MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

HEART DISEASE CLASSIFICATION USING GENETIC GENERATED NEURAL NETWORK LAYOUT

LICENSE THESIS

Graduate: Corina Cătărău-Cotuţiu
Supervisor: Assist.Prof.Dr.Eng. Mihai Negru

2017

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Corina Cătărău-Cotuţiu**

HEART DISEASE CLASSIFICATION USING GENETIC GENERATED NEURAL NETWORK LAYOUT

1. **Project proposal:** *Designing a neural network capable of classifying heart diseases by investigating the effect of tuning parameters on the accuracy of the system and selecting the corresponding layout using a genetic algorithm.*
2. **Project contents:** *Introduction, Project Objectives and Specifications, Bibliographic research, Analysis and Theoretical Foundation, Detailed Design and Implementation, Testing and Validation, User's manual, Conclusions, Bibliography*
3. **Place of documentation:** *Technical University of Cluj-Napoca, Computer Science Department*
4. **Consultants:**
5. **Date of issue of the proposal:** November 1, 2016
6. **Date of delivery:** July 14, 2017

Graduate: _____

Supervisor: _____

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnata
CĂTĂRĂU-COTUŢIU CORINA, legitimată cu CI seria KX nr. 876800
CNP 2940404125489, autorul lucrării CLASIFICAREA BOLILOR CARDIACE FOLOSIND RETELE NEURONALE GENERATE GENETIC (HEART DISEASE CLASSIFICATION USING GENETIC GENERATED NEURAL NETWORK LAYOUT) elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea CALCULATOARE ENGLEZĂ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea IULIE 2017 a anului universitar 2016-2017, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data

Nume, Prenume

_____

_____

Semnătura

Contents

Chapter 1

Introduction - Project Context

1.1 Background

Nowadays we live in an era that is slowly becoming more and more technology driven. We can see smart households, smart automobiles and a lot of predictive applications, all created in order to improve the general quality of life. I will now go over some milestones that made a large contribution to what we have today.

Alan Turing in 1950 developed a test, that was designed to deliberate whether a machine could be considered smart. How this works is that there are human judges at one side and at the other side there are humans and one chatbot, the chatbot needs to convince the judges that he is a human, so far some achievements of robots were to persuade 30% of the judges. But this is still one of the most accurate "robot-smartness" tests.

In 1952 the first machine learning program was written by Arthur Samuel, he developed a program that improved at playing the game of checkers.

Another fundamental milestone was attained in 1957 by Frank Rosenblatt who designed the first neural network for computers, called the perceptron. This study is now at the cornerstone of a lot of machine learning programs that are designed today.

1.2 Project context

For nearly 70 years the field of artificial intelligence has been studied and today we have reached a spot where the computational power of our computers allows us to rapidly develop and train algorithms. This smart epoch, yearns to aid the public not by giving them information but by interpreting it and finding connections that are not visible to the human eye.

A significant landmark in AI, that triggered the interest in the area for numerous people, was the victory of AlphaGo against a human agent. The program, engineered using deep reinforcement learning, has learned by itself how to play the game at a proficient level. Such a breakthrough can only make you wonder what other important associations can a

computer render that you can not. In US clinics artificial intelligence has already began to aid doctors. They analyse data and provide possible medical outcomes, but so far these only facilitate the work of doctors. In a domain where precision is the key there is no room for error.

Having seen how computer can help predict sicknesses, was the inspiration behind the topic of my thesis. Using machine learning algorithms and medical data I will try to predict as accurately as possible the predisposition of a person to heart-attacks.

The reason why I choose heart-attack prediction is the fact that, according to the World Health Organisation, cardiovascular diseases remain the first cause of death globally. In 2015 about 17.7 million people died from cardiovascular problems, this is the equivalent of 31% of all deaths 1.1. Heart attack or strokes were behind 6.7 million of them, which I consider to be a problem, and measures need to be taken in order to diminish these worrying numbers. Most of these deaths occur because they are not early detected and when they are, they are at a point where they can not be fixed anymore.

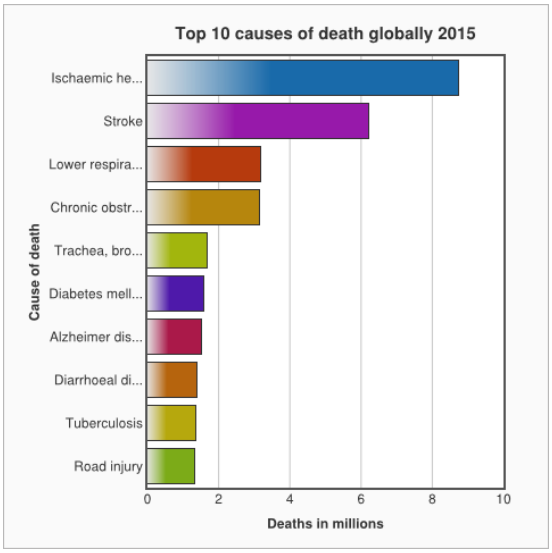

Figure 1.1: Top 10 causes of death globally 2016
[34]

Chapter 2

Project Objectives and Specifications

My thesis has as theme the development of a heart-attack classification algorithm by using neural networks and other algorithms in order to predict accurately the possibility of such an event. In order to have a good understanding of the algorithm and to provide a good solution my objectives are as follows:

1. Understanding neural networks and their mathematics
2. Developing a neural network capable of classifying heart disease predisposition
3. Training the algorithm by using different types of functions activation/loss/optimisation
4. Training the algorithm using a genetic approach
5. Testing and validating the neural network results
6. Comparison of the obtained results with the state of the art results

2.1 Understanding neural networks and their mathematics

Neural networks are a computational model that has as backbone a lot of mathematical parameters. In order to obtain good results, researches have shown that an understanding of the mathematical background of this algorithm is more than necessary.

At this point it is important to pay attention to the details of each parameter, for example how different activation functions can lead to substantially different results. Attention will also be given to the problems that might occur and how this might be fixed.

My objective is to understand what each of the parameters mean and how much they influence the performance of the algorithm.

2.2 Developing a neural network

After having had understood what is behind the black-box of neural networks, my own neural network will be developed capable of classifying the heart disease predisposition of a person based on some attributes provided by the UCI Repository.

In order to achieve this, the dataset will need to be analysed and preprocessed, followed by multiple training processes that have as goal finding the best solution for this problem.

At this point time will also be allocated for research of related solutions, that could stand as inspiration for the implementation of my network. Having this research made could help me prevent making mistakes that have already been analysed by other people. I will use the results to see whether the results I have obtained are good or if they still need improvement.

2.3 Training using different parameters

In the field of neural networks the task of finding the best combination of parameters is most of the times a trial and error process. Due to this fact neural networks will be generate programatically, each time with different parameters to see how this affects the accuracy of the system.

The neural network functions will be first evaluated, the activation, the loss and the optimisation ones. The types of functions will be selected from the documentation of the framework that will be used.

Even though the training with so many parameters is quite a time consuming task, the results tend to be worth the time.

2.4 Training the algorithm using a genetic approach

Having seen what a time consuming task is the training of the algorithm using changing parameters, for the selection of the layout a genetic approach will be used. A genetic approach means using the power of genetic programming in order to optimise the search for the best solution.

The genetic approach means that we select a population of neural networks with different layouts and the algorithm doesn't generate all possible solutions, but it selects the best from each generation and continues from there.

In term of numbers, assuming that the training and evaluation our neural network will take us about 10 minutes, and the parameters that we have are three activation functions, the number of hidden layers and number of neurones for each hidden layer, each with only 4 possible values, even though in our case we have more than 4 possible values,

this will mean

$$(4^5) * 10minutes = 10,240minutes = 170hours = 7days$$

A week is a bit too much for our task, but considering the genetic approach, the time reduces to more than 50%.

2.5 Testing and validating the results

For neural networks validating the results is an essential component. Many problems can occur during training, an example is overfitting, when the algorithm trains on the same part of the dataset, where it obtains good accuracy, but when presented with new data it will perform poorly.

This happens because the algorithm starts learning too much about a particular part of the dataset. In order to solve this problem I will train the dataset using the k-fold method which I will describe later, and then validate the results against a smaller part of the dataset, that was yet to have been seen by the algorithm.

In order to have a better observation of the results that I have obtained I will use different metrics such as: accuracy, true positive, true negative, loss and so on. The results of both the evaluation of the functions and of the layouts will be stored in files, in order to keep track of what I have obtained after each step and whether or not a change in the algorithm improved it or not.

2.6 Comparison of the obtained results with the state of the art results

After deciding on the final solution based on the metrics stated before, I will compare my results with state of the art results in the field of neural networks, as well as with the results obtained by other researchers on the same problem.

This comparison will also be made after each improvement step, in order to observe whether or not I am getting closer to a good solution.

Chapter 3

Bibliographic research

3.1 Machine learning algorithms

In the field of machine learning there are a great variety of algorithms designed to solve problems of classification and prediction.While all of them work very well, they can also perform poorly if the problem you choose to solve is not meant to be solved by these kind of algorithms. A way in which machine learning algorithms are usually categorised is Supervised Learning and Unsupervised Learning.

Supervised learning refers to the type of algorithms that learn to predict in what type of a known class an element belongs to, mathematically speaking supervised learning can be viewed as a function:

$$f(x) = y \tag{3.1}$$

where we know the inputs X, we know the output Y and we need to find out the mapping function. Among the supervised learning algorithms we can enumerate the following:

1. Logistic Regression
2. Back-propagation Neural networks
3. Support vector machine

On the other side unsupervised learning receives data without knowing the labels in which it can be categorised, so the structure needs to be learned as well.In the same function 3.1, we know this time X, but Y is also an unknown, so the algorithm needs to find out the mapping function and the output value types.

Some types of unsupervised learning algorithms are the following:

1. K-means

2. Apriori

Choosing the right type of algorithm implies analysing the data that you have, in order to see in which group of algorithms the problem falls. In our case, we want to solve a prediction problem, so our data contains both input and output information, hence we need only to find the mapping function. As mentioned in the above classification, when wanting to find the mapping function a supervised learning algorithm does the best job.

3.2 Neural networks

Neural networks is a computational model that is based on the architecture of the biological nervous system.A neural network is formed of several entities that are interconnected between each other in the same way in which neurones are connected through synapses in the brain.The electric charge that travels through the synapses is transformed into weights and biases.

3.2.1 Neurones

Before going starting with how does a neural network look, what are optimisations that can be done, let's first see the core elements that form a neural network. We know that the core elements of the human brain are neurones, their number is 10^{11} and they are in more than 20 types. This small but important parts of the brain are in fact also the core of neural networks.

So how can a neurone from the human brain be modelled in order to work as a computational unit. Below we can see two figures of neurones, one that represents the biological neurone 3.1 and one which represents the neurone as a computational unit 3.2.

The neurones both the human brain neurone and the computational one receive inputs through their dendrites and they produce an output which is then passed on to the other neurones that are connected to this one, so the output of this neurone becomes the input of the next ones.

In the computational model, we can see that the input don't only pass through the neurone, they are multiplied with the weight that are assigned to the dendrites.

Weights are values that if applied to an input might change the result of the output. For example if we have a decision making problem, where we have inputs X and Y and output Z, and we know that X should influence more Z ,than X should receives a higher weight. In a neural network, the weights are learnable and they adjust in order to get closer to the desired result. Then the multiplication from each dendrite is sent to the body of the neurone, where they are summed. The decision of whether a neurone is "activated" or not is made by passing the result to an activation function. The most basic example is a step function, where if the result is grater than a certain threshold then the neurone is activated.

To the sum of products, the **bias** is also added. What the bias does is shifting the activation function to the left or to the right, doing this in most cases helps finding the solution faster. Most of the time just changing the weights won't lead to the solution due to the fact that weights only change the steepness of functions, while the bias shifts it to the right or left.

$$Y = \sum_{1}^{n} (W_i * X_i) + B$$

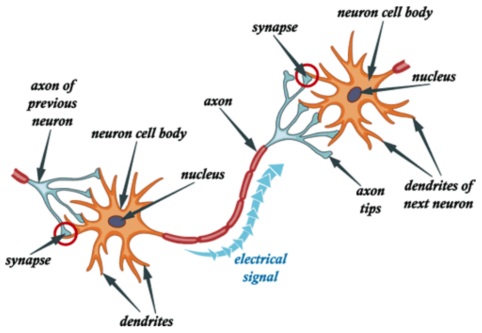

Figure 3.1: Neurone [44]

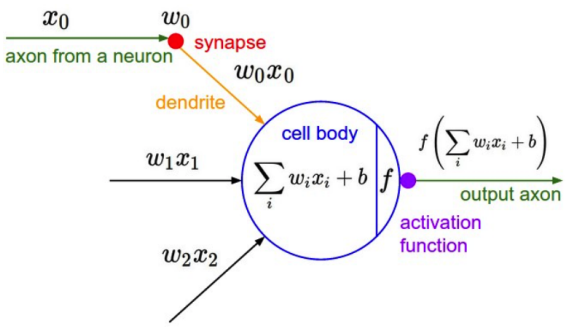

Figure 3.2: Neurone Neural network [28]

3.2.2 Activation functions

As mentioned before, activation functions play a major role in the way the neural network works. There are several types of activation functions that should be considered when designing a neural network.
The characteristics that are usually desired in an activation function are nonlinearity and continuously differentiable. The reason for these two properties is the fact that having a nonlinear function enables a network to learn nonlinear behaviour, while being continuously differentiable is important because most of the networks are trained using a gradient descent method. This is what happens at the neurone level:

1. First the data arrives at the input of the neurone
2. Each layer sends the data to the next layer, this means that each layer processes data from the nth-1 layer.
3. The input is multiplied with a weight parameter, and the resulting products are then added. To this result the bias is also added.
4. After this computations the result of the sum is passed through an activation function, which will decide whether or not to activate the neurone.

1. Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

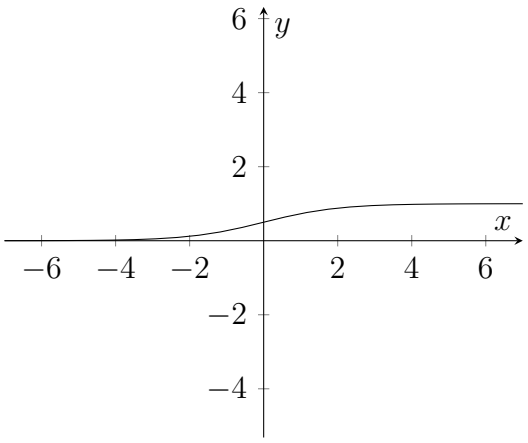

Figure 3.3: Sigmoid function

The function is also known as logistic or soft step. As we can see in the figure 3.3 of the function sigmoid,

$$f : \mathbb{R} \longrightarrow [0, 1]$$

is smooth and differentiable. Sigmoid function gives to large positive numbers the value 1 and to those small negatives ones value 0. This function was widely used as an activation function due to the fact that whether or not a neurone is active is very straightforward. An advantage of this function is that for X between [-2,2] , the values of Y are very steep, which means that a small change in X produces a major change in the output Y. Even though it has advantages, its drawbacks are the ones that made this function less and less popular.

(a) Sigmoid saturates and kills gradients

What does this mean, is that as we can see on the graphical representation of the function ,towards each of the ends of the function, the output(Y) becomes unresponsive to the changes in the inputs. This leads to a small gradient, or vanished gradient, also if the weight are not properly initialised , for example they are too big then the neurones become saturated which will prevent the network from learning.

(b) Sigmoid outputs are not zero centred

This might affect the network in the later processing steps, but this is not as severe as the first drawback.

2. Exponential Linear Unit (ELU)

$$f(a,x) = \begin{cases} a(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases}$$

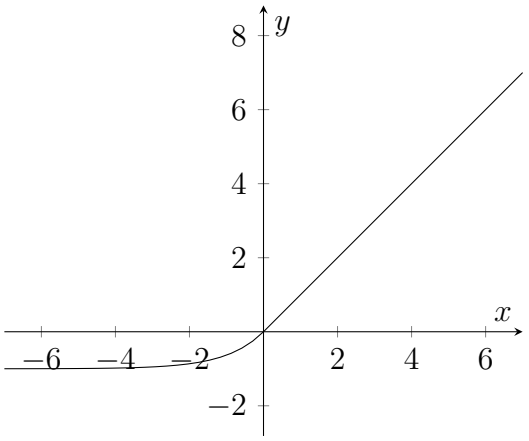

Figure 3.4: ELU

The values of elu go from (-a, inf). The problem that occurs to the sigmoid function, the one of the vanishing gradient disappears when using elu function. The 'a' parameter that appears in the function controls the value to which elu saturates for negative values. In comparison with RELU 3.5 we can see that ELU has negative values which push the mean of the activations closer to zero, such a feature enables faster learning.

3. Rectified linear unit (RELU)

$$f(x) = max(0,x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

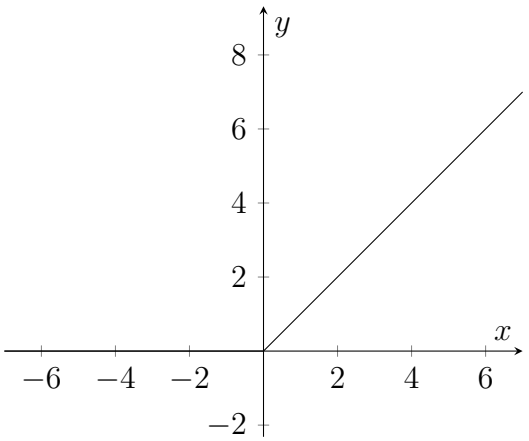

Figure 3.5: RELU

The relu, as we can see it's actually just a threshold at 0. The range of relu values are between [0,inf). As all the other activation functions, the RELU has also benefits and disadvantages. One of its main advantages is that it has a really small computational cost, its implementation being only a max, where one of the variables 0. It is also interesting to emphasise the fact that having the value 0 on the negative part, leads

to both advantages and disadvantages. A randomly initialised network where half of the activations could possibly be 0, will lead to a very lightweight network, which is usually a good feature. But also because of the same part of the graphic, the gradient can also go towards 0, this causing these neurones to stop learning and leading to a network where half of it it is passive. This is called the **Dying Relu problem**.

4. Softplus

$$f(x) = ln(1 + e^x)$$

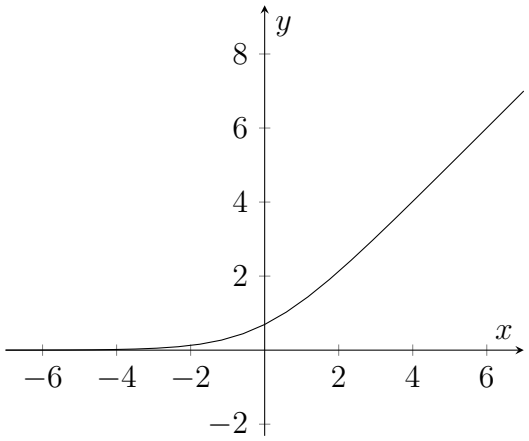

Figure 3.6: Softplus

Softplus goes from (0,inf). As we can in figure 3.5 and in figure 3.6, relu and softplus are not very different. The main difference between the two is at zero where Softplus is more smooth and differentiable, thus making Softplus differentiable everywhere as opposed to relu. From the computational point of view it is much faster to compute relu,max rather than log.

5. Softsign

$$f(x) = \frac{x}{1 + |x|}$$

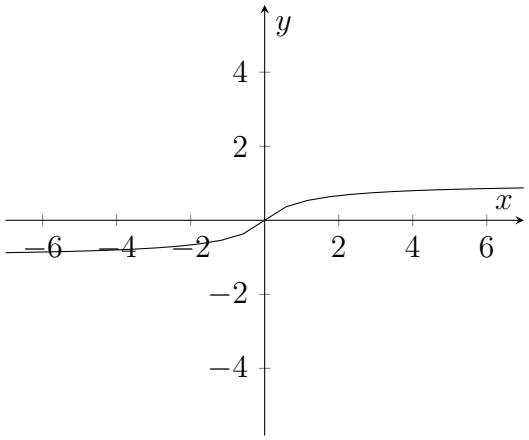

Figure 3.7: Softsign

As other functions this is also a nonlinearity. Similarly to tanh its values are as follows

$$f : \mathbb{R} \longrightarrow [-1, 1]$$

Also due to the fact that it is 0 at the origin, and its derivative is 1 at the origin we can see that this function is an approximation of the identity function at the origin. Computationally speaking this is less expensive than tanh, so it should be sometimes considered as a good replacement for tanh.

6. Tanh

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

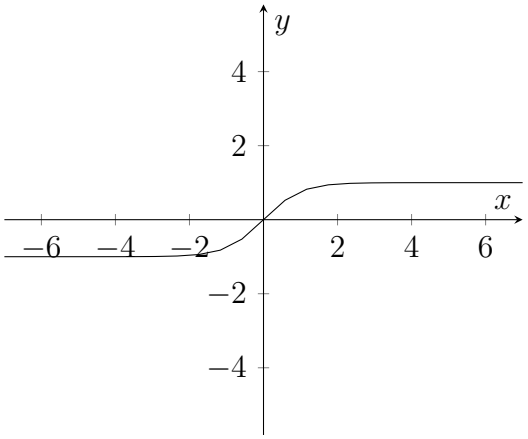

Figure 3.8: Tanh

Tanh is similar to the sigmoid function.

$$f : \mathbb{R} \longrightarrow [-1, 1]$$

In fact it is often considered that the tanh is a scaled sigmoid neurone. From the point of view of the desirable properties this is both smooth and differentiable everywhere. Another important feature is that it approximates the identity function near the origin. Considering the many similarities that exist between this function and the sigmoid function, it is clear that is can suffer from the same problem, which is the vanishing gradient problem.

7. Linear

$f(x) = ax + b$

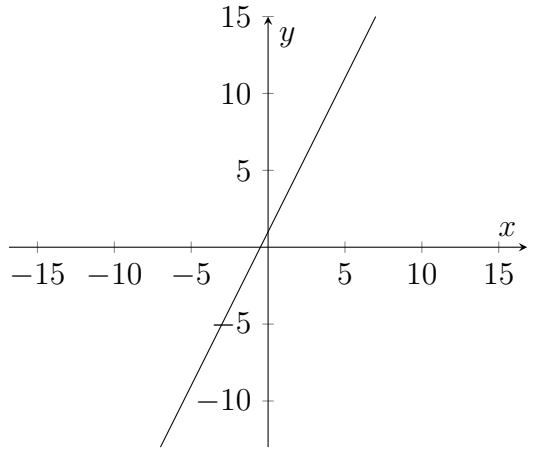

Figure 3.9: Linear

8. Softmax

$$f(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K}(e^{x_k})}$$

This function it is usually used for classification of the output values. The way in which this works is that the sum of the outputs is considered to be one , and therefore the values which form the one are the probabilities for each class.This is most commonly used at the output layer and almost never in the hidden layers.

3.2.3 Types of neural networks

Neural networks come in a great variety of shapes and models. Neural networks come as collections of neurones which are connected as an acyclic graph. The organisational mode of the neurones are layers, first there is the input layer which contains the input data, this is followed or not by a series of hidden layers (0,n) and the last one is the output layer, which can also contain one or more neurones depending on what we want to predict.

Based on the connectivity between layers, we have two types of neural networks **fully-connected** and **partially-connected**. The fully-connected type is the most common, in this type of architecture the neurones between 2 layers are fully pairwise connected.

From the point of view of how the data travels between layers we can enumerate the following:

1. Feed-forward NN

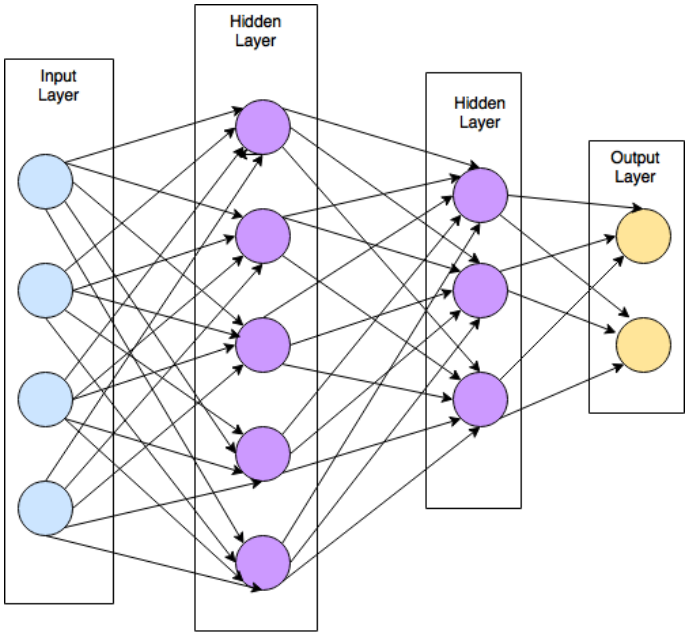

Figure 3.10: Feed-forward neural network

Feed-forward networks have the neurones arranged into layers, where the first layer takes the inputs and the last one produces the output. The middle layers coloured with pink in 3.10 have no connection with the outside world, hence they are called hidden layers. In the image 3.10 we can see that we have a fully-connected network due to the fact that all neurones from each layer are connected to all the others from the next layer. Having this type of connectivity we can say that the information is fed forward from one layer to the next, thus the name.

Forward propagation , is the technique in which the weights are randomly initialised and the output is calculated. Steps:

(a) weights randomly initialised

$$(w_1, w_n)$$

(b) sum product of inputs with weights

$$Y = \sum_{1}^{n} (W_i * X_i) + B$$

(c) apply one of the activation functions mentioned at 3.2.2

Using this approach rarely gives a good result, this is due to the fact that the weights are randomly initialised and the output is only a set of mathematical computations. In order to solve this problem back-propagation is introduced.

Back-propagation is the process of learning in a feed-forward network. The term of learning is usually used in the realm of supervised learning, in which the network has to learn the relationship between inputs and outputs 3.1. An informal explanation is that whenever the algorithm outputs an answer, it back-propagates and adjusts the weights in order to obtain a smaller error.

At the basis of the back-propagation algorithm is the partial derivative of the cost cost function with respect to the weights.

$$\frac{\partial C}{\partial w} \tag{3.2}$$

What the equation 3.2 tells us is how fast the cost changes when changing the weights.

Cost function measures how far from the desired output the obtained solution is.

The formula for cost function is 3.3 where k is the output layer where the error is calculated, d is the expected value while y is the calculated one. The 3.3 equation is for one neurone while the total error on the whole network is the sum of the errors on each neurone.

$$E_k = \frac{1}{2} * (d - y_k)^2 \tag{3.3}$$

Steps of back-propagation for the output layer:

(a) Calculate total error

$$TotalErr = \sum_{1}^{n} (\frac{1}{2} * (d - y_k)^2) \tag{3.4}$$

(b) Calculate the effect of a weight on the output error.
In order to see how much a change in a weight affects the error we use the partial derivative of the cost function 3.2 with respect to that weight.

$$\frac{\partial TotalErr}{\partial w_i} = \delta_{out_1} * out \tag{3.5}$$

(c) Decrease error
Our goal is to decrease the error, we obtain this by subtracting 3.5 multiplied by the learning rate from the current weight 3.6.

$$w_i = w_i - \eta * \frac{\partial TotalErr}{\partial w_i} \tag{3.6}$$

These steps can be repeated for each weight. For the hidden layers the calculations are similar with the addition that we need to take into account that each output of

a hidden neurone affects the output of the next layer neurones. For example having a hidden neurone h1 which affects two neurones from the next layer n1 and n2 the error becomes:

$$\frac{\partial TotalErr}{\partial out_h 1} = \frac{\partial Err_n 1}{\partial out_h 1} + \frac{\partial Err_n 2}{\partial out_h 1} \tag{3.7}$$

2. Recurrent NN

Recurrent neural networks have proven to obtain great results in several fields. What makes them different from the feedforward neural networks described at 1 is the feedback loop. The way in which recursive neural networks work is more similar to how us, human rationalise, meaning we don't start from scratch each time we need to take a decision, we use the information that we already have in order to obtain better results. A problem that occurs with RNN is that sometimes the gap between where the information was given and where it is needed can be quite large, case in which the RNN becomes incapable of connecting the two.

