	31.08.2019 08:24	31.08.2019 08:29	5.0	130	134
561266	YRBMI	737-800	31.08.2019 09:05	31.08.2019 11:15	DY4488	WET	BUD	ARN	31.08.2019 09:08	31.08.2019 09:28	20	31.08.2019 11:22	31.08.2019 11:28	6.0	130	140
561267	YRBMI	737-800	31.08.2019 12:05	31.08.2019 16:30	DY4223	WET	ARN	AGP	31.08.2019 12:21	31.08.2019 12:31	10	31.08.2019 16:21	31.08.2019 16:25	4.0	265	244
561268	YRBMI	737-800	31.08.2019 17:20	31.08.2019 21:30	DY4224	WET	AGP	ARN	31.08.2019 17:20	31.08.2019 17:27	7	31.08.2019 21:10	31.08.2019 21:22	12.0	250	242

Figure 4.1: The data set: relevant columns are marked blue.

4.2.1 Decoding the Airport Combinations by One Hot Encoding

For the columns `DP latacode` and `AR latacode` we have to find a representation that can be interpreted by a neural network. An easy, yet effective way is **One Hot Encoding**. It is used to encode categorical data, such as "flights that departed in Oslo". The idea is to add extra columns for each category, in our case the airports and decode by one if the flight starts at the specific airport and with zero otherwise. The same is done with the arrival airports. This way we have increased the matrix dimension to (546863×410).

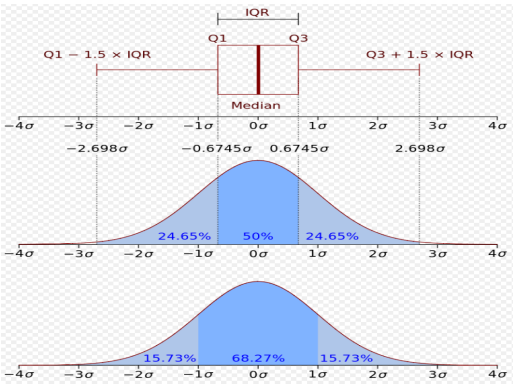
4.2.2 Find and Remove Outliers

An **outlier** describes an observation that differs significantly from other observations in a sample. In the dataset we can find some flights that had block times that exceeded the scheduled block time by far. This can happen due to extraordinary weather conditions or strikes. However, these are exceptions and give no valuable information to the network. They are likely to disturb the network in finding the optimal weights, since high losses $R_E(F)$ are created for a function F, that in general approximates block times well. The network might overfit (see 2.1). Therefore, we detect such outliers and remove them.
In order to do that, we first create a new column denoted by `DiffBlock`, containing the differences between scheduled and actual block time.

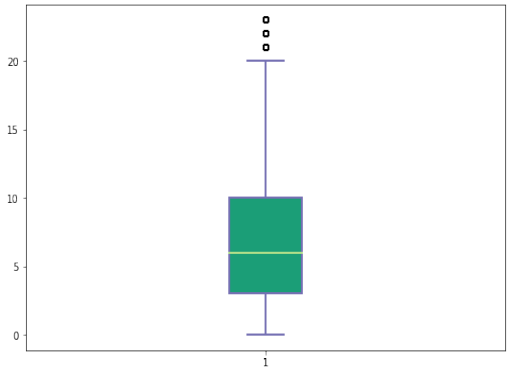
Using **Interquartile Range** we can then detect and remove outliers. The Interquartile range method gives the interval, called the IQR- interval, in which 50% of the difference between scheduled block time and actual block time lie.

Later the interval boundaries are extended by the factor 1.5 and everything that is not included in the extended interval is being removed (see (4.2a)).

Applyinhg the IQR-Method to column `DiffBlock`, we find that, 6% of the flights produce outliers and the IQR interval covers 8 minutes. The extended interval covers values from zero to 23 minutes. This can be visualized by a plot called boxplot (see (4.2b)).



(a) IQR method.



(b) Boxplot of difference between scheduled block times and actual block times.

Figure 4.2: Removing outliers

4.2.3 Spliting and Centering the Data

Since we want to test our predictions for unkown data Z (see (2.0.2)) we first split our data set into two disjoint sets. 80% of the original dataset will be the training set X and the remaining 20% will serve as test set Z. Furthermore, as we could observe in the previous chapter, the optimal choice for a bias vector for the input data is the one that centers the data. Therefore, before processing the data using a Multilayer Perceptron, we center the data.

4.3 Use Multilayer Perceptrons for Regression Problem

We will proceed now by building different types of Multilayer Perceptrons, that we will call models in the further. We want to observe their learning capacity on the training data and evaluate the predictions on the test data.

4.3.1 Build Models

We build four different models.

- `modSimple` a simple model with two hidden layers of dimension 12 and 10.
- `modDeep` a deeper architecture with four hidden layers of dimensions $5, 12, 10, 5$.
- `modWider` a wider model with two hidden layers of dimensions $600, 65$.
- `modPCA` a model with one linear hidden layer of dimension 70 followed by two, non-linear hidden layers of dimensions $12, 10$.

