universität
wien

BACHELORARBEIT

Titel der Bachelorarbeit

A Comparison of Machine Learning Algorithms

Verfasser

Thomas Niedermayer

angestrebter akademischer Grad

Bachelor of Science (BSc.)

Wien, im Monat September 2020

Studienkennzahl lt. Studienblatt:	A 01600968
Studienrichtung lt. Studienblatt:	Mathematik
Betreuer:	Assoz. Prof. Dr. Martin Ehler, Privatdoz.

Abstract

Machine learning is the study of methods that learn how to perform certain tasks using data. Common tasks carried out by machine learning algorithms are categorising a data set into a set of classes or predicting a value based on the data. In this thesis four algorithms will be introduced and discussed qualitatively. The differences and similarities between the linear regression, the perceptron, artificial neural networks and convolutional neural networks will be investigated. While they incorporate similar approaches, the artificial neural network is significantly more flexible than the perceptron and the linear regression. Convolutional neural networks are artificial neural networks with special operations and perform exceptionally well in object recognition tasks on image data. In the last chapter a few methods will be evaluated on their ability to categorise skin images into the correct classes of skin disease.

Contents

1 Notation

Unless otherwise specified the following notation will be used. Lower case bold and stylized letters like $\mathbf{z} = (z_1, \ldots, z_n)^\mathsf{T} \in \mathbb{R}^n$ are vectors and single elements of the vector $\mathbf{z}$ will be denoted like this $z_i \in \mathbb{R}$ whereas multiple different vectors can be indexed using bold stylized letters $\mathbf{z}_i \in \mathbb{R}^n$. The same technique of using a different typeface will be used again later on to group other mathematical objects while keeping the notation simple.

2 Linear Regression

This chapter is based on [3]. The aim of linear regression is to estimate the value of a real valued target variable t based on a D-dimensional vector $\mathbf{x}$ of input variables. The simplest form of linear regression provides a model that is an affinely linear function in terms of the input vector $\mathbf{x}$. Small adjustments have to be made to enable the method to represent more complex, nonlinear functions. By the use of so called basis functions one can achieve models that are nonlinear functions of the input vector, which means greater flexibility. At the same time the model remains linear in terms of the parameters that are found in the training process. This gives the method simple analytical properties. While being a basic model with severe limitations, linear regression forms the basis for more complex algorithms.

2.1 The Prediction Function

An **observation** $\mathbf{x}$ is a real-valued D-dimensional vector and $t \in \mathbb{R}$ is called a **target value**. Considering the simplest form of linear regression the prediction function y is a linear combination of the input $\mathbf{x}$ and the real valued vector $\mathbf{w} \in \mathbb{R}^D$ which is referred to as **weight vector** or **weights** plus the **bias** $b \in \mathbb{R}$. The prediction function y is defined as

$$\begin{aligned} y \colon \mathbb{R}^D \times \mathbb{R}^D \times \mathbb{R} &\to \mathbb{R} \\ (\mathbf{x}, \mathbf{w}, b) &\mapsto w_1 x_1 + \cdots + w_D x_D + b. \end{aligned}$$

Extending the model with M basis functions ϕ_i for $i \in \{1, \ldots, M\}$

$$\begin{aligned} \phi_i \colon \mathbb{R}^D &\to \mathbb{R}^M \\ \mathbf{x} &\mapsto \phi_i(\mathbf{x}) \end{aligned}$$

allows the function y to be nonlinear with regard to the input $\mathbf{x}$ while still remaining linear with regard to the parameters $\mathbf{w}$ and b. After adapting the

dimensions of $\mathbf{w}$ and y to the new inputs $\phi_i(\mathbf{x})$ the definitions change to

$$\begin{aligned} y\colon \mathbb{R}^D \times \mathbb{R}^M \times \mathbb{R} &\to \mathbb{R} \\ (\mathbf{x}, \mathbf{w}, b) &\mapsto w_1\phi_1(\mathbf{x}) + \cdots + w_M\phi_M(\mathbf{x}) + b. \end{aligned}$$

It is important to note that the bias is independent of the input $\mathbf{x}$ and allows for models that are affinely linear in terms of $\boldsymbol{\phi}(\mathbf{x})$. During training it is useful to regard bias and weights together as a vector of parameters to be optimised, thus define $\mathbf{v} = (b, w_1, \ldots, w_M)^\mathsf{T}$, so $y(\mathbf{x}, \mathbf{v}) = y(\mathbf{x}, \mathbf{w}, b)$. For a more compact way of writing the equation above the constant real valued function $\phi_0(\mathbf{x}) = 1$ for $\mathbf{x} \in \mathbb{R}^D$ is introduced and $\boldsymbol{\phi} := (\phi_1, \ldots, \phi_M)^\mathsf{T}$ is defined so that

$$y(\mathbf{x}, \mathbf{v}) = \mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}).$$

In this extended model $\mathbf{v}$ is an M+1 dimensional real valued vector and its elements are parameters that are found minimizing the difference between the predicted values and the target values. A possible measure of difference is the mean squared error, or **MSE**, which is defined as

$$\begin{aligned} \mathrm{MSE}\colon \mathbb{R}^{N\times D} \times \mathbb{R}^N &\to \mathbb{R} \\ (\mathbf{X}, \mathbf{t}) &\mapsto \frac{1}{N}\sum_{i=1}^{N}(y(\mathbf{x}_i) - t_i)^2, \end{aligned}$$

for sets $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and $\mathbf{t} = \{t_1, \ldots, t_N\}$ with observation $\mathbf{x}_i \in \mathbb{R}^D$ and corresponding target value $t_i \in \mathbb{R}$ for $i \in \{1, \ldots, N\}$. The factor of $1/N$ is relevant for comparisons between different regression methods or to gauge the performance on different sets of data and is not relevant for optimisation which is why it can be omitted in that regard. The MSE is precisely the error function that is minimised using the expectation maximisation approach used in the following subsection.

2.2 Fitting

The process of finding the optimal parameters in order to minimise the MSE is called **fitting** or **training**. The following is the definition of the distribution of the Gaussian random variable

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}}\exp(-\frac{1}{2\sigma^2}(x-\mu)^2),$$

where $\mu \in \mathbb{R}$ is called the **mean** and $\sigma^2 \in \mathbb{R}$ is called the **variance** which is required to be nonzero.

Linear regression assumes that t is described by a deterministic function $y(\mathbf{x}, \mathbf{v})$ with additive Gaussian noise $\epsilon \in \mathbb{R}$. This means

$$t = y(\mathbf{x}, \mathbf{v}) + \epsilon.$$

More precisely: ϵ is a Gaussian random variable with variance β^{-1} and mean 0. Thus it follows that $p(t|\mathbf{x}, \mathbf{v}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{v}), \beta^{-1})$ where $p(z|B)$ denotes the probability of z given the set of parameters B. Consider the set of N D-dimensional observations $\mathbf{X}$ with corresponding target variables $\mathbf{t}$. Under the assumption that this data is drawn independently from its distribution the following likelihood function can be obtained:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{v}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}).$$

Thus by defining the sum of squares error function for fixed $\mathbf{X}$ and $\mathbf{t}$

$$\begin{aligned} \mathrm{E_D} \colon \mathbb{R}^{M+1} &\to \mathbb{R} \\ \mathbf{v} &\mapsto \frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_n))^2, \end{aligned}$$

the following holds

$$\begin{aligned} \ln p(\mathbf{t}|\mathbf{X}, \mathbf{v}, \beta) &= \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) \\ &= \frac{N}{2}\ln(\beta) - \frac{N}{2}\ln(2\pi) - \beta\,\mathrm{E_D}(\mathbf{v}). \end{aligned}$$

In the next step, $\mathbf{v}$ and β will be determined using a maximum likelihood approach,

$$\begin{aligned} 0 &= \nabla \ln p(\mathbf{t}|\mathbf{X}, \mathbf{v}, \beta) \\ &= \sum_{n=1}^{N}(t_n - \mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_n))\boldsymbol{\phi}(\mathbf{x}_n)^\mathsf{T} \\ &= \sum_{n=1}^{N} t_n\boldsymbol{\phi}(\mathbf{x}_n)^\mathsf{T} - \mathbf{v}^\mathsf{T}(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^\mathsf{T}). \end{aligned}$$

This yields the **normal equations** for the least squares problem. Solving them for $\mathbf{v}$ gives the parameters $\mathbf{v}_{ML}$ yielding the maximum likelihood

$$\mathbf{v}_{ML} = (\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\mathsf{T}\mathbf{t},$$

where $\mathbf{\Phi}$ is given by

$$\mathbf{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \dots & \phi_M(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \dots & \phi_M(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \dots & \phi_M(\mathbf{x}_N) \end{pmatrix}.$$

Further minimising the log likelihood function with respect to β yields

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n=1}^{N} (t_n - \mathbf{v}_{ML}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{x}_n))^2,$$

where $\frac{1}{\beta_{ML}}$ denotes the variance according to the maximum likelihood approach.