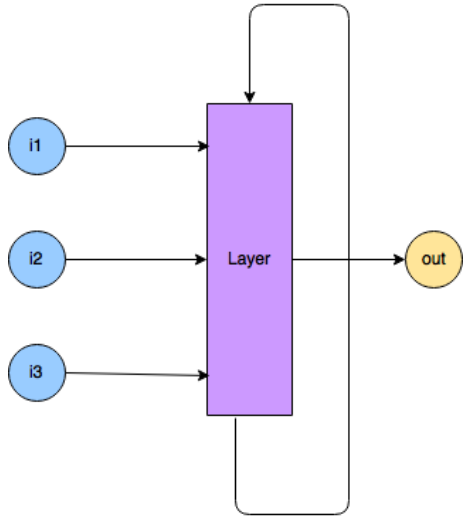

Figure 3.11: Recurrent neural network

A special case of recurrent neural networks is the **LSTM**, they appeared as a solution to the long-term dependencies problem.

(a) Long Short Term Memory (LSTM)

This type of networks are designed to learn long-term dependencies. At the core of a LSTM there is the cell state, where the neural network can add or remove information using gates. This cells react to the signals by allowing information to pass or by blocking it. What the LSTMs need to learn during time is when to perform one of the three operations: delete, keep, discard.

3.3 Genetic Algorithms

Genetic algorithms are a technique for searching optimisations. The inspiration behind genetic programming is found in the evolutionary theory. This type of algorithms are part of the branch of Evolutionary Computation and they are mainly based on the natural selection found in genetics.

Some of the most common terms that are used when talking about this kind of algorithms are:

1. Population - as the name suggests, it is similar to the human population, representing a subset of solutions

2. Chromosomes - are a subset of the population, meaning one individual from a set of solutions

3. Gene - subset of the chromosomes, an element belonging to an individual

4. Allele - the value of a gene or attribute for one individual

5. Fitness function - is a function that has as input a solution and gives as output how suitable that solution is

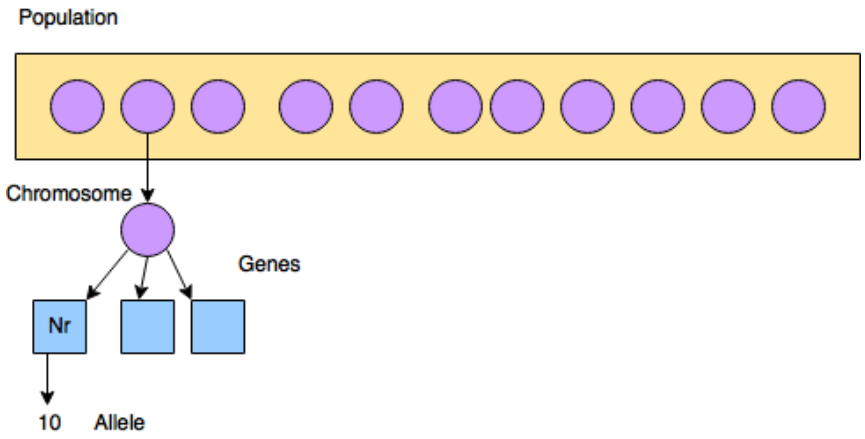

Figure 3.12: Population

In genetic algorithms we start with an initial population, or an initial set of solutions which are then recombined which each other and also mutations might occur which lead to the creation of new children. As in real life this process can spread on several generations. In order to evolve a population, we need to use operators:

1. Crossover

The crossover process between solutions is represented by the assignation of fitness value to each of the individuals. Fitter individuals, the solutions that have a higher fitness function, are more likely to go through the matting process. Choosing only the best solutions for matting, assures us that the next generations will be better and better. The reason why genetic algorithms perform better than a normal search is because these algorithms take into account the history, and what did or did not perform good.

2. Mutation

   This is in essential step in the evolution of the solutions. Changes are randomly made to chromosomes in order to allow new solutions to appear even after more generations. Without mutation a genetic algorithm would only a set of permutations of the same characteristics.

3. Selection

   As its name suggests, during this operation we select the individuals that will go through the process of crossover. There are several types of selection strategies:

   (a) Tournament

   The parents are selected as result of a "tournament " between K individuals chosen randomly. This process is repeated until all the parents are selected. This is selective strategy allowing only fit individuals to participate to crossover.

   (b) Roulette

   Roulette selection as opposed to Tournament does not select only the best individuals, the weaker ones also have a change to participate to the crossover process but with a smaller probability. This is due to the fact that all individuals receive a probability to participate to the crossover proportional to their fitness value. By using a fixed value we select the individuals that qualify.

   (c) Stochastic Universal Sampling

   It is similar to Roulette but comes with the addition of several points, allowing in this way to select all parents at once.

   (d) Rank

   This is a method that comes in handy when the fitness values of individuals are close to each other. This would mean that using Roulette most of the individuals would have equal probabilities. Rank comes with the solution of ranking the individuals and then choosing them based on their rank.

   (e) Random

   As the name suggests, this means that we select the individuals randomly.

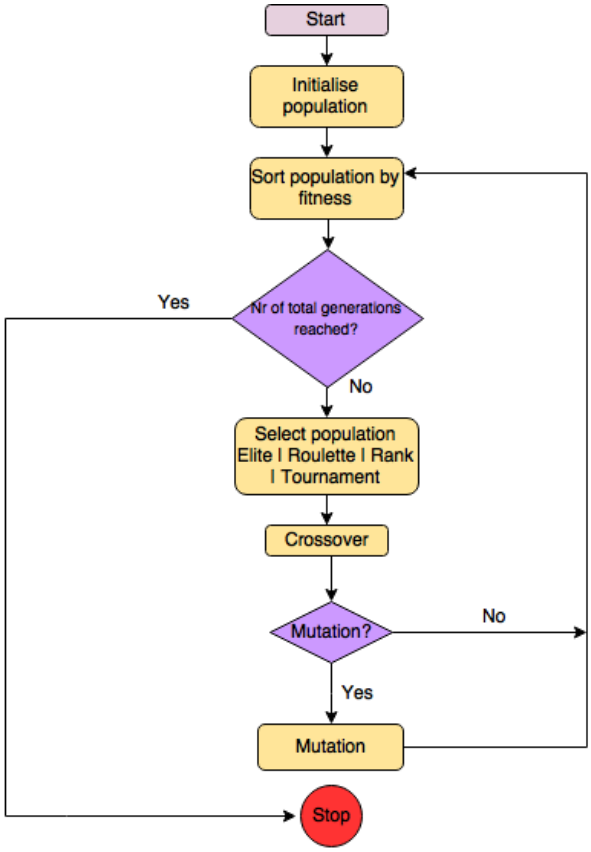

Figure 3.13: Genetic Algorithm

3.4 Previous work

In the field of machine learning algorithms several studies have been made, either by implementing a neural network with different tuning parameters or by trying different hybrid algorithms.On the topic of coronary heart disease prediction a lot of approaches have been studied, due to the fact that it is a sensitive domain where even a 1% error rate could mean the life of people. Some of the approaches worth mentioning, that have obtained impressive results are the following:

1. **Classification of Heart Disease using Artificial Neural Network and Feature Subset Selection By M. Akhil Jabbar, B.L Deekshatulu ,Priti Chandra** [17]

   Their approach of combining Neural networks and feature selection has proven to be effective and obtain accuracies grater than 80% on different datasets such as weather, heart disease, breast cancer, hypothyroid and many others.

Apart from the implementation of a 3 layer neural network without loops, in this approach an important part seems to be the feature selection stage.

The technique used in this paper is the PCA (Principle Component Analysis) which is basically used to reduce the dimensionality of data. By reducing the dimensionality of data, they succeed in removing irrelevant and redundant data and hence obtaining on the Statlog Heart Disease Dataset an accuracy of 92.8%.

2. **Cardiovascular Disease Prediction System using Genetic Algorithm and Neural Network by Bhuvaneswari Amma N.G.** [18]

   This method takes advantage of the power of learning that neural networks have and combines it with the optimisation capabilities of genetic algorithms.This approach has several steps:

   (a) Data preprocessing

   (b) Weight Optimisation

   (c) Training engine

   (d) Weight base

   (e) Prediction engine

   Genetic algorithms are used for weight initialisation, sets of weights are evaluated and fitness values are assigned to them in order to select the best combination. After selecting the next step is training the algorithm using the selected weights. By approaching the problem using this algorithm they have obtained a slightly better result that the one mentioned at point 1, that is an accuracy of 94.17%.

3. **Heart Disease Diagnosis using Extreme Learning Based Neural Networks by Muhammad Fathurachman,Noviyanti Safitri,Umi Kalsum,Chandra Prasetyo Utomo** [19]

   Due to the drawbacks that neural networks tend to have, such as long trainning time ,a lot of parameters that need to be tuned in order to obtain good and satisfying results, this approach uses another technique called Extreme Learning Machine(ELM).

   ELM is in fact a feedforward neural network, but with a single hidden layer. The authors state that this simple feedforward algorithm is much faster than a backpropagation neural network due to the fact that it has to perform only 3 steps:

   (a) initialise weights randomly

   (b) calculate the hidden layer output matrix

   (c) calculate the weight value of beta

Comparing ELM with SVM they have obtained slightly better results, but compared with Back-propagation the results haven't shown improvement.In conclusion the ELM does not bring much of an improvement as compared to back-propagation neural networks, but it brings an improvement to the training time.

4. **Intelligent Heart Disease Prediction System Using Probabilistic Neural Network by Indira S. Fal Dessai**[20]

   This approach tries to overcome some of the drawbacks of back-propagation neural networks by using Probabilistic neural networks which are a class of radial basis function networks.This type of neural networks are usually used in pattern recognition. In comparison with classical neural network architectures, Probabilistic Neural networks ,contain an extra layer called the pattern layer.

   Compared with classical back-propagation algorithms and with decision tree algorithm this approach showed a specificity improvement from 88% using back-propagation to 91.22%.

Chapter 4

Analysis and Theoretical Foundation

4.1 General overview

The system that I choose to develop, is a heart-attack predictor using neural networks. The purpose is to gain a deeper understanding of the algorithm, choose and evaluate different solutions and improve it by using different techniques such as genetic programming and feature selection. A general view of the main operations that I have implemented are presented in the diagram 4.1. During this chapter I will present an overview of each of the modules: Data preprocessing, Choosing neural network setup, Weight initialisation, Neural network validation.

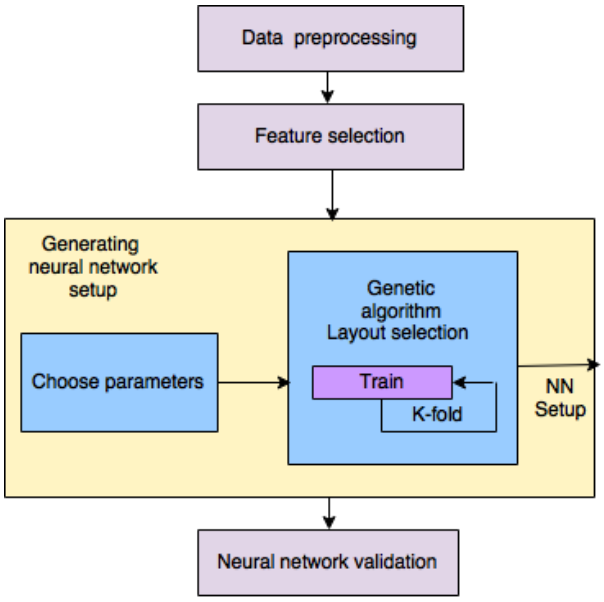

Figure 4.1: Algorithm modules

4.2 Dataset

The task of choosing the right algorithm, involves a careful analysis of the input data, we need to see what data we have and what we need to find out in order to choose the best algorithm.

The dataset that I choose to use is from the UCI Machine Learning Repository and it is called Heart Disease Data Set, this dataset has been widely used in related works on cardiovascular issues and machine learning solutions. The repository contains 4 files containing heart disease data from Cleveland,Hungary, Switzerland, and the VA Long Beach.All contain the same attributes but besides the Cleveland dataset, the others have a lot of missing data, which can cause the algorithm to perform poorly. Given this observation it is clear that I have chosen the Cleveland dataset.

According to medical studies there are 13 attributes needed in order to differentiate between the presence of heart disease and its absence.

The attributes used and what they mean are enumerated below:

1. age

   The values range between 29 - 77 with most of the values between 40-60.

2. gender

3. chest pain

   Types:

   - typical angina
   - atypical angina
   - non-anginal pain

4. trestbps - resting blood pressure (in mm Hg on admission to the hospital)

5. chol - serum cholesterol

6. fbs - fasting blood sugar

   Values: if the fasting blood sugar $>$ 120mg/dl then 1, otherwise 0

7. restecg - resting electrographic results Types:

   - 0 - normal
   - 1 - having ST-T wave abnormality
   - 2 - showing probable or definite left ventricular hypertrophy

8. thalach - maximum heart rate achieved

9. exang - exercise induced angina

   Values: if angina is present 1, otherwise 0

10. oldpeak - st depression induced by exercise relative to rest

11. slope - the slope of the peak exercise ST segment Types:

    - upsloping
    - flat
    - downsloping

12. ca - number of major vessels coloured by fluoroscopy

13. thalassemia(Thal) - it is an inherited blood disorder in which the body produces abnormal form of haemoglobin.

    Types:

    - normal
    - fixed defect
    - reversable defect

Output:
num - the predicted attribute, it can be a result between 0-4. Here, value 0 represents no presence of heart disease which means less than 50% diameter narrowing and values 1-4 represent the presence of heart disease that means greater the 50% diameter narrowing.