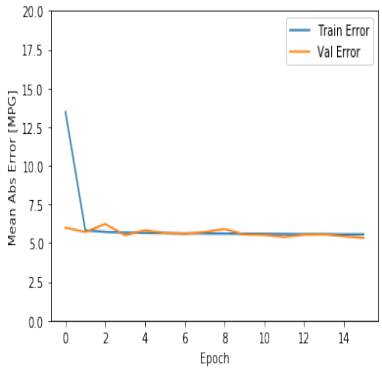
Notice that since we are looking at a regression problem, the output layer must be of dimension one in all models.
As non-linear activation function we chose the *relu function*

$$g(s) = max(0, s)$$

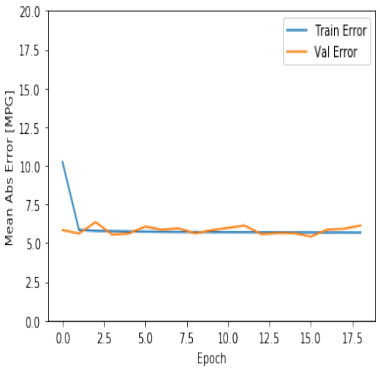
The loss function L is, as in the previous chapter, the mean square error. As an optimizer we chose `Adam`, that is an implementation of a stochastic gradient descent method. In order to avoid overfitting, we will force the model to stop training, if the loss does not improve after 5 training epochs. Additionally, to prevent overfitting, 20% of the training dataset is extracted as validation set that helps to measure the error for unknown values during the training phase. If the mean absolute error for the validation set differs significantly from the train error, the model is overfitting. We observe that the deeper model (see (4.3b)) and the simple model (see (4.3a)) tend to overfit faster.

4.3.2 Train Models

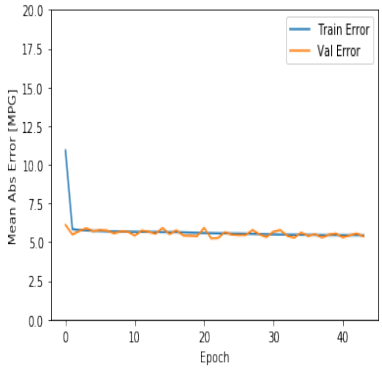
After having built the models we have to train them on our generated training set X. Now we can observe the training capacity of each model. We notice that the



(a) Mean absolute error by epochs for the simple model



(b) Mean absolute error by epochs for the deeper model



(c) Mean absolute error by epochs for the wider model

Figure 4.3: Training error.

wider model (see (4.4c)) learns a bit better than the other two models. It ends after 43 epochs with a mean absolute error of 5.445 minutes , while the simple model (see (4.4a)) ends with a mean absolut training error of 5.562 and the deeper model (see (4.4b)) with 5.682. The better performance of the wider model could be an indicator of the data set providing too little information to the network (see (2.1)).

```
This is a simple Network

.............Training loss and mean square error
        loss  mean_absolute_error  ...  val_mean_squared_error  epoch
11  61.190949             5.588821  ...               61.715286     11
12  61.049165             5.574359  ...               61.011272     12
13  61.063182             5.574674  ...               60.936382     13
14  60.989832             5.565788  ...               61.376331     14
15  60.964922             5.562210  ...               61.314491     15
```

(a) Training progress of the simple model.

```
This is a deeper Network

.................Training loss and mean square error
        loss  mean_absolute_error  ...  val_mean_squared_error  epoch
14  63.244949             5.698143  ...               62.227375     14
15  63.306667             5.693247  ...               63.612152     15
16  63.049147             5.683008  ...               63.355755     16
17  63.084830             5.685726  ...               63.776772     17
18  63.103977             5.682090  ...               65.847076     18
```

(b) Training progress of the deeper model.

```
This is a wider Network

.........................................Training loss and mean square error
        loss  mean_absolute_error  ...  val_mean_squared_error  epoch
39  59.923006             5.452300  ...               60.128525     39
40  59.854354             5.454023  ...               61.250813     40
41  59.936436             5.452287  ...               59.712307     41
42  59.852756             5.447939  ...               59.897202     42
43  59.843633             5.445881  ...               59.829151     43
```

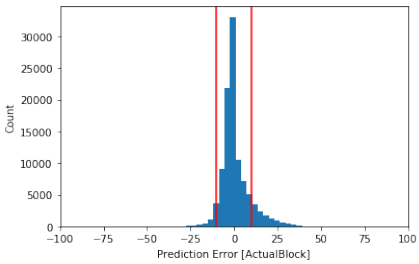
(c) Training progress of the wider model.

Figure 4.4: Training progress.