3 The Perceptron Algorithm

This chapter is based on [3] and [10].

3.1 The Decision Function

The perceptron is a binary linear discriminant model which means it divides observations in two different classes by the use of a linear function. First discovered in 1962, this algorithm is a landmark achievement in the field of machine learning. In essence the goal is to find a hyperplane that separates all of the training observations according to their class membership. An input vector $\mathbf{x} \in \mathbb{R}^D$ is first transformed by a fixed nonlinear function $\boldsymbol{\phi}$ with M dimensional output to obtain a more flexible model as discussed in the chapter about linear regression. Consider the following function f

$$\begin{aligned} f \colon \mathbb{R} &\to \{-1, +1\} \\ a &\mapsto f(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0. \end{cases} \end{aligned}$$

Let $\mathbf{w} \in \mathbb{R}^M$ and $b \in \mathbb{R}$, then the generalised linear model for the perceptron can then be written as an affinely linear function of $\boldsymbol{\phi}(\mathbf{x})$ fed into f which maps to one of the 2 categories -1 and $+1$

$$\begin{aligned} y \colon \mathbb{R}^D \times \mathbb{R}^M \times \mathbb{R} &\to \{-1, +1\} \\ (\mathbf{x}, \mathbf{w}, b) &\mapsto f(\mathbf{w}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{x}) + b). \end{aligned}$$

To keep the notation uncluttered as seen in Linear Regression define

$$y(\mathbf{x}, \mathbf{v}) = f(\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x})),$$

after extending $\boldsymbol{\phi}$ with $\phi_0 = 1$ and defining the parameters $\mathbf{v} = (b, w_1, \ldots, w_M)$. Let N and D be positive whole numbers, $\mathbf{X}$ a set of N input vectors $\mathbf{x}_i \in \mathbb{R}^D$, $i \in \{1, \ldots, N\}$ and $t_i \in \{-1, +1\}$ the labels for an $\mathbf{x}_i \in \mathbf{X}$. $T = \{(\boldsymbol{\phi}(\mathbf{x}_i), t_i) | i \in \{1, \ldots, N\}\}$ is then called the training set. The function y is used to predict the class membership of an instance $\mathbf{x}_i \in \mathbf{X}$ which is given by t_i. The set of $\boldsymbol{\phi}(\mathbf{x}_i)$ with $t_i = +1$ will be referred to as class A and the set of $\boldsymbol{\phi}(\mathbf{x}_i)$ with $t_i = -1$ as class B. In the training process a vector of real valued parameters $\mathbf{v}$ will be found so that $y(\mathbf{x}_i, \mathbf{v}) = t_i$ for all $i \in \{1, \ldots, N\}$. If such a $\mathbf{v}$ exists T is called **linearly separable**. If this condition is not met convergence of the training algorithm is impossible. This means that there exists no hyperplane such that one class lies on one side of the hyperplane including the plane itself and the other class lies on the other side excluding the hyperplane. The hyperplane separation theorem as seen in [11] states that for two disjoint nonempty closed convex subsets of $\mathbb{R}^N$ A and B, one of which is compact, there exists a nonzero vector $\mathbf{u}$ and real numbers $c_1 < c_2$ such that

$$\mathbf{u}^\mathsf{T}\mathbf{z}_1 < c_1$$

and

$$\mathbf{u}^\mathsf{T}\mathbf{z}_2 > c_2$$

for all $\mathbf{z}_1 \in A$ and $\mathbf{z}_2 \in B$. This guarantees that A and B form a linearly separable training set which can be seen when $b := -c_1$ and $\mathbf{w} := \mathbf{u}$. It will be proven that this is sufficient for the parameters in the perceptron training algorithm to converge to a hyperplane that separates A and B in finitely many steps.

The weights and the bias together can be interpreted as a hyperplane that separates the set $S = \{\boldsymbol{\phi}(\mathbf{x}) | \mathbf{x} \in \mathbf{X}\}$ into classes A and B where the weights define the orientation of the plane and the bias the location for fixed weights. The hyperplane is the set of $\mathbf{z} \in \mathbb{R}^M$ such that $\mathbf{w}^\mathsf{T}\mathbf{z} + b = 0$. This is depicted in Figure 1.

3.2 Training

Like in many machine learning algorithms the goal is to minimise an error function. A naive error function would be the number of misclassified elements. However, in contrast to most other machine learning algorithms the standard

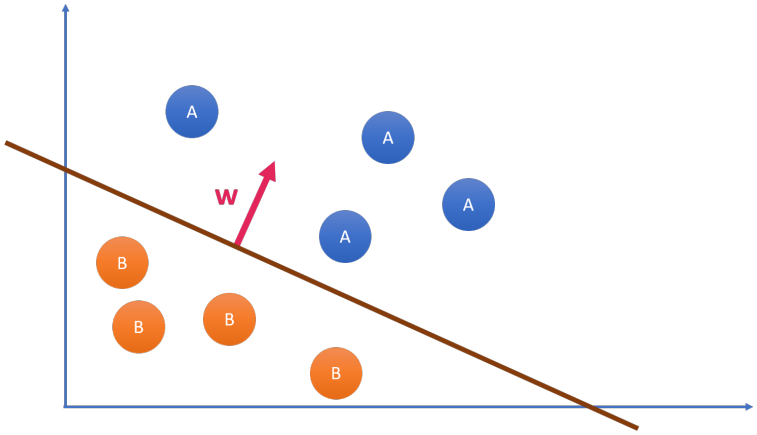

Figure 1: Depiction of a linearly separable training set and a separating hyperplane.

optimisation technique, the gradient descent, cannot be applied here. This is due to the fact, that f is discontinuous in 0 and constant everywhere else, yielding a gradient that is zero which in turn makes progression in the gradient descent algorithm impossible. So instead a different error function known as the **perceptron criterion** is considered. Because of how f is defined, a vector $\mathbf{x}_i$ is predicted to be in class A if and only if $\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i) \geq 0$ and $\mathbf{x}_i$ is predicted to be in class B if and only if $\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i) < 0$. So if $t_i\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i) > 0$ for a certain i, the observation has been classified correctly. Since it would be beneficial for a misclassified observation to come closer to satisfying this condition, $-t_i\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i)$ needs to be minimised for each $i \in \{1, \ldots, N\}$ motivating the perceptron criterion

$$\begin{aligned} \mathrm{E}\colon \mathbb{R}^M &\to \mathbb{R}_0^+ \\ \mathbf{v} &\mapsto -\sum_{i \in \mathcal{M}} \mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i)t_i, \end{aligned}$$

where $\mathcal{M}$ is the set of the indices of all the misclassified instances. If a $\mathbf{v}$ is found such that $E(\mathbf{v}) = 0$ then all instances are classified correctly. The perceptron criterion can be considered as a sum over all instances $\mathbf{x}_i$ $i \in \{1, \ldots, N\}$ where each summand is either zero if the corresponding $\mathbf{x}_i$ is classified correctly or linear with respect to $\mathbf{v}$ otherwise. This makes the perceptron criterion a piecewise linear error function that the stochastic gradient descent can be applied on. This is an iterative algorithm that yields an updated parameter vector in the n-th step as

$$\mathbf{v}^{(n)} = \mathbf{v}^{(n-1)} - \eta\nabla E(\mathbf{v}) = \mathbf{v}^{(n-1)} + \eta\boldsymbol{\phi}(\mathbf{x}_j)y_j,$$

where $\eta \in \mathbb{R}^+$ is called the **learning rate** and the weights and bias are initialised as follows

$$\mathbf{v}^{(0)} = \underbrace{(0, \ldots, 0)^\mathsf{T}}_{\text{M+1 times}},$$

and j is the index of a misclassified element that is used to update the parameters in the n-th step. The algorithm continues until all the observations have been classified correctly. By linearity of $\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i)$ with respect to $\mathbf{v}$ it follows that y is invariant under scalar multiplication of $\mathbf{v}$ with a positive real number, thus η can be set to 1. This concludes in algorithm 1. The following theorem justifies why

Algorithm 1 Perceptron

```
Initialise v = 0
m_total = 0                                   ▷ number of total mistakes made
while TRUE do
    m = 0                  ▷ number of mistakes made for current parameters
    for i ∈ {1, ..., n} do
        if t_i v^T φ(x_i) ≤ 0 then        ▷ If the i-th observation is misclassified
            v ← v + t_i φ(x_i)
            m ← m + 1
            m_total ← m_total + 1
    if m = 0 then ▷ If current parameters classified all observations correctly
        break
```

this algorithm converges after finitely many steps for linearly separable training sets. It will be assumed that all the $\boldsymbol{\phi}(\mathbf{x}_i)$ are in the unit sphere centered at zero. Otherwise they can be scaled down before the application of the algorithm and rescaled afterwards. Since the scaling of the parameter vector $\mathbf{v}$ by a positive real valued vector has no effect on the output of the function y, they will be assumed to have unit length.