As the data shows this is a classification problem, where we know the inputs, the 13 attributes and we know the output, 4 classes that tell us the presence of heart disease. These observations brought me to the conclusion that this is a supervised learning problem 3.1. From the great variety of supervised learning algorithms , the ones that has proven to give the best results in the field of heart attack prediction were the neural networks 3.2.

4.3 Data preprocessing

What is data preprocessing,well as defined in several books such as Data preprocessing in Data mining, this step consists in operations of cleaning, removing of noisy or inconsistent data and also transformations of values in order to be into a format that is appropriate for the task. Data preprocessing is the first step that must be performed in all machine learning algorithms.In the case of Neural networks data preprocessing has proven to be an essential step that increases the performance of the algorithm. The task of data preprocessing addresses the following problems:

1. noisy data - weird symbols or unusual data

2. missing data - some values from the dataset might be missing, fact which leads to poor task performance
3. wrong data type - some data might need conversion to computable values
4. data normalisation

4.3.1 Data preprocessing steps

1. Cleaning the dataset

   This step focuses on having a consistent dataset, where all values are present and there are no bad values.In our case, the dataset was mainly clean, the only issues that I encountered were the missing values. When talking about missing values we usually mean having null values in some of the columns. In order to take care of missing data, there are several strategies:

   (a) Fill gaps forward or backward

   This method fills the missing values with values from its neighbours, the problem with this approach is if several values from the beginning of the dataset are empty, case in which the neighbours are no help, and the problem will persist.

   (b) Average filling

   Calculate an average of the values, and replace all missing values with that average.

   (c) Dropping null labels

   In some cases where the dataset is big enough, it can also be a solution to drop all the rows containing missing values

2. Transforming data

   Most datasets initially contain data in different formats, strings ,numbers, types. In order to be able to feed the data to an algorithm, the values should be numeric. In order to obtain numerical values from strings a careful analysis needs to be made, and the following questions answered:

   (a) Are this values some categories ?

   (b) How many labels do you have in a category?

   (c) Can they be encoded ?

   By answering these questions, you find out whether you need to perform a binary encoding or a label encoding.

   Binary encoding is applicable to categories that have only two possible values, for example female, male.

Label encoding is performed to those values, that have several possible values, by doing this, to every possibility you assign an integer.The problem with a simple label encoding is that there are some cases where simply encoding labels with integers leads to bad performance, this is because the algorithm understands that for example, labels 1,2,3 are actually prioritised meaning that 1 is better than 2 and so on. A solution for this problem can be performing a One hot encoding , this means that that one column transforms into several columns that can fit the binary representation of all labels. For example for three labels we would have 3 columns, when a individual would have label one as attribute the columns would look like this : 1,0,0.

In order too see whether you need a one hot encoding labelling it is enough to run your algorithm with and without this step and see which one performs better.

3. Feature scaling

   Computations can be poor if the data elements are from different scales, some important features might get lost because the scale is too small with respect to the others. In order to solve this problem we need to make sure that all attributes are on the same range, this leads to attributes having the same weights in the algorithm .

4.4 Feature selection

Datasets contain data that is not specifically tailored to the machine learning algorithms, some attributes present might not be relevant for the prediction task. This is usually solved by the weights that adjust during the training, but another method that can help us reduce the dataset and hence make the algorithm faster is called feature selection.

There are several algorithms that I evaluated for feature selection. This works by assigning a score to each attribute and choosing the ones that have the highest score.

1. Decision Tree Classification

   This method performs well as it considers all attributes and creates a split on the one that is separating the classes the best. The feature importance is calculated as the total reduction of the criteria brought by that feature.

2. Extra Tree Classification

   The difference from the Decision Tree Classification is when choosing the best split, we select random splits from each max-features.

3. Lasso Classification

   It penalises some coefficients by bringing them towards 0. The selection of variables is done using L1 penalisation (minimises the absolute differences between the target value and the estimated value). It's main drawback is when we have p feature and n samples, where p>n then the lasso select at most n variables.

4. ElasticNet Classification

   The ElasticNet Classification often outperforms Lasso, especially when the number of attributes is much greater than the number of observations.

5. LassoLars Classification

4.5 Neural network architecture

Choosing the architecture of a neural network can be quite a difficult task, many aspects need to be taken into consideration in order to obtain good results. In this section I will describe the architectural decision that I have made and the motivation behind them.

4.5.1 Parameters

Neural networks are fine tuned computing algorithms, that need to be calibrated based on each problem that they need to solve. The parameters might depend from a developing environment to another, some environments provide a more black-box type of approach where only some of the parameters can be changed while others provide a more open approach. The parameters that I will be considering while choosing the design of the neural network are the following:

1. **Activation function**

   As described at 3.2.2 activation functions can be of different types. For the hidden layers the process of choosing the best activation function was designed programatically, the program receiving a set of activation functions and then training the algorithm for each of them. For the output layer the selected activation function was the softmax 8, I choose this because it seems to be the best for multi-class classification problems. In our example the output is from 0-4 where 0 means the absence of a disease, while from 1-4 means increasing presence.

2. **Loss function**

   This function is used to measure the quality of a prediction as described at 1. If the value of this function is 0, then the result is 100% accurate.

   The loss function for which we trained the neural network are as follows:

   (a) mean squared error

   $$MSE = \frac{1}{n} * \sum_{1}^{n} (P_i - A_i)^2 \tag{4.1}$$

   Where P represents the vector of predicted values, while A represents the vector of actual values.

(b) mean absolute error

$$MAE = \frac{\sum_1^n |e_i|}{n} \tag{4.2}$$

(c) mean absolute percentage error

$$MAPE = \frac{100}{n} * \sum_1^n |\frac{A_t - E_t}{A_t}| \tag{4.3}$$

(d) mean squared logarithmic error

(e) squared hinge

(f) hinge

$$hinge(y) = max(0, 1 - a * p) \tag{4.4}$$

Where a is the actual values and p is the prediction.

(g) categorical cross-entropy

$$J = -t * log(y) + (1 - t) * log(1 - y) \tag{4.5}$$

Where t is the target value and y is the predicted value. For example if the prediction is correct yields in J=0.

(h) kullback leibler divergence

$$D_K L(p||q) = \sum_1^n p(x_i) * log\frac{p(xi)}{q(xi)} \tag{4.6}$$

(i) poisson

(j) cosine proximity

3. **Optimisation algorithms**

The purpose of this parameter, the optimisation algorithm, is to minimise the loss function about which we discussed at the previous point. This improvement is obtained by adjusting the values of the bias and of the weights.

There are two types of optimisation algorithms:

(a) First order optimisation algorithms

This type of algorithms minimise or maximise the loss function with respect to the internal parameters of the neural network. This category uses the first order derivative of the loss function, which holds information about points where the function increases or decreases.

(b) Second order optimisation algorithms

In comparison with the other category, these use the second-order derivative, or the Hessian to minimise or maximise the loss. The hessian matrix is a collection of second order partial derivatives. Due to the high computational cost of second order derivatives this is rarely used. The advantage of this method is that even though is expensive to compute, the minimum is found in less steps.

Gradient descent is one of the most techniques and it represents the basis of the training process. In 4.7 η is the learning rate, ∇ (θ).J(θ) is the gradient of the loss function J(θ).

$$\theta = \theta - \eta * \nabla(\theta).J(\theta) \tag{4.7}$$

Even though this was initially considered a good decision, in the last years variants of this approach have been more used.

The optimisation algorithms that will be evaluated are the following:

(a) SGD

Solves some of the problems that occur while using gradient descent. This performs a parameter update for each training example.

(b) Adagrad

It is still a gradient-based optimisation, but as an extra feature it adapts the learning rates to the parameters. This has shown good results when used in large neural networks. This approach using different learning rates for each parameter we must first compute the update per parameter. Using this approach eliminates the need to manually tune the learning rate, but it also can become very small which will stop the algorithm from learning.

(c) Adadelta

Is an improvement of Adagard, which diminished the problem of the small learning rates.

(d) RMSprop

It is in an unpublished learning rate method, that was proposed for the same reason as Adadelta, to solve the problem with the small learning rates in Adagard.

(e) Adam (Adaptive Moment Estimation)

As opposed to Adadelta and RMSProp this algorithm stores not only the exponentially decaying average of past squared gradients but also the exponentially decaying average of past gradients.

(f) Adamax

(g) Nadam (Nesterov-accelerated Adaptive Moment Estimation)

Is a combination between Adam and Nesterov Accelerated .

In order to better understand the differences in the image 4.2, we can see how they perform regarding a loss function. From the image we can observe how Adagard, Adadelta and RMSProp are from the beginning going towards the right direction, while the others are going off-track.

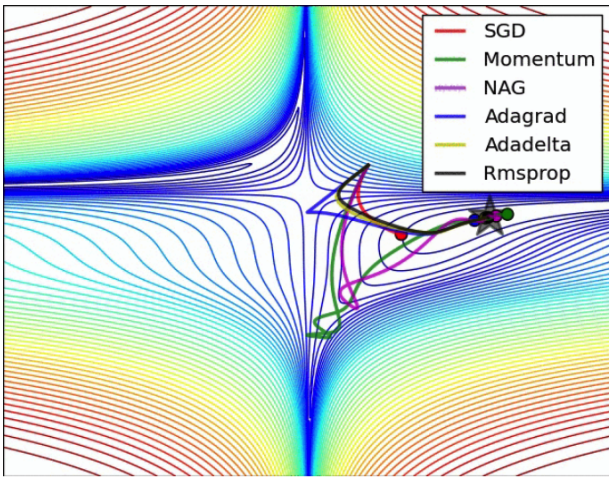

Figure 4.2: Optimisation algorithms behaviour [39]

4.5.2 Training

An important step when dealing with neural networks, is the training. To start this process the weights are initialised using different methods. In this field we can have two types of learning, supervised and unsupervided 3.1. During this stage the task of the network is to get better by minimising the error. The network is designed to learn connections among attributes that lead to a generalisation of the mapping function.

A network is considered to be fit if the result obtained using the approximation function matches the target values. In order for our algorithm to be fit many parameters need to be considered, the architecture established and also additional optimisation can be implemented.

Overfitting

Considering this information problems might occur while learning to generalise. When training, the algorithm might fall in the trap of learning unnecessary data only to obtain the target output, but when presented with new data it will have a poor ability to predict the correct result. This is a big problem called overfitting, that ruins the effectiveness of the machine learning algorithm.

The reason behind this phenomenon is firstly because we train the network using a batch of data and testing it with another one. The algorithm hence, does what is expected

of it, meaning it learns to maximise the accuracy on the given data. But as we know its fit attribute is given by how well it performs on new data.

The easiest solution for this problem is to have a large dataset, having a large dataset makes it harder for the algorithm to learn noisy data. In our case this is not a solution, because we do not have access to a lot of data 4.2.

Not having the luxury of a large dataset leads us to the solution of cross-validation. There are several approaches of cross-validation:

1. Holdout method

   The strategy simply implies splitting the data into two parts, training and testing. The problem with this approach is the fact that it can still lead to overfitting when the test data is very different from the training data.

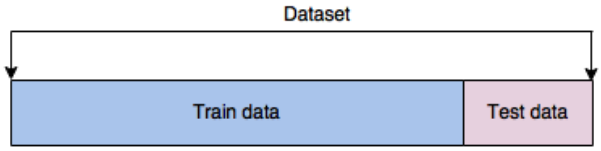

Figure 4.3: Holdout

2. Random subsampling

   We split the data into random number of samples, and evaluate for each experiment. The performance is measured as the average of the results from each experiment. The drawback of this method is based on the fact that the sampling is random, which can cause for some parts of the dataset to never be evaluated.

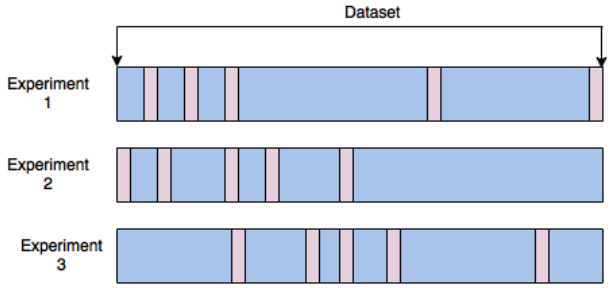

Figure 4.4: Random subsampling

The method which I chose for cross-validation is called **K-fold** and it overcomes both the drawbacks of holdout and random subsampling.

K-fold

This method splits the data into k folds, from which uses k-1 to train and one to test. This approach resembles Random subsampling but as opposed to it, k fold iteratively uses all the data from the dataset, not leaving unexplored data.

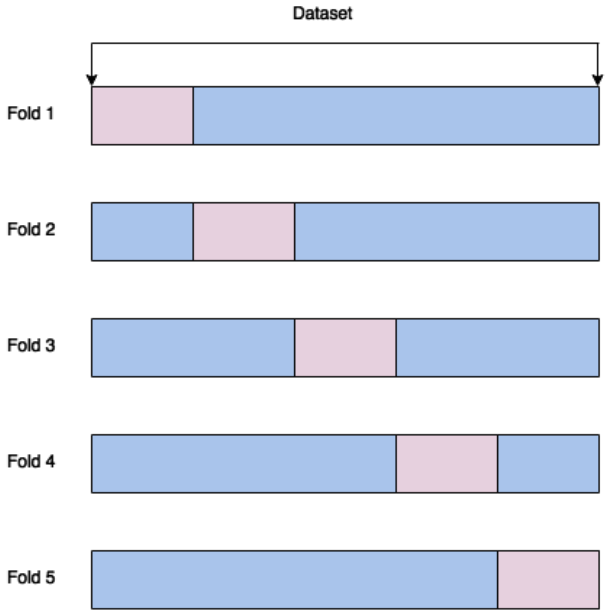

Figure 4.5: 5-Fold

In 4.5 I present exactly the method I used for my algorithm, which is a 5-fold. The process of splitting and evaluating is performed until all the subsets are used as validation set. The decision of choosing k=5 is due to the fact that the evaluation of deep-learning models takes quite a lot of time and a bigger k would mean a larger amount of time.