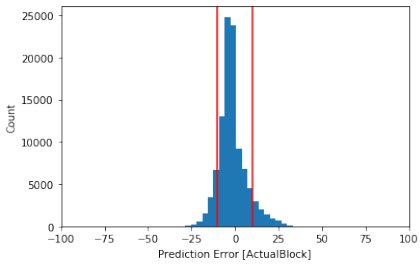
4.3.3 Evaluate Models on Test Data

The final step is to apply the trained models to the test data set Z and evaluate the result. Since we extracted Z from our initial data set, that only includes past flights, we have information about actual block times and can therefore evaluate the predictions, the model did for data that was not seen in the training phase before. We are not only interested in minimizing the prediction error, but also on choosing a model whose predictions of block times tend to be rather too long than too short. Besides that, in general we are interested in choosing a model that predicts block times in a way that the majority is less than 10 minutes too short or too long. Therefore, we need extra evaluation functions that measure the percentage of flights' predicted block times are more than 10 minutes too short and the percentage of flight whose predicted block times deviate from the actual block time by more than 10 minutes. To get a more clear idea of the results, we will also plot the distribution of the error. `Count` describes the amount of flights for which the predictions have the corresponding `Prediction Error`. The red lines indicate the interval of 10 minutes deviation,in which we want the error to be contained in. We notice that the simple model (see (4.6a)) and the wider model(see (4.6c)), although having produced a smaller absolute error, have an error that mostly lies on the positive side. That means, the error comes mostly from block times that were predicted too short.

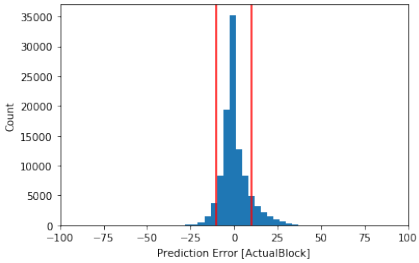
The evaluation functions display the same result. The simple model predicts the block times, such that 14.69% differ by more than 10 minutes from the actual block times and for the wider model we get 15.79%. The deeper model performs significantly worse, with 18.62, but it has the advantage that only 8.62% of the



(a) Distribution of the error, simple model.



(b) Distribution of the error, deeper model.



(c) Distribution of the error, wider model.

Figure 4.5: Distribution of the error.

predictions were too short, while the others perform worse in this regard. Since that prevents causing delays in other flights, for our purpose the deeper model is the best choice.

```
this is the median of error, negative indicates too long, positive too short
-1.121429443359375
this is the average error, negative indicates too long, positive too short
5.328068664283369
this is the percentage of blocktimes whos predictions differmore than +/- 10 minutes from actual block times:
0.14693430727647908
this is the percentage of blocktimes whos predictions exceed by more than + 10 minutes:
0.11209711014018194
```

(a) Evaluation functions, simple model.

```
this is the median of error, negative indicates too long, positive too short
-2.4891281127929688
this is the average error, negative indicates too long, positive too short
6.13202533278638
this is the percentage of blocktimes whos predictions differmore than +/- 10 minutes from actual block times:
0.18625693886052527
this is the percentage of blocktimes whos predictions exceed by more than + 10 minutes:
0.08628890847969929
```

(b) Evaluation functions, deeper model.

```
this is the median of error, negative indicates too long, positive too short
-0.8427009582519531
this is the average error, negative indicates too long, positive too short
5.360972717191436
this is the percentage of blocktimes whos predictions differmore than +/- 10 minutes from actual block times:
0.15791054319289305
this is the percentage of blocktimes whos predictions exceed by more than + 10 minutes:
0.10276779399939936
```

(c) Evaluation functions, wider model.

4.3.4 Conclusion

We have applied Multilayer Perceptrons to a regression problem with relatively few variables. Although the networks were given little information, the approximations were satisfying. A deeper architecture outperformed the other architectures because it has the effect of creating more variance. However, for our purpose, avoiding delays, the deeper network performed best.

Bibliography

References

[1] J.R. M. and H. Neudecker,(1986). **Symmertry,** $0-1$ **matrices and Jacobians** Economic Theory, **2** ,157-190.

[2] K. Hornik, M. Stinchcombe, H. White (1989) **Multilayer Networks are Universal Approximators** Neural Networks, **2** ,359-366.

[3] P. Baldi and K. Hornik, (1988). **Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima,** Neural Networks, **2**, 53-58.

[4] D. Serre, **Matrices**, Springer, 2010.

Sources

[5] S. Geman, E. Bienenstock, R. Doursat, (1986). **Neural Networks and the Bias/Variance Dilemma** Economic Theory, **2** ,157-190.

[6] S. Bosch, **Lineare Algebra**, Springer Verlag, 2001.

[7] P. Petersen, **Linear Algebra**, Springer, 2012.

[8] *Die Zeit.*
`https://www.zeit.de/digital/internet/2016-08/kuenstliche-intelligenz-geschichte-neuronale-netze-deep-learning`
(viewed `January 12, 2020`).