Definition: The margin γ of $\mathbf{v}$ is defined as

$$\gamma = \min_{\mathbf{x} \in \mathbf{X}} |\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x})|.$$

Theorem 3.1. *Suppose there exists a* $\mathbf{v} \in \mathbb{R}^{M+1}$ *such that* $t_i\mathbf{v}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i) > 0$ *for* $i \in \{1, \ldots, N\}$. *Then the total number of updates* m_{total} *is bounded by* $1/\gamma^2$.

Proof. Denote j the index of the observation in the current iteration. For the sake of simplicity define the current observation $\mathbf{x} := \boldsymbol{\phi}(\mathbf{x}_j)$ and label $t := t_j$. Suppose the weights are updated for the k-th time. It follows that $t\mathbf{v}^\mathsf{T}\mathbf{x} < 0$

since there would be no update if $\mathbf{x}$ was classified correctly. For the updated weights $\mathbf{v}^{(k)} := \mathbf{v}^{(k-1)} + t\mathbf{x}$ the following inequality holds

$$\begin{aligned} \mathbf{v}^{(k)} &= {\mathbf{v}^{(k-1)}}^{\mathsf{T}}\mathbf{v}^* \\ &= (\mathbf{v}^{(k-1)} + t\mathbf{x})^{\mathsf{T}}\mathbf{v}^* \\ &= {\mathbf{v}^{(k-1)}}^{\mathsf{T}}\mathbf{v}^* + t\mathbf{x}^{\mathsf{T}}\mathbf{v}^* \\ &\geq {\mathbf{v}^{(k-1)}}^{\mathsf{T}}\mathbf{v}^* + \gamma. \end{aligned} \tag{3.1}$$

Also consider the following boundary

$$\begin{aligned} \left\|\mathbf{v^{(k)}}\right\|^2 &= (\mathbf{v^{(k-1)}} + t\mathbf{x})^{\mathsf{T}}(\mathbf{v}^{(k-1)} + t\mathbf{x}) \\ &= {\mathbf{v}^{(k-1)}}^{\mathsf{T}}\mathbf{v}^{(k-1)} + \underbrace{2t{\mathbf{v}^{(k-1)}}^{\mathsf{T}}\mathbf{x}}_{<0} + \underbrace{t^2\mathbf{x}^{\mathsf{T}}\mathbf{x}}_{\in[0,1]} \\ &\leq {\mathbf{v}^{(k-1)}}^{\mathsf{T}}\mathbf{v}^{(k-1)} + 1. \end{aligned} \tag{3.2}$$

After a total of m_{total} updates the following inequalities hold because of 3.1 and 3.2 respectively

$$\begin{aligned} {\mathbf{v}^{(m_{\text{total}})}}^{\mathsf{T}}\mathbf{v}^* &\geq m_{\text{total}}\gamma \\ \left\|\mathbf{v}^{(m_{\text{total}})}\right\|^2 &\leq m_{\text{total}}. \end{aligned}$$

Thus it can be inferred that

$$m_{\text{total}}\gamma \leq {\mathbf{v}^{(m_{\text{total}})}}^{\mathsf{T}}\mathbf{v}^* \leq \left\|{\mathbf{v}^{(m_{\text{total}})}}^{\mathsf{T}}\mathbf{v}^*\right\| = \left\|{\mathbf{v}^{(m_{\text{total}})}}^{\mathsf{T}}\right\| \leq \sqrt{m_{\text{total}}}.$$

Therefore, we have

$$m_{\text{total}} \leq \frac{1}{\gamma^2}.$$

□

The example of a training set seen in Figure 2 is not linearly separable and demonstrates that linear separability is a strong precondition. In addition to this restriction of the algorithm, the perceptron does not yield probabilistic outputs and does not readily generalize to more than 2 classes. An even greater limitation is that it depends like the linear regression on a linear combination of fixed basis functions. All of these limitations are eliminated in the multilayer perceptron algorithm.

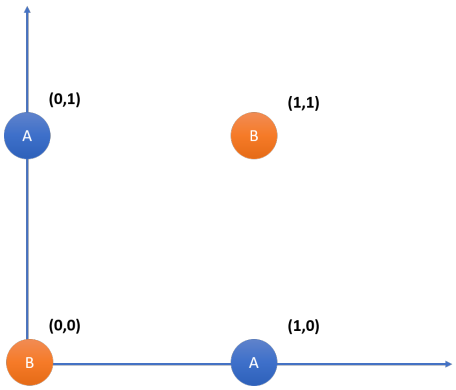

Figure 2: Illustration of the XOR problem, named after the logical XOR operator, that cannot be modelled by the perceptron. Group A representing the logical True and Group B representing the logical False.

4 Neural Networks

This chapter is based on [3] and [6]. This section will concentrate on one type of architecture of neural networks, the **feed-forward neural network**, also known by the name **multilayer perceptron**.

4.1 Activation Functions

In Section 3 about the perceptron the function f mapped any possible value to $\{-1, 1\}$ depending on the predicted category. In neural networks this part of the perceptron is replaced by a continuous nonlinear function which enables the algorithm to model nonlinear functions. This is called an activation function. Two important examples will be compared here. The **logistic sigmoid** activation function is defined by

$$\begin{aligned} \sigma &\colon \mathbb{R} \to (0,1) \\ a &\mapsto \sigma(a) = \frac{1}{1+e^{-a}}. \end{aligned}$$

The rectified linear unit, or **ReLU** function was proposed by Nair and Hinton in 2010 and has become the prevalent activation function since then

$$\begin{aligned} \mathrm{ReLU} &\colon \mathbb{R} \to \mathbb{R}_0^+ \\ a &\mapsto \mathrm{ReLU}(a) = \begin{cases} 0 & a \leq 0 \\ a & a > 0. \end{cases} \end{aligned}$$

Because of the importance of derivatives in the optimisation of error functions and thus in the training process of neural networks, the undefined derivative in 0 has to be dealt with. The derivative of the ReLU in zero is often defined to be 0, 1 or 1/2. The choice does not have grave consequences since inputs that are exactly 0 are rare.
One advantage of the ReLU is that it has improved computation speed compared to the logistic sigmoid function since there are no exponentials or divisions to calculate. A significant disadvantage of the logistic sigmoid function is that for very large and very small inputs, the gradient flattens which is called the vanishing gradient problem. Similar problems can be observed in the behaviour of the ReLU as it has a flat gradient when the input a is smaller than 0. This poses difficulties in the optimisation of the weights. Variants of the ReLU not covered here aim to remedy this weakness. Another problem that can arise with the use of the ReLU is that the activation function can blow up since the ReLU is not bounded like the logistic sigmoid function and values can accumulate and become very large in deeper layers of the network. Despite these downsides remaining with the ReLU, in practice the rectified linear unit shows a better convergence performance than the logistic sigmoid function, see [5].