4.5.3 Layout

When choosing the correct layout for a neural network there are not very many formulas that give you the exact solution. There exist theories regarding the number of layers, or the number of neurones in each layer but most of the time the solution is found out by trying. Problems from the wrong size of a neural network occur in both the cases when we either have a large neural network, which memorises the data, hence leading to overfitting 4.5.2, or a small one that can only partially solve a problem.

Existing no analytical solution, can be considered to be quite a big problem, due to the fact that there are infinite times combinatorial possibilities. Some approaches suggest training the network using several layouts and choosing the one with smallest error, but this is also limited to the ”ideas” of layout that the programmer has.

Some approaches that have been used along time to calculate the number of neurones are the following:

1. Xu and Chen [14] proposed in their work an approach the determines the number of hidden neurones by using the root means squared error.

$$Nrofneuronesforhiddenlayer = C_f(\frac{N}{d * logN})^{\frac{1}{2}} \tag{4.8}$$

   In [4.8], N represents the number of training pairs, d is the input size and Cf is the first absolute moment of Fourier magnitude distribution.

2. An approach that doesn't involve trying different combinations is Hunters work [13]. Three networks are implemented MLP, fully connected cascaded network and MLP bridged. The number of neurones in a hidden layer is calculated using 3 different formulas:

   Bridged MLP

$$N_h = 2N + 1 \tag{4.9}$$

   Fully connected cascaded

$$N_h = 2^n - 1 \tag{4.10}$$

   MLP

$$N_h = N + 1 \tag{4.11}$$

3. Sequential Orthogonal Approach(SOA) [15], this is another approach used to determine the optimal number of neurones. This works by adding neurones one by one, this stops when the error is considered to be small enough.

Above I described only a few approaches that are able to compute the optimal number of neurones in a hidden layer, but the problem of choosing the number of hidden layers remains more of a cross-validation problem. Not having a solution for both the problem of choosing the number of neurones and layers is the reason why the approach that I choose isn't one of the above. Hence the approach that I choose to implement is evolving the neural network layout using genetic programming.

Genetic algorithm approach

Genetic algorithms as described at 3.3 provide an optimisation to the large search problems. An iterative approach for this case would only be possible with a subset of the possible number of layers and neurones, otherwise the search space is infinite.

Solution

1. Initialise random population of layouts

   This means we create random layout individuals, that have random number of hidden layers and random number of neurones on each hidden layer.

2. Assign fitness values

   Each layout gets a fitness assigned. In this case the fitness I considered, is the accuracy of the network. The accuracy is computed by using 5-fold cross validation technique 4.5.2.

3. For a number of 10 generations we repeat the following steps:

   (a) We apply a selection method on the individuals of the population. The two selection method that I use are:

      i. Tournament
      The way that I use tournament is by selecting three individuals from the population and determining which one is the best from those three.

      ii. Roulette
      When designing the roulette selection, each individual receives a probability to be selected proportional to its fitness function value. Then a fixed point is selected and according to that point the parents are selected. The probability for an individual to be chosen is proportional to its fitness, but this method as opposed to Tournament allows some probability to weaker individuals to be selected, hence avoiding the problem of local maxima.

   (b) We create offsprings of the best individuals.

   (c) We perform mutation on a random number of individuals, in order to sustain diversity.

   (d) We evaluate the new individuals.

   (e) We perform an optimisation step called Plague, which eliminates the individuals that are below average.

   (f) We replace the population with the new ones.

4.5.4 Final architecture

The evaluation of parameters lead to a couple of solutions performing similarly. The final decision was made also based on the theoretical background that I gathered from neural network literature.

The final parameters are as follows:

1. Activation function: sigmoid

2. Optimisation method: Adagrad, which has as benefit adaptive learning rates

3. Loss function: squared hinge

4. Weight initialisation: he normal, it draws samples from a truncated normal distribution centered on 0

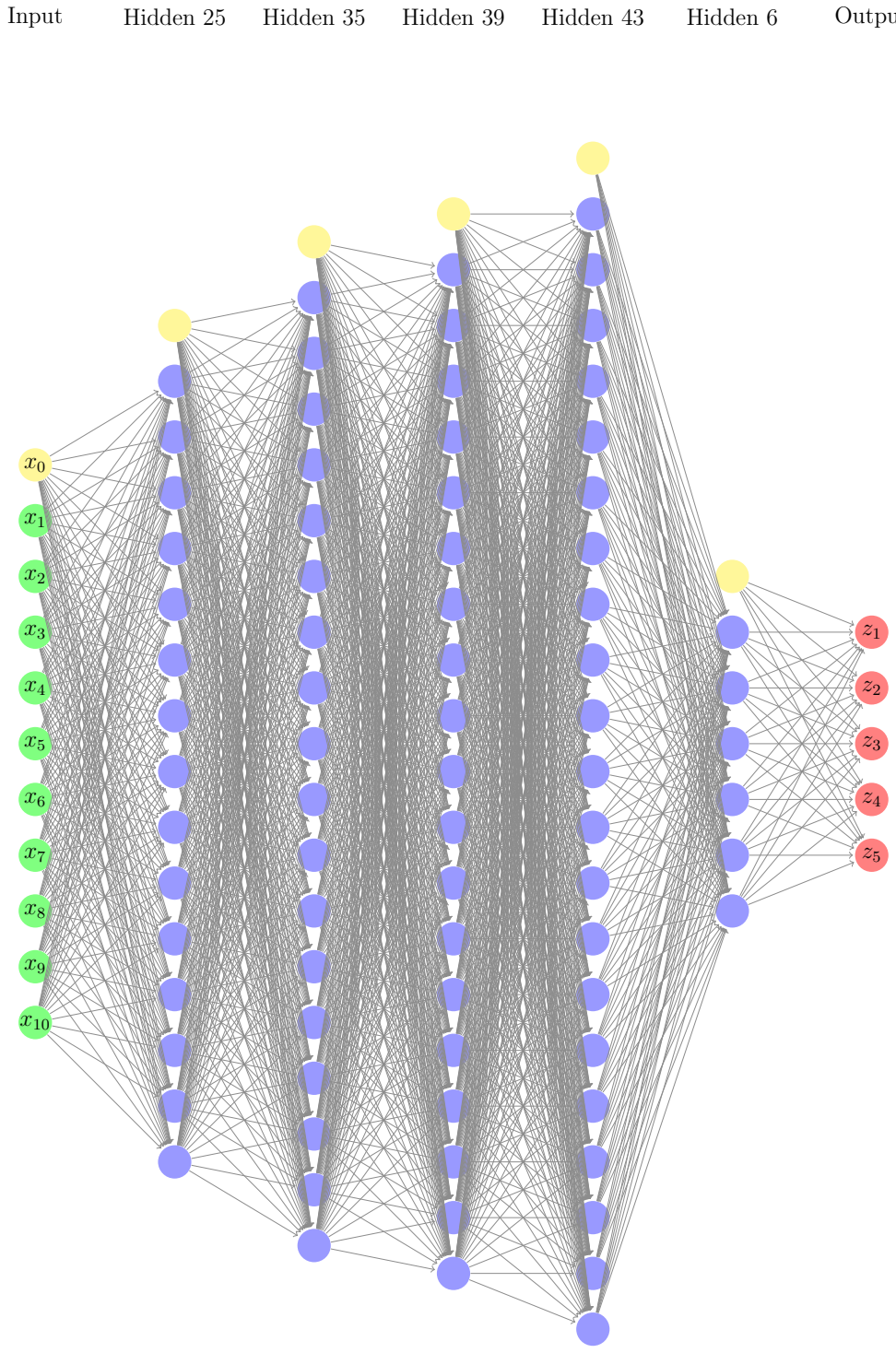

Figure 4.6: Neural network layout generated by the genetic algorithm

The genetic algorithm ran with an initial population of 10 individuals, using the Tournament selection method and a period of 10 generations. After this process the best individual selected is represented by the above architecture [4.6].

Chapter 5

Detailed Design and Implementation

5.1 Detailed diagram of components

In this chapter the elements of the system will be described. The following diagram [5.1] is an extension of the overview [4.1] presented in the previous chapter. The main modules from the overview are divided into their corresponding functions:

1. Data preprocessing

   Has as components preprocessing functions : data encoding, data splitting and feature scaling.

2. Feature selection

   Represents the component responsible for feature selection. This module contains all the feature selection methods which are evaluated one by one.

3. Generating neural network setup

   This module is divided into two main parts:

   (a) Choosing the neural network algorithm and parameters

   Feedforward using back-propagation and LSTM networks are evaluated using different combinations of parameters.

   (b) Genetic Algorithm layout selection

   Receiving the parameters from the previous function, the selection of the best layout is preformed using a genetic algorithm. The population of the algorithm is formed of Neural network layouts.

4. Neural network validation

   This step is performed using new unseen data, in order to see how much the algorithm was able to learn.

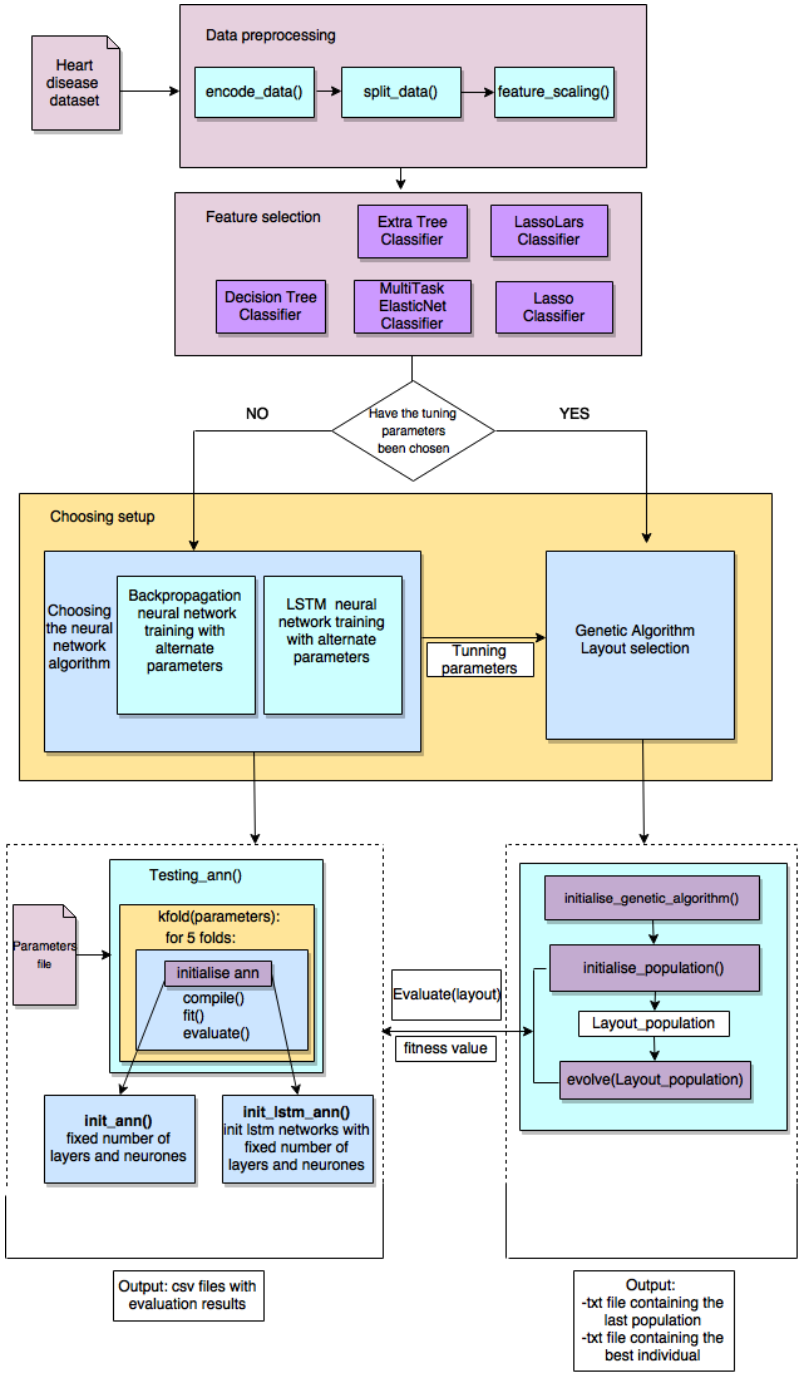

Figure 5.1: Detailed view of the implementation components

5.2 Preprocessing

The preprocessing step consists of operations performed on the dataset. The following set of operations are performed before feeding the dataset to the neural network. The accuracy of the neural network is tightly coupled to the dataset due to the fact that it is the starting point of all tuning that takes place inside the neural network.

Below there is an image of the dataset before the processing, this is important in order to understand the changes.

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
58	male	atypical	125	220	false	normal	144	no	0.4	flat	nan	reversable
56	male	atypical	130	221	false	hypertrophy	163	no	0	upsloping	0.0	reversable
56	male	atypical	120	240	false	normal	169	no	0	downsloping	0.0	normal
67	male	non-anginal	152	212	false	hypertrophy	150	no	0.8	flat	0.0	reversable

Figure 5.2: Dataset values before preprocessing

5.2.1 Encode data

This function performs two operations, the first one is cleaning the dataset by filing the missing values [1] and the second one is encoding the strings that are not into a numerical format [2].

1. Cleaning missing values

   The cleaning operations of the dataset are performed using Python's library Pandas. The dataset is loaded into a **pandas.Dataframe**, which allows us to modify easily the values in order to obtain a format acceptable by the algorithm.

   In order to fill in the missing values in a Dataframe, Pandas contains a method **pandas.Dataframe.fillna** that has multiple strategies available, the one that I choose is **ffill** 1a which propagates the last value that is valid to the next missing one.

   ```
   heart=heart.fillna(method='ffill')
   ```

2. Encoding values

   The data encoding process is performed using the library called **sklearn.preprocessing**. The first step of data encoding is choosing the categorical data, the values that are categorical are the following:

   (a) Gender - 2 possible values

   (b) Chest pain types - 3 possible values [3]

   (c) Fasting blood sugar - 2 possible values [6]

   (d) Resting electrographic results - 3 possible values [7]

(e) Exercise induced angina - 2 possible values [9]

(f) The slope of the peak exercise ST segment - 3 possible values [11]

(g) Thal - 3 possible values [13]

Using the LabelEncoder() class from sklearn.preprocessing, we transform the categorical values into numbers.

```
label_X_cp= LabelEncoder()
heart[['cp']]=label_X_cp.fit_transform(heart[['cp']])
```

As described in Chapter 4 [2] there is another type of encoding called OneHotEncoding, which represents categorical values as binary numbers, but performing some evaluations using this method resulted in a weaker accuracy than when using simple LabelEncoder.

1	2	3	4	5	6	7	8	9	10	11	12
0	2	120	209	0	1	173	0	0	1	0.0	1
0	2	122	213	0	1	165	0	0.2	1	0.0	1
1	0	130	254	0	0	147	0	1.4	1	1.0	2
1	0	125	300	0	0	171	0	0	2	2.0	2
0	1	126	306	0	1	163	0	0	2	0.0	1

Figure 5.3: Dataset values after encoding

5.2.2 Split data

Even though there are many approaches that split the dataset into train, test and validation, in my implementation in order to avoid the problem of overfitting 4.5.2 the splitting operation is done using the K-fold cross-validation method, where K=5 4.5.2. This will be described in more details in the following sections.

5.2.3 Feature scaling

As mentioned before this is an important step, that shouldn't be omitted due to the fact that different scales can cause the algorithm to perform poorly. This happens due to the fact that some learning algorithms consider that all data is centred at 0 and hence their variance is in the same order, this can cause features that have a significantly bigger variance to dominate the objective function.

The same library **sklearn.preprocessing** comes in handy, this contains a class called StandardScaler.

```
from sklearn.preprocessing import StandardScaler
sc_x=StandardScaler();
X=sc_x.fit_transform(X)
```

By default the function fit_transform, scales the data to unit variance and if needed it also transforms is.

0	1	2	3	4	5	6	7	8	9	10	11	12
-1.369	-1.471	0.974	-0.676	-0.747	-0.427	0.859	1.028	-0.680	-0.871	-0.655	-0.725	-0.562
-1.257	-1.471	0.974	-0.562	-0.666	-0.427	0.859	0.663	-0.680	-0.701	-0.655	-0.725	-0.562
0.972	0.680	-0.953	-0.108	0.166	-0.427	-1.041	-0.160	-0.680	0.319	-0.655	0.308	1.117
0.415	0.680	-0.953	-0.392	1.099	-0.427	-1.041	0.937	-0.680	-0.871	0.950	1.341	1.117

Figure 5.4: Dataset values after scaling

5.3 Feature selection

In order to see which feature selection method to choose, we trained the neural network with the types of feature selection mentioned in the previous chapter. Each of them will be evaluated and afterwards I will choose the best one.

The **sklearn** library provides implementations for Lasso, ElasticNet, ExtraTreeClassifier, LassoLars and Decision Tree.

Steps of using the Classifiers are:

1. Initialise Classifier and fitting it to dataset

```
clf = DecisionTreeClassifier()
clf = ExtraTreesClassifier()
clf = MultiTaskElasticNetCV(l1_ratio=0.5, eps=0.001, n_alphas=100,
    alphas=None, fit_intercept=True,normalize=False, max_iter=1000,
    tol=0.0001, cv=None, copy_X=True, verbose=0, n_jobs=1,
    random_state=None, selection='cyclic')
clf = LassoCV()
clf = LassoLarsCV()


clf = clf.fit(X, Y)
```

2. Selecting from the model the only the important features

```
model = SelectFromModel(clf, prefit=True)
model.transform(X)
```

The features selected for each of the Classifiers are the following:

1. Decision Tree Classifier

   After I applied the Decision Tree, the features were reduced to 8 from 13.

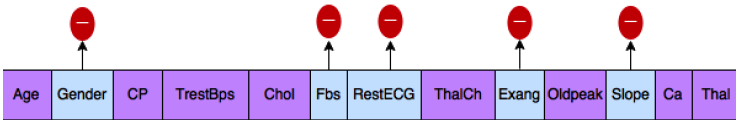

Figure 5.5: Feature scaling with decision tree

2. Extra Tree Classifier

   After I applied the Extra Tree, the features were reduced to 7 from 13.

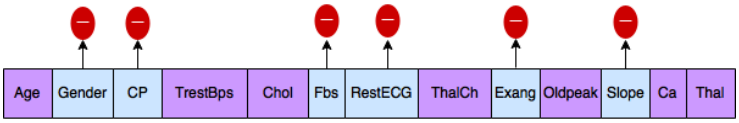

Figure 5.6: Feature scaling with extra tree

3. ElasticNet Classifier

   After I applied ElasticNet Classifer, the features were reduced to 7 from 13.

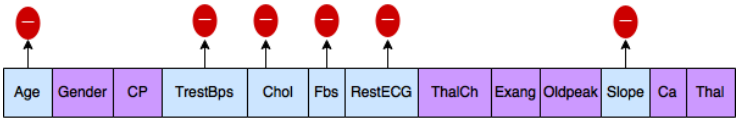

Figure 5.7: Feature scaling with ElasticNet

4. Lasso Classifier

   After I applied the Lasso Classifer, the features were reduced to 10 from 13.

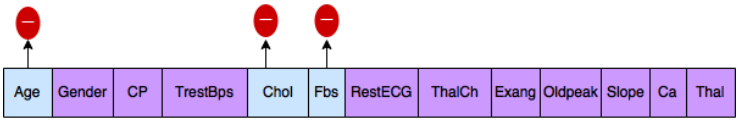

Figure 5.8: Feature scaling with Lasso

5. Lasso-Lars Classifier

   After I applied the Lasso Classifer, the features were reduced to 10 from 13.

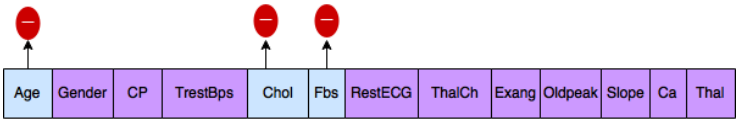

Figure 5.9: Feature scaling with Lasso-Lars

5.4 Choosing the tuning parameters

The library that I use to design the neural network is **Keras**, a high-level API that can run on top of Tensorflow. It is a user friendly framework that allows for both a high-level implementation as well as a low-level one by accessing Tensorflows methods.

The first step that I perform in designing my neural network is initialising neural networks with different parameters and cross-validate them in order to observe the results [4.5].

5.4.1 Initialising the neural network

1. Create a Sequential

   The Sequential is a Keras model, to which layers can be added. The layers can be added to the model either by using the Sequential constructor or by using it's add method.

   An important element when using Sequential is to add to its first layer the input shape, or the number of attributes of the dataset, this actually represents the initialisation of the input layer.

```
ann= Sequential()
ann.add(Dense(output_dim=int(layout.get_layers()[0].neurones),
                         input_dim=int(layout.get_input_size()),
                         kernel_initializer=init_w,
                         activation=activ_f))
```

2. Add layers

   Each layer that we add is a Keras core model called Dense. The Dense layer implements the neural network operation:

$$y = activation(x * weight + bias) \tag{5.1}$$

   Dense receives as parameters:

   (a) Input dimension

   The input dimension must be equal to the output dimension of the previous layer.

   (b) Output dimension

   (c) Init/Kernel initialiser

   This represents the strategy that we use to initialise the weights in the neural network.

   (d) Activation

The activation function that we we will use. Keras provides quite a large set of activation functions for us to use, but we also have the possibility to implement new activation functions and register them with this function.

```
classifier.add(Dense(output_dim=12,input_dim=X.shape[1],init=init_w
    ,activation=activ_f))
```

Apart from Layers, we can also add Dropout, either at the input layer or between hidden layers. This model works by setting a rate of inputs to zero hence removing the total connectivity between layers. Using dropout often helps with the problem of overfitting.

```
ann.add(Dropout(0.5))
```

3. Choosing the tuning parameters

As described at the previous point, Dense models take as parameters weight initialisation and activation. In order to have the best setup for the neural network, I trained the neural network model with all the available parameters [4.5.1]. I saved the neural network parameters into a csv file, and then read them and gave to the neural network initialiser. In this way I can observe how the neural network performs with different setups.

Below are all the possible values of parameters that Keras provides, these are also the values for which I evaluated the neural network.

Activation	Init	Optim	Loss
elu	zeros	SGD	mean_squared_error
softplus	ones	RMprop	mean_absolute_error
softsign	lecun_uniform	Adagrad	mean_absolute_percentage_error
relu	glorot_normal	Adadelta	mean_squared_logarithmic_error
tanh	he_normal	Adam	squared_hinge
sigmoid	he_uniform	Adamax	hinge
hard_sigmoid		Nadam	categorical_crossentropy
linear			sparse_categorical_crossentropy
			kullback_leibler_divergence
			poisson
			cosine_proximity

Figure 5.10: CSV File with parameters

5.4.2 Evaluate the neural network

After the initialisation of the neural network , the next step is to compile it, fit it and then evaluate it using K-fold method.

The Sequential model is the element that provides us with the possibility of performing these operations.

1. Compile

   This is the method responsible for the learning process. The parameters received by this method are the optimisation algorithm, the loss function and the desired metrics that we wish to use.

2. Fit

   This is the method responsible for the training of the neural network. One of its main parameters is the number of epochs for which we want the training to perform. Even though the algorithm would perform better if trained for more epochs, usually this value is restricted by the machine we are running the algorithm on.

3. Evaluate

   Computes the loss on some input data, the result is returned in terms of the metrics specified when compiling the model.

The initialisation, compilation and evaluation are all done inside the k-fold loop, due to the fact that we need to evaluate the performance of a newly trained algorithm and not a pre-trained one.

The K-fold belongs to the **sklearn.cross_validation**, the k-fold method is initialised with 5 folds. Inside the K-fold loop we initialise the model, compile, fit and evaluate it. The K-fold method works by splitting the dataset into k number of folds and it returns the indices of the train and test sets. The result is given by the mean of all the results from all of the 5 folds.

Each fold runs for 1000 epochs, that means that the score of a combination of parameters is given by the mean of 5 training session on a period of 1000 epochs.

```
kfold = KFold(categorical_labels.shape[0],n_folds=folds, shuffle=
    True)
    for train, test in kfold:
        ann=init_ann_model(layout)
        ann.compile(layout.params.optim_f,layout.params.loss_f,
            metrics=['accuracy'])
        ann.fit(x_train[train],categorical_labels[train],batch_size
            =10,epochs=1000,verbose=0)
        scores = ann.evaluate(x_train[test],categorical_labels[test
            ], verbose=0)
```

This evaluation is done for all the combinations of parameters, and afterwards I select the one that performs the best.

Apart from choosing the parameters this is the step where I choose the algorithm. The two algorithms that I evaluate are the Feedforward [1] algorithm and the LSTM algorithm [2a] .

The only difference when evaluation from LSTM is the initalisation. A LSTM needs to be added to the neural network Sequential model.

```
classifier.add(LSTM(6,input_shape=(trainX.shape[1],trainX.shape
    [2] ), activation=activ_f))
```

5.5 Genetic Algorithm Layout Selection

After having chosen the parameters, the layout is the next selection that I need to perform. This step is done using a genetic algorithm, designed using the **Deap** library. The first step when designing a genetic algorithm, is creating the individuals, in my case the individuals are possible layouts.

The Layout Class defines the individuals of the genetic population.

The attributes are the following: **input size, output size, layers** . The attribute layers contains elements of class Layer, these have as attributes the number of neurones, due to the fact that each hidden layer can have a different number of neurones.

5.5.1 Initialising the algorithm

Before starting the genetic algorithm, the Deap environment needs to be setup. Deap contains some predefined functions for the genetic algorithm, but in our case when the individual is a Neural network, the evolutionary functions need to be designed accordingly. Hence I need to define custom methods that randomise on the Layout class.

This methods are called evolutionary operators and in order for Deap to recognise them they need to be stored inside the Toolbox. The storing process is performed using the method of Toolbox called register.

Below I register the evolutionary operators needed for the algorithm:

```
toolbox.register("individual", create_random_nn,layout,creator.
    Individual)
toolbox.register("population", tools.initRepeat, list, toolbox.
    individual)
toolbox.register("mate", mate)
toolbox.register("mutate", mutate)
toolbox.register("select", tools.selRoulette)
toolbox.register("evaluate",ann_kfold)
```

1. Initialising individual function

   The individual initialisation is performed by creating random Layout objects, with random number of layers and random number of neurones in each hidden layer. The

number of layers I chose to vary between 1 and 10, while the number of neurones in the hidden layers vary between 2 and 50.

Creating individuals can be done one by one by using the command:

```
individual=toolbox.individual()
```

Another way to initialise individuals is by initialising the whole population, we can see that I registered the population concept in the toolbox. The way this works is by using the toolbox.individual() initialisation function repeatedly for the number of individuals that we want.

```
pop = toolbox.population(n=500)
```

2. Evaluation of individual function

   In order for an individual to be valid in the genetic algorithm, it must have a fitness value assigned. After creating individuals as described previously they automatically receive the attribute fitness which, initially has no value. The process of assigning a value to the individuals fitness is performed using toolbox.evaluate.

   The evaluation function that I choose to use is similar to the method that I used when choosing the tuning parameters, that is a k-fold evaluation with 5 folds and 1000 epochs per fold.

3. Mate function

   This might also be encountered also as crossover. It receives as parameters two individuals of type Layout and create 2 children individuals which inherit random features from either of the parents, such as number of layers or neurones in each hidden layer. This method of randomly choosing ”genes” from each parent gives both of them an equal probability of sharing their attributes. The crossover probability that I choose to use is 0.5.