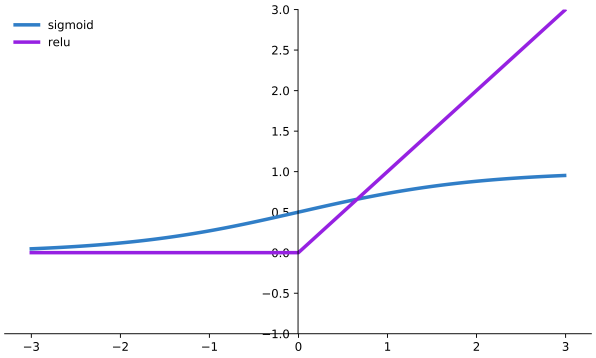

Figure 3: Depiction of the ReLU and the logistic sigmoid activation function.

4.2 Feed-Forward Neural Networks

The following is an introduction to the **fully connected** feed-forward neural network. Figure 4 illustrates such a fully connected neural network and the weights can be considered as connections between the vertices of the network which are called **neurons** and will be introduced shortly. Note that if arbitrary weights in the following algorithm were set to zero permanently and were not subject to optimisation this network would not be considered fully connected

anymore because certain links between neurons would be missing. In the perceptron algorithm the prediction function y was defined as

$$y(\mathbf{x}, \mathbf{w}, b) = f(\mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}) + b).$$

In addition to the replacement of the discrete function f from the perceptron with an activation function, in the neural network each basis function $\boldsymbol{\phi}_j$ will be defined to be a nonlinear function of a linear combination of the input and the weights plus the bias where the parameters are adjustable and optimisable by a training process. Neural networks achieve this by defining these ϕ_j in the same manner as y was defined. This leads to the definition of the **activations** of the first layer $\mathbf{a}^{(1)}$. First, define the equivalents to the weights $\mathbf{w}$ and b in the definition of y. For neural networks these are the weight matrix of the first layer $\mathbf{W}^{(1)} \in \mathbb{R}^{M_1 \times M_0}$ and the bias vector $\mathbf{b}^{(1)} \in \mathbb{R}^{M_1}$. For an input of the first layer $\mathbf{x}^{(1)} \in \mathbb{R}^{M_0}$ the activation of the first layer can now be defined as

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{x}^{(1)} + \mathbf{b}^{(1)}.$$

The right superscript signifies the layer of the variables. Layers are basic building blocks of a neural network. In the next step each $a_j^{(1)}$ for $j \in \{1, \ldots, M_1\}$ is transformed, using an activation function f and the output of the first layer is obtained

$$x_j^{(1)} = f(\mathbf{a}_j^{(1)}).$$

The operation of linear combinations, here written as a matrix multiplication plus a bias to form an affinely linear function, followed by an activation function, is called a **neuron**. Note that $\mathbf{x}^{(1)} = (x_1^{(1)}, \ldots, x_{M_1}^{(1)})$ in the second layer plays the same role as $\mathbf{x}$ in the perceptron. This shows how the neural network is a more flexible algorithm, as $\boldsymbol{\phi}$ is not set, but found by optimisation. In the first layer there are M_1 neurons. After application of the activation function, the idea of creating a nonlinear function of a linear combination of the input and weights plus bias is iterated and the activations of the second layer are defined by

$$\mathbf{a}^{(2)} = \mathbf{W}^{(2)}\mathbf{x}^{(2)} + \mathbf{b}^{(2)},$$

where $\mathbf{W}^{(2)} \in \mathbb{R}^{M_2 \times M_1}$, $\mathbf{b}^{(2)} \in \mathbb{R}^{M_2}$ and M_2 denotes the number of neurons in the second layer. Next, the output of the second layer is defined as

$$x_j^{(2)} = f(\mathbf{a}_j^{(2)}),$$

for $j \in \{1, \ldots, M_2\}$. Next, an arbitrary number of linear combinations followed by an activation function in the manner seen above can follow. For the ℓ-th layer

define the number of neurons $M_\ell \in \mathbb{N}$, the weight matrix $\mathbf{W}^{(\ell)} \in \mathbb{R}^{M_\ell \times M_{\ell-1}}$ and the bias $\mathbf{b}^{(\ell)} \in \mathbb{R}^{M_\ell}$. Then the activations of the ℓ-th layer are defined as

$$\mathbf{a}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{x}^{(\ell)} + \mathbf{b}^{(\ell)}.$$

The outputs of the ℓ-th layer are then defined as

$$x_j^{(\ell)} = f(\mathbf{a}_j^{(\ell)}),$$

for $j \in \{1, \ldots, M_\ell\}$. In the end there has to be an activation function that yields the final output. In this context one **layer** consists of an output yielded by one iteration of matrix multiplication with the weights plus the bias and application of an activation function. In the simplest case there is only one so called **hidden layer** which is an output of an activation function that is not the final output. This is the case where $\mathbf{a}^{(2)}$ is only followed by one final activation function. $\mathbf{x}^{(1)}$ would form this hidden layer. A visualisation of such a simple neural network can be seen in Figure 4. There the elements of the weight matrix for each layer represent connections between neurons and $w_{mn}^{(i)}$ denotes the element in the m-th row and the n-th column of the i-th weight matrix and $b_n^{(i)}$ denotes the n-th entry of the bias vector of the i-th layer. As a neural network is a function of all the weights and biases denoted $\mathbf{V}$ and the input $\mathbf{x}$ the entire prediction function can be denoted as $y(\mathbf{x}, \mathbf{V})$. Note that $\mathbf{V}$ needs to contain all the weights and biases for all the layers of the network.

4.3 Training

First comes a sketch of the training process of neural networks for regression as the principle has been covered in the fitting of a linear regression model. For regression the final activation function is the identity.
Consider the set of N D-dimensional inputs $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with corresponding target variables $\mathbf{t} = \{t_1, \ldots, t_N\}$. If the task of the neural network is the regression of a single variable the minimisation of the error function is the same as maximising the likelihood function given by

$$p(t|\mathbf{x}, \mathbf{V}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{V}), \beta^{-1}),$$

following the steps seen in the fitting of linear regression. The difference to linear regression is that the sum of squares error function that makes its appearance in the course of the maximum likelihood approach is not necessarily convex due to the nonlinearity of y that is introduced by the activation functions. As a result finding the global minimum is not trivial and thus the general approach is to

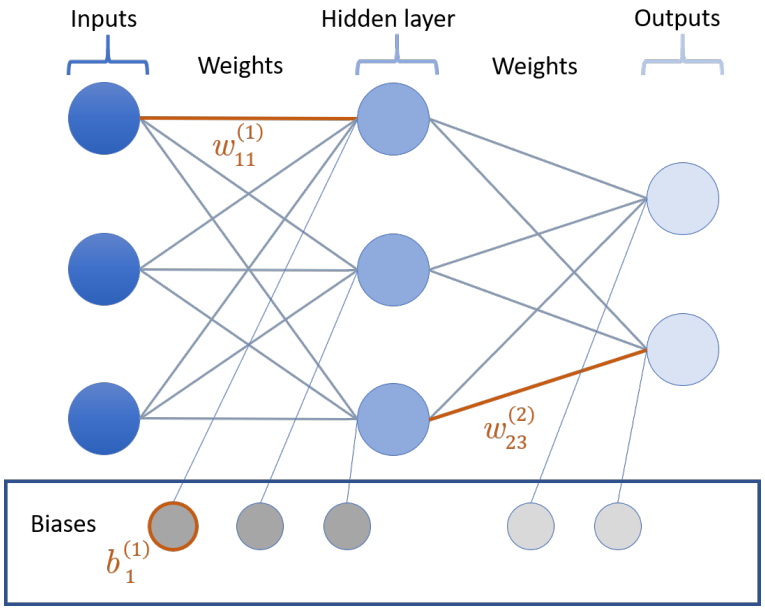

Figure 4: Schematic overview of the architecture of a simple, fully connected feed forward neural network. Single parts highlighted to show the relationship to the notation introduced. In this case $M_0 = M_1 = 3$, $M_2 = 2$. On the left the network begins with the inputs. In the hidden layer nodes, the incoming edges are multiplied by their corresponding input values and summed. The bias is added and an activation function is applied which yields the values that are represented by the hidden layer nodes. This is done for all nodes in the hidden layer and the process is iterated until the final layer which is producing an output.

find a local minimum by the use of the gradient descent algorithm. The function that is subject to minimization is again the sum of squares error function

$$E(\mathbf{V}) = \frac{1}{2} \sum_{n=1}^{N} \|y(\mathbf{x}_n, \mathbf{V}) - t_n\|^2.$$