4. Mutate function

   The mutation function receives as parameter a single individual, and it randomly mutates its characteristics. Mutation is an important method in genetic programming due to the fact that it enables diversity. The strategy that I choose to use is by randomly deciding whether or not to mutate a feature. If the mutation will occur a new random number of neurones or layers will be generated. The mutation doesn't always occur it has a probability of 0.3.

```
for mutant in offspring:
            if random.random() < MUTPB:
                toolbox.mutate(mutant)
```

5. Select strategy

   The genetic algorithm is ran twice, once using the Tournament Selection Method, and once using the Roulette Selection Method.

   The two selection methods are predefined in Deap library and the way to use them is by registering the selected function using Toolbox.

```
toolbox.register("select", tools.selRoulette)
toolbox.register("select", tools.selTournament,tournsize=3)
```

5.5.2 Genetic Algorithm flow

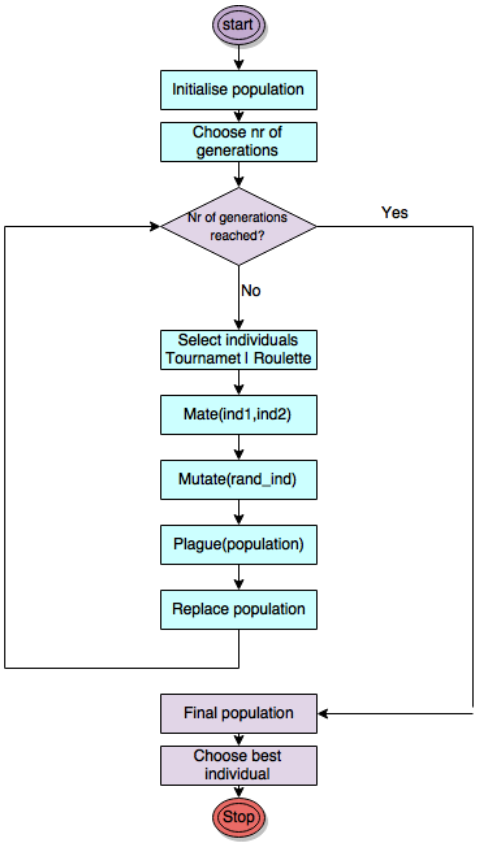

Figure 5.11: Genetic algorithm function

Using the methods defined using toolbox, the algorithm will perform a loop on a number of 10 generations. I will run this algorithm twice once using Roulette and once using Tournament. The small size of initial individuals is solely due to the time limitations.

5.5.3 Visualisation

During the genetic algorithm, several layouts are generated. In order to see the variation in layers and neurones I created a method that draws the neural networks created. This allows me to see how the algorithm evolves and to see whether the changes are big enough to allow evolution.

For drawing the neural networks I use **Pydot** library. I use the models attributes in order to see which component I need to draw.

The results generated are as follows:

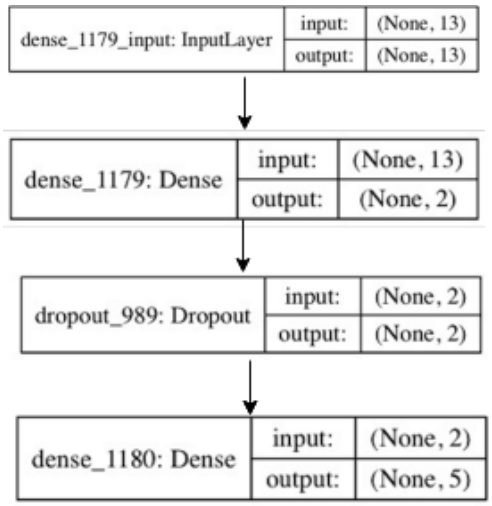

Figure 5.12: Generated layout

Chapter 6

Testing and Validation

In the following chapter I will present the results obtained after each of the training steps.

6.1 Feature selection results

In order to evaluate the best feature selection algorithm, I trained the neural network with a random fixed structure using the k-fold technique. For each of the training sessions feature selection was performed using a different algorithm. The results are the following:

Name	Score
Decision Tree	65.7650272065
Extra Tree	66.0928957645
Elastic Net	65.4371585299
Lasso	68.4480872643
Lasso Lars	70.4043711632

Table 6.1: Feature selection results

Therefore we can see that the method that has performed the best is the Lasso-Lars Feature Selection method. The evaluation was performed using a fixed size neural network with 2 hidden layers.

6.2 Parameters evaluation

Choosing the parameters was performed using an iterative approach that involves training the neural network with all the possible combinations of attributes (activation functions, optimisation algorithm, loss function, weight initialisation methods). I saved

the results in CSV files, in order to be able to easily visualise them and sort them. The number of results obtained is the cartesian product of the 4 sets of values:

Activation = A ={elu,softplus,softsign,relu,tanh,sigmoid,hard sigmoid,linear}

Weight init = W ={ones, lecun uniform, glorot normal, he normal, he uniform}

Optimisation = O ={SGD, RMSProp,Adagrad, Adadelta, Adam, Adamax, Nadam}

Loss = L ={ mean squared error, mean absolute error, mean absolute percentage error, mean squared logarithmic error, squared hinge, hinge, categorical crossentropy, kullback leibler divergence, poisson, cosine proximity}

$$|A \times W \times O \times L| = n * (A \times W \times O \times L)$$

$$n * (A \times W \times O \times L) = n * (A) \times n * (W) \times n * (O) \times n * (L)$$

$$n * (A) \times n * (W) \times n * (O) \times n * (L) = 8 * 5 * 7 * 10 = 2800$$

As the computations above show, I have evaluated 2800 neural networks, all with different parameters, and the results are as follows:

Activ	Init	Optim	Loss	ClassifRate	Acc	Hamming
softplus	he_normal	SGD	squared_hinge	0.7096774193548387	0.7096774193548387	0.2903225806451613
sigmoid	glorot_normal	Adadelta	squared_hinge	0.6774193548387096	0.6774193548387096	0.3225806451612903
sigmoid	glorot_normal	Adagrad	squared_hinge	0.6774193548387096	0.6774193548387096	0.3225806451612903
softplus	ones	RMSprop	poisson	0.6774193548387096	0.6774193548387096	0.3225806451612903
elu	ones	RMSprop	poisson	0.6774193548387096	0.6774193548387096	0.3225806451612903
sigmoid	ones	Nadam	poisson	0.6774193548387096	0.6774193548387096	0.3225806451612903
hard_sigmoid	glorot_normal	Nadam	poisson	0.6774193548387096	0.6774193548387096	0.3225806451612903
sigmoid	ones	RMSprop	mean_squared_logarithmic_error	0.6774193548387096	0.6774193548387096	0.3225806451612903
sigmoid	ones	Adam	mean_squared_logarithmic_error	0.6774193548387096	0.6774193548387096	0.3225806451612903
softplus	he_uniform	SGD	mean_squared_error	0.6774193548387096	0.6774193548387096	0.3225806451612903
sigmoid	ones	RMSprop	mean_squared_error	0.6774193548387096	0.6774193548387096	0.3225806451612903
sigmoid	ones	Nadam	mean_squared_error	0.6774193548387096	0.6774193548387096	0.3225806451612903
hard_sigmoid	glorot_normal	Adadelta	mean_squared_error	0.6774193548387096	0.6774193548387096	0.3225806451612903
hard_sigmoid	glorot_normal	Adamax	mean_squared_error	0.6774193548387096	0.6774193548387096	0.3225806451612903
softplus	glorot_normal	RMSprop	mean_absolute_percentage_error	0.6774193548387096	0.6774193548387096	0.3225806451612903
tanh	glorot_normal	SGD	mean_absolute_percentage_error	0.6774193548387096	0.6774193548387096	0.3225806451612903
softplus	ones	RMSprop	kullback_leibler_divergence	0.6774193548387096	0.6774193548387096	0.3225806451612903
sigmoid	ones	Nadam	kullback_leibler_divergence	0.6774193548387096	0.6774193548387096	0.3225806451612903
tanh	lecun_uniform	Nadam	kullback_leibler_divergence	0.6774193548387096	0.6774193548387096	0.3225806451612903
softplus	lecun_uniform	RMSprop	cosine_proximity	0.6774193548387096	0.6774193548387096	0.3225806451612903
softplus	he_uniform	SGD	cosine_proximity	0.6774193548387096	0.6774193548387096	0.3225806451612903
sigmoid	lecun_uniform	Adadelta	cosine_proximity	0.6774193548387096	0.6774193548387096	0.3225806451612903
sigmoid	ones	Nadam	categorical_crossentropy	0.6774193548387096	0.6774193548387096	0.3225806451612903
softplus	glorot_normal	Adagrad	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
softplus	ones	RMSprop	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
softplus	ones	Adam	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
softplus	he_uniform	SGD	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
softsign	he_normal	SGD	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
softsign	he_uniform	Adamax	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
sigmoid	lecun_uniform	Adadelta	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
sigmoid	lecun_uniform	Adagrad	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
sigmoid	he_normal	Adadelta	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
tanh	he_uniform	Nadam	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194
hard_sigmoid	glorot_normal	Adam	squared_hinge	0.6451612903225806	0.6451612903225806	0.3548387096774194

Figure 6.1: Example of results obtained after parameters evaluation

The statistics of the obtained data are as follows:

Name	Value
Minimum accuracy	0.0645161290
Maximum accuracy	0.7096774193
Mean	0.5423683586

Table 6.2: Feature selection results

In order to trim the number of results, I performed a selection of the best results for each activation function, this limited our results from 2800 to 37. The results obtained after trimming are displayed in the table [6.2].

	Activ	Min_Hamming_Loss	Accuracy	Init	Optim	Loss
0	softplus	0.290322580645	0.709677419355	he_normal	SGD	squared_hinge
1	softsign	0.354838709677	0.645161290323	glorot_normal	Nadam	kullback_leibler_divergence
2	softsign	0.354838709677	0.645161290323	ones	Nadam	poisson
3	softsign	0.354838709677	0.645161290323	lecun_uniform	Nadam	hinge
4	softsign	0.354838709677	0.645161290323	he_normal	SGD	poisson
5	softsign	0.354838709677	0.645161290323	he_normal	SGD	squared_hinge
6	softsign	0.354838709677	0.645161290323	he_normal	Adadelta	kullback_leibler_divergence
7	softsign	0.354838709677	0.645161290323	he_uniform	Adam	cosine_proximity
8	softsign	0.354838709677	0.645161290323	he_uniform	Adamax	squared_hinge
9	softsign	0.354838709677	0.645161290323	he_uniform	Adagrad	hinge
10	elu	0.322580645161	0.677419354839	ones	RMSprop	poisson
11	sigmoid	0.322580645161	0.677419354839	glorot_normal	Adadelta	squared_hinge
12	sigmoid	0.322580645161	0.677419354839	glorot_normal	Adagrad	squared_hinge
13	sigmoid	0.322580645161	0.677419354839	ones	Nadam	categorical_crossentropy
14	sigmoid	0.322580645161	0.677419354839	ones	Nadam	poisson
15	sigmoid	0.322580645161	0.677419354839	ones	RMSprop	mean_squared_error
16	sigmoid	0.322580645161	0.677419354839	ones	Nadam	mean_squared_error
17	sigmoid	0.322580645161	0.677419354839	ones	RMSprop	mean_squared_logarithmic_error
18	sigmoid	0.322580645161	0.677419354839	ones	Adam	mean_squared_logarithmic_error
19	sigmoid	0.322580645161	0.677419354839	ones	Nadam	kullback_leibler_divergence
20	sigmoid	0.322580645161	0.677419354839	lecun_uniform	Adadelta	cosine_proximity
21	tanh	0.322580645161	0.677419354839	glorot_normal	SGD	mean_absolute_percentage_error
22	tanh	0.322580645161	0.677419354839	lecun_uniform	Nadam	kullback_leibler_divergence
23	hard_sigmoid	0.322580645161	0.677419354839	glorot_normal	Nadam	poisson
24	hard_sigmoid	0.322580645161	0.677419354839	glorot_normal	Adadelta	mean_squared_error
25	hard_sigmoid	0.322580645161	0.677419354839	glorot_normal	Adamax	mean_squared_error
26	linear	0.354838709677	0.645161290323	glorot_normal	SGD	hinge
27	linear	0.354838709677	0.645161290323	ones	Adam	categorical_crossentropy
28	linear	0.354838709677	0.645161290323	ones	Nadam	cosine_proximity
29	linear	0.354838709677	0.645161290323	ones	RMSprop	mean_squared_error
30	linear	0.354838709677	0.645161290323	ones	Adam	mean_squared_error
31	linear	0.354838709677	0.645161290323	ones	Nadam	mean_squared_error
32	linear	0.354838709677	0.645161290323	ones	RMSprop	mean_squared_logarithmic_error
33	linear	0.354838709677	0.645161290323	ones	Adam	mean_squared_logarithmic_error
34	linear	0.354838709677	0.645161290323	ones	Adamax	mean_squared_logarithmic_error
35	linear	0.354838709677	0.645161290323	ones	Nadam	kullback_leibler_divergence
36	linear	0.354838709677	0.645161290323	lecun_uniform	Adagrad	hinge
37	linear	0.354838709677	0.645161290323	lecun_uniform	SGD	mean_squared_error

The same evaluation I repeated using a LSTM architecture but the results were as expected worse or equal to those obtained using a feedforward network. This is an expected result due to the fact that the classification does not depend on previously learned information as it is the case in applications like Natural language processing.