This error function maps the weights onto a real value and acts as an indicator of how accurate the prediction is. This also holds for all the following error functions. Next, consider the case of binary classification where $t = 0$ denotes class A and $t = 1$ class B. Let the output layer consist of a single node and a be the final activation that is fed to the logistic sigmoid function to yield the output $y(\mathbf{x}, \mathbf{V})$. Assume this output to be the conditional probability that A is the correct label for the input $\mathbf{x}$ denoted $p(A|\mathbf{x})$ and likewise $1 - y(\mathbf{x}, \mathbf{V})$ to be the conditional probability $p(B|\mathbf{x})$. The conditional distribution of t, given the

input $\mathbf{x}$ is given by

$$p(t|\mathbf{x}, \mathbf{V}) = y(\mathbf{x}, \mathbf{V})^t (1 - y(\mathbf{x}, \mathbf{V}))^{1-t}.$$

It follows the likelihood L of N independent tuples of input and output

$$L(\mathbf{V}) = \prod_{n=1}^{N} y(\mathbf{x}_n, \mathbf{V})^{t_n} (1 - y(\mathbf{x}_n, \mathbf{V}))^{1-t_n}.$$

Taking the negative logarithm yields the **cross-entropy** error function

$$E(\mathbf{V}) = -\sum_{n=1}^{N} t_n \ln y(\mathbf{x}_n, \mathbf{V}) + (1 - t_n) \ln(y(1 - \mathbf{x}_n, \mathbf{V})).$$

Lastly, consider the multiclass classification problem that aims to assign an observation to one of K mutually exclusive classes $C_1, \ldots, C_K$. The target variables $t_k \in \{0, 1\}$ for $k \in \{1, \ldots, K\}$ are **one-hot encoded** which means if an observation belongs to C_i for a fixed i, then $t_i = 1$ and $t_j = 0$ for $j \in \{1, \ldots, K\} \setminus \{i\}$. The final activation function is the **softmax** activation function and ensures that all elements of the output are between 0 and 1 and the sum of the outputs is 1. For the k-th output it represents the conditional probability that C_k is the correct class given input $\mathbf{x}$ and parameters $\mathbf{V}$ and is denoted by

$$p(t_k = 1|\mathbf{x}, \mathbf{V}) = y_k(\mathbf{x}, \mathbf{V}) = \frac{\exp a_k(\mathbf{x}, \mathbf{V})}{\sum_{j=1}^{K} \exp a_j(\mathbf{x}, \mathbf{V})},$$

where the vector $a_k(\mathbf{x}, \mathbf{V})$ denotes the final activations. By the independence of N tuples of input and output the negative log-likelihood function can be formed resulting in the **cross-entropy** error function for the multiclass classification problem

$$E(\mathbf{V}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(\mathbf{x}_n, \mathbf{V}),$$

where $t_{nk} \in \{0, 1\}$ denotes the class membership of the n-th observation for C_k.

5 Convolutional Neural Networks

This chapter is based on [3], [7] and [8]. Convolutional neural networks comprise a class of algorithms that currently achieve state of the art results in many image classification and object recognition tasks. Some of the most prominent applications of machine learning are based on this set of algorithms including

autonomous vehicles and facial recognition.
One key issue with the multilayer perceptron is, that it has difficulties detecting objects that occured in the training data, but are merely shifted to another location in the image. While it is possible for multilayer perceptrons to carry out tasks like object recognition, augmented training data is required that simulates shifts in the image which gives rise to new challenges as well as increasing resources required for training. This problem is remedied in the convolutional neural network by introducing convolutions.

5.1 Convolutional Layers

Convolutional neural networks are characterised by containing convolutional layers. All parts of such a layer are described here.

Let T be the set of functions from $\mathbb{Z}^2$ to $\mathbb{R}$. Next, the cross-correlation function will be defined for a **2-dimensional input** $X \in T$ and a **2-dimensional filter** $W \in T$. This input X could represent an $m \times n$ gray scale image if $X \in \{0, \dots, 255\}$ for $(a, b) \in \{0, \dots, m\} \times \{0, \dots, n\}$ and $X(a, b) = 0$ otherwise. Now let $J = X * W$, the cross-correlation of X and W, be

$$\begin{aligned} J &\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{R} \\ (a, b) &\mapsto J(a, b) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} X(a+i, b+j) W(i, j). \end{aligned}$$

Now consider a **3-dimensional input** of length n, that is a sequence $\mathbf{X}$ of n 2-dimensional inputs X_i, so $\mathbf{X} = (X_1, \dots, X_n) \in T^n$ and a 3-dimensional filter of length n $\mathbf{W}$ which is a sequence of n 2-dimensional filters W_i, so $\mathbf{W} = (W_1, \dots, W_n) \in T^n$. A sequence of 3 2-dimensional inputs could represent a red-green-blue (RGB) color image. The n elements of the three dimensional input and filter can be referred to as **channels**. Channel-wise application of the cross-correlation plus a bias yields the convolution function:

$$\begin{aligned} \mathrm{conv} &\colon T^n \times T^n \times \mathbb{R} \to (\mathbb{Z}^2 \to \mathbb{R}) \\ (\mathbf{X}, \mathbf{W}, b) &\mapsto \mathrm{conv}(\mathbf{X}, \mathbf{W}, b) = \sum_{i=1}^{n} X_i * W_i + b. \end{aligned}$$

In an implementation, $\mathbf{X}$ and $\mathbf{W}$ would be represented by 3-dimensional tensors. $\mathbf{W}$ and b are subject to optimisation. The convolution function has an equivalent in feed-forward neural networks. In the definition of the activations one can also see a product of the weights and input plus a bias. Similar to the process seen in feed-forward neural networks, the convolution function is followed by an

activation function. So the output of the convolution function can be regarded as an activation. Let the length of the first input and filter be $k^{(0)} \in \mathbb{N}$ and the number of filters for the first layer be $k^{(1)} \in \mathbb{N}$. For a 3-dimensional input $\mathbf{X}^{(1)}$ of length $k^{(0)}$, $k^{(1)}$ 3-dimensional filters $\mathbf{W}_i^{(1)}$ of length $k^{(0)}$ and a bias $b^{(1)} \in \mathbb{R}$ the activations of the first layer can be defined as

$$A_i^{(1)} = \text{conv}(\mathbf{X}^{(1)}, \mathbf{W}_i^{(1)}, b^{(1)}),$$

for $i \in \{1, \dots, k^{(1)}\}$. Let f be an activation function, then

$$X^{(2)} = (Y_1^{(1)}, \dots, Y_{k^{(1)}}^{(1)}),$$

where $Y_i^{(1)}$ can be defined pointwise:

$$\begin{aligned} Y_i^{(1)} \colon \mathbb{Z}^2 &\to \mathbb{R} \\ (a, b) &\mapsto Y_i^{(1)}(a, b) = f(A_i^{(1)}(a, b)). \end{aligned}$$

This process can be iterated arbitrarily often, as $X^{(2)}$ is again a 3-dimensional input. The ℓ-th iteration represents the ℓ-th layer of the convolutional neural network and its components can be defined as follows. Firstly assume $k^{(\ell)}$ 3-dimensional filters $\mathbf{W}_i^{(\ell)}$ of length $k^{(\ell-1)}$ and the output of the $(\ell-1)$-th layer $\mathbf{X}^{(\ell)}$ which also has length $k^{(\ell-1)}$. The following are the activations of the ℓ-th layer for $i \in \{1, \dots, k^{(\ell)}\}$

$$A_i^{(\ell)} = \text{conv}(\mathbf{X}^{(\ell)}, \mathbf{W}_i^{(\ell)}, b^{(\ell)}).$$

Then the output $\mathbf{X}^{(\ell+1)}$ of the ℓ-th layer is defined by

$$X^{(\ell+1)} = (Y_1^{(\ell)}, \dots, Y_{k^{(\ell)}}^{(\ell)}),$$

where $Y_i^{(\ell)}$ can be defined pointwise:

$$\begin{aligned} Y_i^{(\ell)} \colon \mathbb{Z}^2 &\to \mathbb{R} \\ (a, b) &\mapsto Y_i^{(\ell)}(a, b) = f(A_i^{(\ell)}(a, b)). \end{aligned}$$

Convolutions capture local information and are able to extract distinct features like edges and curves. In deeper networks these features can be used to extract larger, more complex structures.

5.2 Regularisation

Overfitting happens when the model is complex enough to adjust its weights for specific training samples and so it performs well on training data without generalising. Efforts to prevent overfitting are called **regularisation**. The techniques described here are often used as regularisation after a convolutional layer of a convolutional neural network.
After the last step of the convolutional layer downsampling can be applied. This is for regularisation and reducing the number of weights to be trained and thus improving performance and computational cost. Here, only the **max-pooling** method will be considered. First define the length and width of the pool $K \in \mathbb{N}$ and $L \in \mathbb{N}$. For a 3-dimensional input $\mathbf{X}$ of length I max-pooling can be defined point-wise for each channel X_i

$$\begin{aligned}\text{maxpooling}_2 \colon T \times \mathbb{N} \times \mathbb{N} &\to T \\ (X_i, K, L) &\mapsto \text{maxpooling}_2(X_i, K, L),\end{aligned}$$

where for all $(a, b) \in \mathbb{Z}^2$

$$\text{maxpooling}_2(X_i, K, L)(a, b) = \max_{\substack{1 \le k \le K \\ 1 \le l \le L}} X_i((a-1)K + k, (b-1)L + l).$$

So max-pooling for $\mathbf{X}$ can be defined as

$$\begin{aligned}\text{maxpooling} \colon T^I \times \mathbb{N} \times \mathbb{N} &\to T^I \\ (\mathbf{X}, K, L) &\mapsto \text{maxpooling}(\mathbf{X}, K, L) \\ &= (\text{maxpooling}_2(X_i, K, L))_{i \in \{1, \dots, I\}}.\end{aligned}$$

Note that for a concrete implementation where only a finite subset of $\mathbb{Z}^2$ is considered for channels of the input, this method makes the input shrink.
Finally, there is one more operation to combat the network's tendency to overfit, called **dropout**. First, let $\mathbf{B}$ be a 3-dimensional input where each channel B_i maps each $(a, b) \in \mathbb{Z}^2$ to 1 with probability p and to 0 with probability $1 - p$. Dropout can now be defined as

$$\begin{aligned}\text{dropout} \colon T^I \times (0, 1) &\to T^I \\ (\mathbf{X}, p) &\mapsto \text{dropout}(\mathbf{X}, p),\end{aligned}$$

where for every $i \in \{1, \dots, I\}$ and $(a, b) \in \mathbb{Z}^2$

$$\text{dropout}(\mathbf{X}, p)_i(a, b) = B_i(a, b) X_i(a, b).$$

This can make a neural network more robust because in the training process it learns how to make the right predictions even when there is a chance for information to be lost. Dropout is only applied during training and not while predicting. As a part of a neural network max-pooling and dropout can be referred to as layers.

A common architecture for convolutional neural networks is to send the input through a number of triples of convolutional layer, max-pooling layer and dropout layer in this order to extract features and finish the network with fully connected layers. This approach will be tested in the next chapter.

6 Application to Skin Disease Image Data

The following is an example of the application of machine learning to a real world problem. Skin diseases such as skin cancer are often more easily treatable when diagnosed in early stages which motivates the research of screening methods that rely on technology instead of scarce professionals to provide better healthcare for a wider range of the population. The goal is to predict the type of skin disease based on a photograph of the suspicious region of skin and find out which methods yield the best results.

6.1 The Data Set

This section is based on [4]. The data set used for learning is the HAM10000 (see [9]). It contains seven different conditions linked to pigmented skin lesions: Actinic keratosis and intraepithelial carcinoma (AKIEC), basal cell carcinoma (BCC), benign keratosis-like lesions (BKL), dermatofibroma (DF), melanoma (MEL), melanocytic nevi (NV) and vascular lesions (VASC). The images are taken using a dermatoscope, which reveals more information about the skin region than a regular camera because structures beneath the skin are revealed better. Examples of skin lesions drawn from the HAM10000 are seen in Figure 5. The distribution of the 10015 images in the different classes is found in Figure 6 and shows that HAM10000 is a heavily skewed dataset which poses problems in the learning process of neural networks. As the error function is minimised it is likely that the gradient descent happens along a slope that is beneficial for recognising the class containing the most instances referred to as **majority class**, ignoring classes with a low count of images referred to as **minority class**, as they do not have as large of an impact on the error function. This leads to poor recognition performance on the minority classes which is detrimental in healthcare applications, since the minority classes often represent

diseases as they are less common than the normal, healthy state. The method chosen to address this problem is **random oversampling** or **ROS** which means repeatedly drawing from the minority classes until all classes have the same number of instances. It will be evaluated whether this method can improve the performance of the models. An apparent shortcoming of ROS is that it might promote overfitting on the images of minority classes. Additionally, information about the distribution of the training data is lost and cannot be utilised by the model anymore.

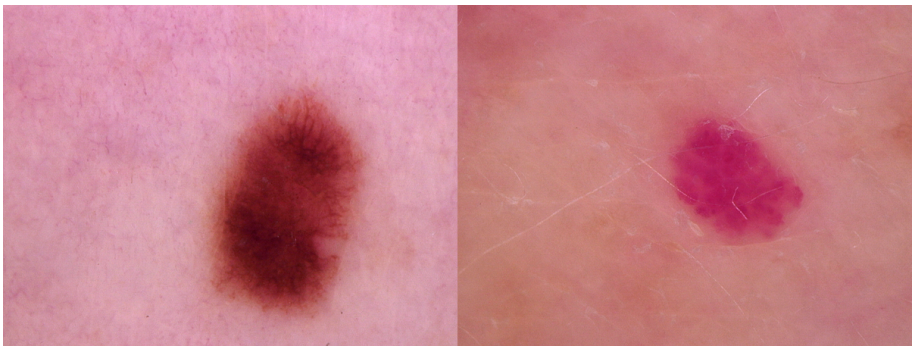

Figure 5: Two observations from the HAM10000 dataset. (NV on the left and VASC on the right)

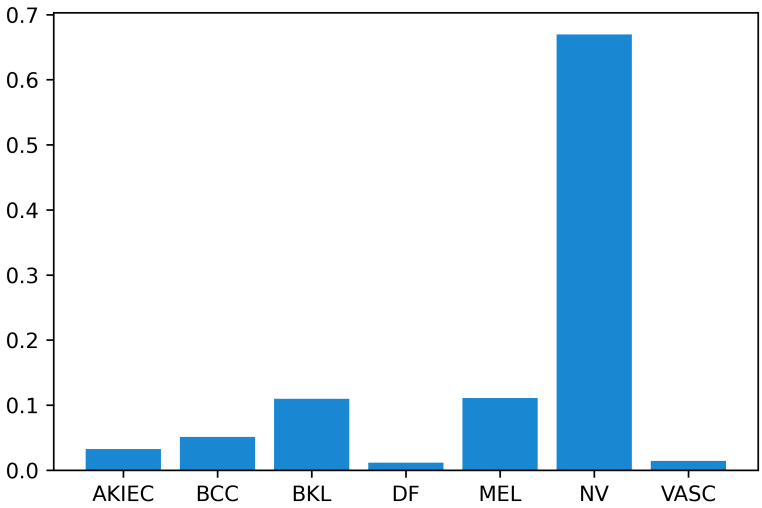

Figure 6: Distribution of the HAM10000 dataset.

6.2 Setup and Metrics

A common metric for the performance of a classification algorithm is the **accuracy** which is the percentage of correctly predicted instances. The skewness of the initial data requires additional metrics for the determination of performance.

The 7 classes of skin lesions AKIEC, BCC, BKL, DF, MEL, NV and VASC will be called $C_1, \ldots, C_7$ respectively. For an observation $\mathbf{x}$ and parameters $\mathbf{V}$ the output of the neural network $y(\mathbf{x}, \mathbf{V})$ will be a 7 dimensional vector, with ℓ_1 norm 1. The index of the output with the maximum value signifies the predicted class. The function, that takes $\mathbf{x}$ and parameters $\mathbf{V}$ as input and yields this index will be denoted $y'(\mathbf{x}, \mathbf{V}) = \text{argmax}(y(\mathbf{x}, \mathbf{V}))$.

An observation $\mathbf{x}$ is correctly classified by the prediction function y if its class label is $t \in \{1, \ldots, 7\}$ and $y'(\mathbf{x}, \mathbf{V}) = t$, otherwise the observation is called classified incorrectly or misclassified. Let N be the number of predictions and N_{correct} the number of observations that were correctly classified then the accuracy is defined as

$$\text{Accuracy} = \frac{N_{\text{correct}}}{N}.$$

The following terms are defined for a specific class C_i. Let $\mathbf{x}$ be an observation belonging to class C_i. If $y'(\mathbf{x}, \mathbf{V}) = i$, then $\mathbf{x}$ is a **true positive** and otherwise it is a **false negative**. Let $\mathbf{x}$ belong to a class other than C_i. If $y'(\mathbf{x}, \mathbf{V}) = i$, then $\mathbf{x}$ is a **false positive** and otherwise it is a **true negative**. Now it is possible to define the number of true positives TP_i, false negatives FN_i, false positives FP_i and true negatives TN_i for class C_i.
The following metrics can now be defined for C_i:

$$\text{Precision}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i},$$

$$\text{Recall}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i},$$

and

$$\text{F}_i = 2\frac{\text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}.$$

So the **precision** estimates the likelihood that the result is true if the algorithm's prediction is class C_i. The **recall** estimates the likelihood that an element of class C_i is predicted to belong to class C_i. The F score is the harmonic mean of

precision and recall. Averaging over K classes yields the **macro metrics**

$$\text{Precision} = \frac{1}{K}\sum_{i=1}^{K}\text{Precision}_i,$$

$$\text{Recall} = \frac{1}{K}\sum_{i=1}^{K}\text{Recall}_i,$$

and

$$\text{F} = \frac{1}{K}\sum_{i=1}^{K}\text{F}_i\,.$$

For the following basic experiments, the data will be split into approximately 80% training data and 20% validation data to check whether the model has learned to generalise beyond the training data.

6.3 Basic Experiments

In this section it is studied how different basic neural network architectures perform on the HAM10000 dataset.

A **naive classifier** will be defined. It classifies every observation as the majority class NV. This makes sense for such an unbalanced dataset because as Figure 6 visualises a classifier that classifies everything as NV would be correct about 68% of the time. This naive classifier will serve as a baseline for all of the metrics.

Next, a feed-forward neural network called N_1 with one hidden layer with 128 neurons will be trained on the training dataset. The activation function for the hidden layer is the ReLU and for the output layer the softmax activation function. Without data augmentation that involves rotating and shifting the image in its frame, this is expected to be able to learn the exact data it was trained on but to generalise poorly, because the weights are dependent on the exact pixel. Small differences in the image could result in vastly different results. Since the input is $450 \cdot 600 \cdot 3 = 810.000$ elements large, the number of weights to be tuned is $(810.000 + 1) \cdot 128 + (128 + 1) \cdot 7 = 103.681.031$ which results in a model that is more than 1 gigabyte in size. While 100 million is multiple times as many weights as the convolutional neural networks will have, there are no convolutions which are computationally more expensive per weight.

The second neural network architecture called N_2 is a convolutional neural network with 4 identical triples of convolution, max-pooling and dropout followed by a fully connected layer holding 512 neurons and finally a fully connected layer

yielding the scores for the 7 classes by the use of the softmax activation function. For all of the convolutional layers, the ReLU activation function and 10 filters are used. The max-pooling uses pools of the dimension 3x3 and the chance for ignoring a value in the dropout layer is 20%. Between the convolutional layers and the fully connected layers, the tensor yielded by the last convolution is **flattened**, which means the elements are ordered in a vector. Since the neural network has no a priori preference for certain elements of this vector, the order in which the elements of the tensor are filled into the vector is not important but has to remain the same once defined.

The idea behind this architecture is to use the convolutions and a standard pool-size of 3x3 to extract 350 features before the interpretation of these features using fully connected layers. More features could potentially extract more structures in the images, but too many could introduce features without much information content. This makes the model more susceptible to overfitting because there are too many free weights, which could be used to target specific training observations. Using 350 features, the model should be resistent to overfitting. Another benefit of an architecture resembling a bottleneck like this is, that the flattened layer contains fewer neurons, resulting in significantly fewer weights to be trained, since each element in the flattened layer connects to 512 fully connected neurons.

The third architecture called N_3 is identical to N_2 except for the number of filters used which is increased to 64 per convolutional layer and a reduction of the dropout rate to 10%. Running the risk of overfitting, this architecture has the potential to extract more information from the images because less dropout is applied and 2240 instead of 350 features are extracted. The architecture of N_3 is visualised in Figure 7 and contains 1.363.239 weights to be tuned. Table 1 shows the performance of the different methods using the metrics defined above. ROS is tested on N_2 and N_3 only, because N_1 shows poor performance with the initial data and has higher computational cost. The winner in terms of F Score is N_3 without ROS. Its accuracy is also significantly better than that of a naive classifier that predicts all observations as the mode class. The reason its F score is superior to all the others is that it achieves a better precision which means positive predictions for a certain class are more likely to be true. In terms of accuracy N_1 performed hardly better than the naive classifier. It seems like it only learned the distribution of the training set resulting in an accuracy of 68.5%. N_2 performs better in terms of accuracy without ROS but the F score is better when using ROS. The difference in accuracy can be attributed to the fact that the neural network learns to leverage the distribution of the training set. When trained using ROS, the classifier assigns classes under the assumption

that all classes are equally likely to occur and thus only features in the image are used for classification without any regard for their real prior distribution. Under the consideration of Bayes' theorem it makes sense, that a model that is trained on a training set with the same prior distribution as the validation set performs better. The same phenomenon can be seen in regard to N_3 and its ROS counterpart.

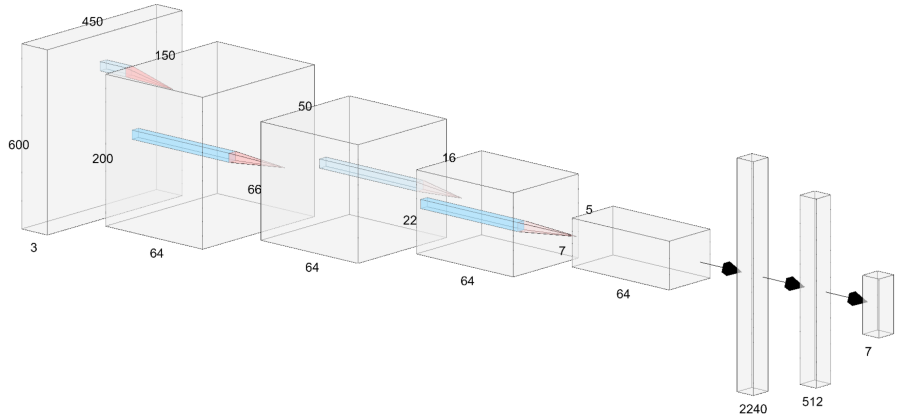

Figure 7: Convolutional neural network architecture visualised.

Model	**Accuracy**	**Precision**	**Recall**	**F Score**
N_1	0.685	0.273	0.915	0.306
N_2 ROS	0.571	0.549	0.949	0.543
N_2	0.754	0.427	0.952	0.497
N_3 ROS	0.664	0.546	0.953	0.508
N_3	0.798	0.603	0.946	0.608
naive classifier	0.660	0.030	0.996	0.049

Table 1: Validation metrics of different classifiers on HAM10000.

6.4 Transfer Learning using Google's Big Transfer model

Many image recognition tasks suffer from too small datasets available. A state of the art approach to solving this problem is **transfer learning**. The idea is to train a model to perform a similar task on significantly larger amounts of data and then **fine tune the model** which means training parts of the model again on the small dataset at hand. In the case of convolutional neural networks this is easier to comprehend when the filters are regarded as feature extractors. The first layers of the network extract low level features like lines and circles which have general relevance. Having trained the feature extractors

on a large amount of data means that they are well optimised and show good performance for extracting relevant information from the image. These features can be transferrable between different image recognition tasks.

The type of transfer learning employed here is using the pretrained Big Transfer (BiT) model BiT-M R50x1 which is a large convolutional neural network using advanced techniques laid out in [1]. As in the convolutional neural networks seen before, BiT-M R50x1 can be split into the convolutional part extracting features and the part that interprets these features. Here, the first part is kept as is and the second part is removed. On top of the flattened layer yielded by the last layer of the feature extracting part a fully connected feed-forward neural network can be trained to learn to interpret the features extracted by BiT-M R50x1 in the context of skin disease recognition. This represents the fine tuning process.

An **epoch** is one iteration of training a classifier on the entirety of the training data. For measuring the performance of the final classifier called tf **10-fold cross validation** will be used. K-fold cross validation first divides the data set into K equally large parts, then the first part is used as validation set after training on the rest of the data set. Iterating through all the K parts, each time taking a different one of the K parts for validation and the rest for training results in a sample of length K for each metric. This sample can be used to estimate a confidence interval or calculate a mean. Assuming a Gaussian distribution for the accuracy, the 95% confidence interval and mean values in Table 2 can be obtained.

Model	**Accuracy**	**Precision**	**Recall**	**F Score**
tf	0.862 ± 0.009	0.678	0.967	0.712

Table 2: 95% Confidence interval for accuracy and the mean for other metrics for the classifier tf.

To see whether or not certain classes are harder to predict than others, a **confusion matrix** as seen in Figure 8 can be consulted. Each element in the matrix contains the percentage of images of the true class represented by the row which is predicted by the classifier to be in the class indicated by the column. It is worth noting that many images of the class DF are misclassified as BCC and many images are wrongly classified as NV, which is in line with considerations about the skewed distribution. Despite VASC being the class with the second least amount of images, many VASC cases are spotted by the classifier. This can be explained by their often but not always distinct red colour, a feature that

can set it apart from other classes.

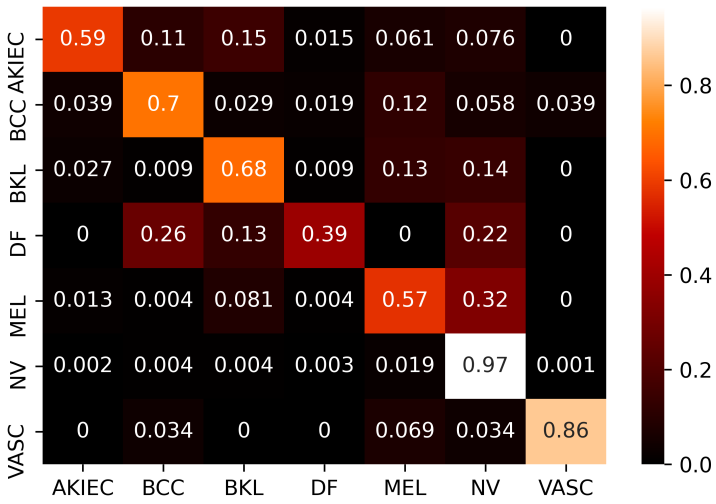

Figure 8: Confusion matrix of the transfer learning approach on HAM10000.

This model will also be tested on the ability of generalisation to another sample of skin images not included in the HAM10000 dataset. 3000 Images from the BCN20000 (see [2]) are picked from the ISIC 2019 training dataset and will be referred to as **BCN3000**. Having trained the model tf on the training set of HAM10000, metrics will be calculated using BCN3000 and can be seen in Table 3. Metrics are significantly worse compared to the HAM10000 evaluation set, which means this model is unlikely to generalise to BCN3000. For further analysis, distributions of the training set, prediction classes and true classes of the predicted set can be investigated. Figure 9 shows that the distribution of the predictions resembles the distribution of the training set.

Model	**Accuracy**	**Precision**	**Recall**	**F Score**
tf	0.505	0.884	0.344	0.348

Table 3: Metrics for the BCN3000 dataset.

For a comparison that is purely based on image features, photos from BCN3000 will be chosen such that the ratio of images contained in NV and other classes is the same as in HAM10000. This is possible for each class except for AKIEC since there are 0 AKIEC images present in BCN3000. The procedure results in 1724 images and the resulting metrics can be seen in Table

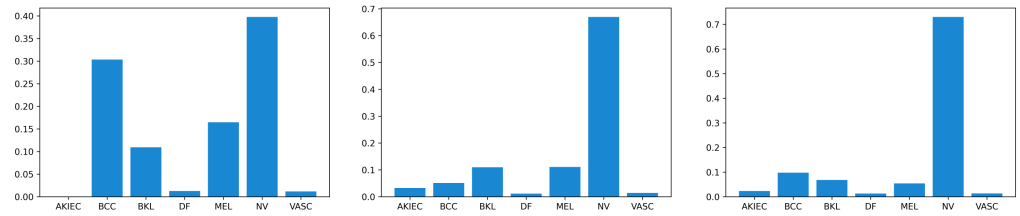

Figure 9: Distributions. left: ground truth of the BCN3000, middle: ground truth of the data trained on, right: predicted classes

4. The accuracy is drastically improved but is still just marginally above the accuracy of the naive classifier. Meanwhile precision and recall only differ by some noise because precision and recall for each class estimate a percentage that is independent of the size of the class, but the more images are in that class, the better the estimation. Finally an empirical comparison between a fully

Model	**Accuracy**	**Precision**	**Recall**	**F Score**
tf	0.707	0.884	0.354	0.386

Table 4: Metrics for the BCN3000 dataset with correction of the distribution.

connected neural network and the perceptron will be made. Using the features extracted by the feature extractor used in the transfer learning approach, a binary problem will be considered. The raw image data was not considered because of lacking computational resources for the perceptron algorithm. The classifier has to differentiate between malignant and non-malignant images. The neural network is the same as the one on top of the feature extractor in the transfer learning approach. The naive classifier here selects the non-malignant class for every observation and the data is HAM10000 with a 80% training 20% validation split. Table 5 shows that the perceptron is able to find patterns in the

Model	**Accuracy**	**Precision**	**Recall**	**F Score**
Naive Classifier	0.837	0.5	0.419	0.456
Perceptron	0.860	0.575	0.884	0.593
Neural Network	0.896	0.762	0.826	0.788

Table 5: Metrics for different models on the binary problem.

extracted features but the neural network is superior in accuracy and F score. The perceptron shows a slightly better recall so dependent on the situation the perceptron could be preferrable but it is possible that there is a neural network architecture that performs better in all of the metrics.

6.5 Conclusion

The perceptron has demonstrated the ability to detect patterns in data extracted from the HAM10000 image data set. However, common implementations did not allow for the use on raw image data as this data would have had 810.000 dimensions. Convolutional layers are able to extract crucial information from large images in a condensed form which enables more basic classifiers to interpret these extracted features. Basic neural networks showed to be ineffective on raw image data but more effective than the perceptron in interpreting information extracted by a convolutional neural network. Different smaller architectures of convolutional neural networks already turned out to perform better than a naive approach where each image would be classified as the class containing the most images. In these experiments efforts to balance the number of elements for each class before training resulted in worse performance of the classifiers. This can partly be attributed to the fact that the classifier learns to reason in a Bayesian manner. This means if an image looks like class A and like the much more frequently encountered class B with similar certainty, it is logical for the classifier to opt for the class B because this way it correctly classifies the image more frequently. Finally, transfer learning was applied to the problem. A large convolutional neural network, that was fine tuned on massive amounts of image data was used to extract features from the HAM10000 dataset. A neural network interpreted these features and scored exceptionally well. When tested on another dataset, the BCN3000, the metrics worsened significantly and remained low after re-weighting the classes of BCN3000 to match the class distribution of HAM10000. To summarize, the transfer learning approach was the most successful, followed by custom built convolutional neural networks and finally a fully connected neural network and the perceptron that showed limited utility.

References

[1] Alexander Kolesnikov et al. *Big Transfer (BiT): General Visual Representation Learning*. 2019. eprint: arXiv:1912.11370.

[2] Marc Combali et al. *BCN20000: Dermoscopic Lesions in the Wild*. 2019. eprint: arXiv:1908.02288.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[4] Justin M. Johnson and Taghi M. Khoshgoftaar. "Survey on deep learning with class imbalance". In: *Journal of Big Data* 6.1 (Mar. 2019), p. 27. URL: https://doi.org/10.1186/s40537-019-0192-5.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[6] Chigozie Nwankpa et al. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. Nov. 2018.

[7] A. Rosebrock. *Deep Learning for Computer Vision with Python: Starter Bundle*. PyImageSearch, 2017. URL: https://books.google.at/books?id=9Ul-tgEACAAJ.

[8] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[9] Philipp Tschandl. *The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions*. Version DRAFT VERSION. 2018. URL: https://doi.org/10.7910/DVN/DBW86T.

[10] Kilian Q. Weinberger. *Lecture 3: The Perceptron*. URL: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote03.html.

[11] Wikipedia contributors. *Hyperplane separation theorem — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-August-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Hyperplane_separation_theorem&oldid=966174876.