	Activ	Init	Optim	Loss	Acc
0	softplus	he_normal	SGD	squared_hinge	60.3232326106
1	softsign	glorot_normal	Nadam	kullback_leibler_divergence	59.9191916358
2	softsign	ones	Nadam	poisson	55.1043770968
3	softsign	lecun_uniform	Nadam	hinge	55.4612797045
4	softsign	he_normal	SGD	poisson	62.5117846226
5	softsign	he_normal	SGD	squared_hinge	59.1986526279
6	softsign	he_normal	Adadelta	kullback_leibler_divergence	66.9898991464
7	softsign	he_uniform	Adam	cosine_proximity	59.5151515143
8	softsign	he_uniform	Adamax	squared_hinge	74.7070709519
9	softsign	he_uniform	Adagrad	hinge	68.4175084135
10	elu	ones	RMSprop	poisson	68.8080801306
11	sigmoid	glorot_normal	Adadelta	squared_hinge	67.2996636473
12	sigmoid	glorot_normal	Adagrad	squared_hinge	65.0639730632
13	sigmoid	ones	Nadam	categorical_crossentropy	64.0404042078
14	sigmoid	ones	Nadam	poisson	67.3131305248
15	sigmoid	ones	RMSprop	mean_squared_error	63.6565659106
16	sigmoid	ones	Nadam	mean_squared_error	50.9966323865
17	sigmoid	ones	RMSprop	mean_squared_logarithmic_error	52.5185181501
18	sigmoid	ones	Adam	mean_squared_logarithmic_error	69.1582493132
19	sigmoid	ones	Nadam	kullback_leibler_divergence	65.4680134182
20	sigmoid	lecun_uniform	Adadelta	cosine_proximity	64.3501685493
21	tanh	glorot_normal	SGD	mean_absolute_percentage_error	54.4175079334
22	tanh	lecun_uniform	Nadam	kullback_leibler_divergence	57.9528614535
23	hard_sigmoid	glorot_normal	Nadam	poisson	64.4646462314
24	hard_sigmoid	glorot_normal	Adadelta	mean_squared_error	61.4074074724
25	hard_sigmoid	glorot_normal	Adamax	mean_squared_error	68.4107750658
26	linear	glorot_normal	SGD	hinge	61.40067366
27	linear	ones	Adam	categorical_crossentropy	63.9730633915
28	linear	ones	Nadam	cosine_proximity	69.4612791386
29	linear	ones	RMSprop	mean_squared_error	52.5252526469
30	linear	ones	Adam	mean_squared_error	59.4814816573
31	linear	ones	Nadam	mean_squared_error	62.4781144123
32	linear	ones	RMSprop	mean_squared_logarithmic_error	56.9225589009
33	linear	ones	Adam	mean_squared_logarithmic_error	57.2659929433
34	linear	ones	Adamax	mean_squared_logarithmic_error	62.8013466504
35	linear	ones	Nadam	kullback_leibler_divergence	66.8484852202
36	linear	lecun_uniform	Adagrad	hinge	65.4545458888
37	linear	lecun_uniform	SGD	mean_squared_error	63.1986533696

6.3 Genetic algorithm results

The first run of the genetic algorithm was performed using a Tournament Selection, the drawback of using this method is the fact that we can get stuck in local maxima. This is because we only focus on the characteristics of fit individuals.

The parameters of the first run were:

1. Size of initial population: 10
2. Selection method: Tournament
3. Number of generations: 10
4. Individual evaluation:

   K-fold with 5 folds and 1000 epochs

The initial population generated was the following:

1. **Population 0**

   Nr layers: 8

   Neurones for each layer:

   a) layer 0 - 9 b) layer 1 - 28 c) layer 2 - 29
   d) layer 3 - 16 e) layer 4 - 39 f) layer 5 - 32
   g) layer 6 - 31 h) layer 7 - 21

   Fitness value after evaluation: 58.181817910887972

2. **Population 1**

   Nr layers: 1

   Neurones for each layer:

   (a) layer 0 - 27

   Fitness value after evaluation: 58.868686606988362

3. **Population 2**

   Nr layers: 7

   Neurones for each layer:

   a) layer 0 - 11 b) layer 1 - 22 c) layer 2 - 4
   d) layer 3 - 26 e) layer 4 - 36 f) layer 5 - 16
   g) layer 6 - 32

Fitness value after evaluation: 58.808080963012756

4. **Population 3**

   Nr layers: 10

   Neurones for each layer:

   a) layer 0 - 30 b) layer 1 - 41 c) layer 2 - 30 d) layer 3 - 28
   e) layer 4 - 32 f) layer 5 - 21 g) layer 6 - 8 h) layer 7 - 29
   i) layer 8 - 35 j) layer 9 - 12

   Fitness value after evaluation: 57.867564316818068

5. **Population 4**

   Nr layers: 5

   Neurones for each layer:

   a) layer 0 - 26 b) layer 1 - 11
   c) layer 2 - 42 d) layer 3 - 16
   e) layer 4 - 5

   Fitness value after evaluation: 59.575757727478482

6. **Population 5**

   Nr layers: 3

   Neurones for each layer:

   (a) layer 0 - 29
   (b) layer 1 - 23
   (c) layer 2 - 38

   Fitness value after evaluation: 58.814814969746763

7. **Population 6**

   Nr layers: 10

   Neurones for each layer:

   a) layer 0 - 14 b) layer 1 - 37 c) layer 2 - 29 d) layer 3 - 40
   e) layer 4 - 31 f) layer 5 - 28 g) layer 6 - 45 h) layer 7 - 16
   i) layer 8 - 14 j) layer 9 - 29

Fitness value after evaluation: 56.655443325454804

8. **Population 7**

   Nr layers: 9

   Neurones for each layer:

   a) layer 0 - 33 b) layer 1 - 37 c) layer 2 - 13

   d) layer 3 - 13 e) layer 4 - 24 f) layer 5 - 22

   g) layer 6 - 20 h) layer 7 - 22 i) layer 8 - 14

   Fitness value after evaluation: 61.010100677759965

9. **Population 8**

   Nr layers: 1

   Neurones for each layer:

   (a) layer 0 - 46

   Fitness value after evaluation: 68.430976584302869

10. **Population 9**

    Nr layers: 1

    Neurones for each layer:

    (a) layer 0 - 8

    Fitness value after evaluation: 57.744107744107737

After 10 generations the best solution found the algorithm is:
Best population after 10 generations
Nr layers: 5
Neurones for each layer:

a) layer 0 - 25 b) layer 1 - 35

c) layer 2 - 39 d) layer 3 - 43

e) layer 4 - 6

Fitness value after evaluation: 71.430976584302869

6.4 Final model performance

The final model that was generated using the above described methods has proven to be especially successful classifying 2 labels at a time. This issues is due to the fact that the UCI Machine Learning Dataset has about half of the data labeled with 0, absence of disease, and half with all the other values, which shows that there are few examples from which the algorithm can learn the stages of heart disease.

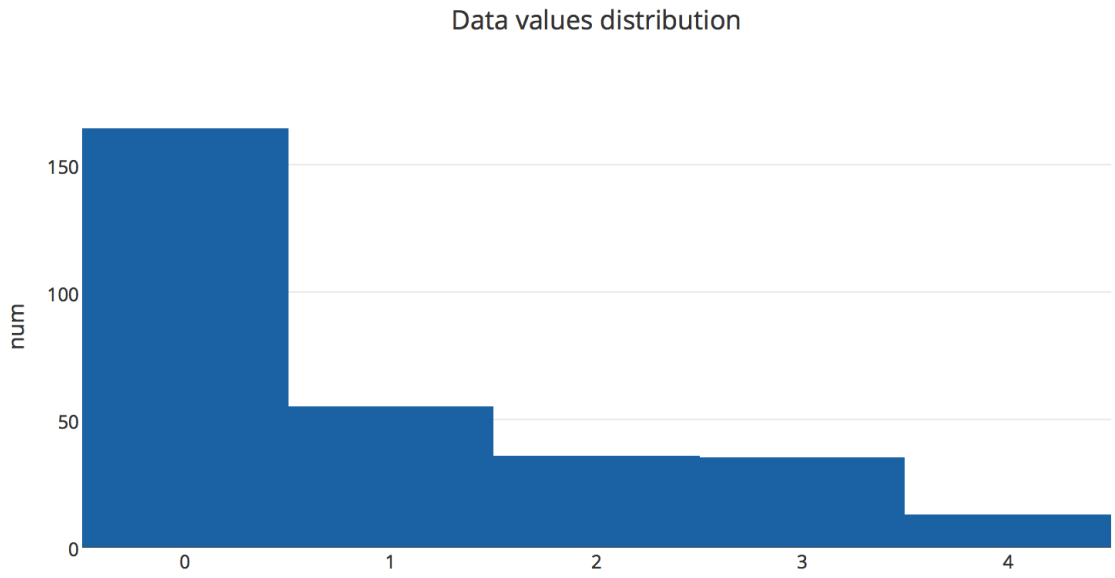

Figure 6.2: Data distribution

The method chosen to view how the algorithm predicts the classes is confusion matrix. The confusion matrix contains the correct predicted classes on the diagonal, this means that the more values on the diagonal the better the algorithm performs. The sum on each row represents the total number of values of each class. The other values represent the errors that the algorithm made.

Below is the confusion matrix generated by the layout described at [4.5.4]. Interpreting the results we can see that the algorithm receives as new data to predict :

1. 16 entries labeled with 0
2. 7 values labeled with 1
3. 3 values labeled with 2
4. 4 values labeled with 3

5. 1 value labeled with 4

The most correct predictions were made for label 0 that was predominant in the dataset. The diagonal contains the accurate predictions:

1. 15/16 entries labeled correctly with 0
2. 1/7 entries labeled correctly with 1
3. 1/3 entries labeled correctly with 2

Even though all the other values from {3,4} were badly labeled we can observe that the algorithm can very well distinguish between the absence(0) of a heart disease and its presence(1-4). Therefore instead of labelling the values with 3, they were labeled with 2, the same for 4 it was also labeled with 2.

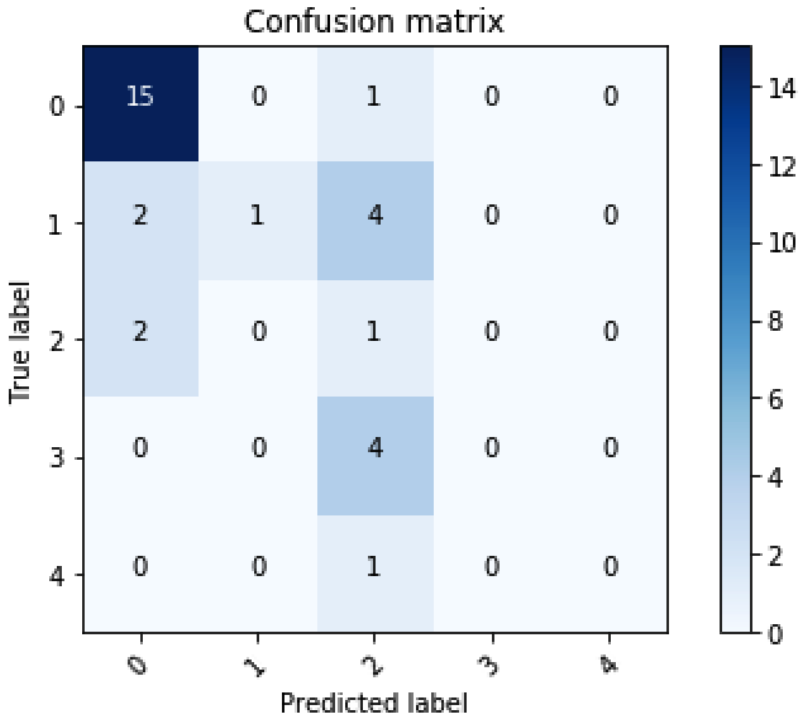

Figure 6.3: Confusion matrix

Chapter 7

User's manual

7.1 Environment setup

In order to run the application any Python Development Environment can be used, but I chose to use Anaconda due to the fact that you can create multiple environments with different setups. For example on one environment you have a Python 2.7 environment setup, while on another you work with Python 3.5.

For this application I choose to use **Python 3.5**.

Anaconda can be installed using the visual installer found on the Anaconda website. After having installed Anaconda, the following step is to install Miniconda, which will allow us to install packages on the environment using:

conda install

The Anaconda environment has 3 components **conda**, **conda-env** and **anaconda-navigator**. The versions for each of them are 4.3.22, 2.6.0 and 1.6.3.

7.2 Libraries

The application is developed using several python packages that need to be installed on the desired environment in order to be able to run the application. The packages can be installed either using *conda commands* which are specific to the Anaconda environment or by using *pip install* which is available on any system that has Python installed.

7.2.1 Installing tensorflow

Tensorflow is the backbone of the application being responsible for the deep learning operations. For the tensorflow installation we will use the anaconda environment and the commands are as follows:

1. Create a new environment for tensorflow

   conda create -n tensorlfow

2. Activate environment

   source activate tensorflow

3. Install tensorflow

   pip install ignore-installed upgrade Tensorflow_url

   Tensorflow_url can be retrieved from tensorflows website based on the operating system that the system will run on and based on whether or not we want it to run on the GPU or the CPU.

Having tensorflow installed will allow us to work with Keras, which is a Python library for deep learning, that runs on top of Tensorflow.

7.2.2 Other libraries

Using anaconda cloud, we will next install the following libraries:
Numpy
conda install -c anaconda numpy=1.12.1
Pandas
conda install -c conda-forge pandas=0.19.2
Scikit-learn
conda install -c conda-forge scikit-learn=0.18.2
Scipy
conda install -c conda-forge scipy=0.19.1
Matplotlib
conda install -c conda-forge matplotlib=2.0.2
Pydotplus
conda install -c conda-forge pydotplus=2.0.2
Keras
conda install -c conda-forge keras=2.0.5
Deap
conda install -c conda-forge deap=1.0.2.post2
Graphviz
conda install -c conda-forge graphviz=2.38.0

7.3 Running the algorithm

In order to train the algorithm the python file is opened in Anconda's IDE for Python, Spyder and the play icon is pressed. If we want to run only a piece of code we can also select it and press CMD+CTRL+ENTER (for OSX operating system).

Chapter 8

Conclusions

8.1 Results

During the development of the project I understood how neural networks work, what are their main drawbacks and how to improve them.

Research was a key action that I performed during the development of this project. This helped me understand the parameters of neural networks and the importance of choosing the correct setup.

The main goal of the project, designing a a neural network capable of classifying heart diseases was achieved by designing at first a basic 3 layered network. The following actions after this step all had the sole purpose of improving the prediction accuracy.

Having successfully designed an algorithm capable of generating neural networks with parameters read from a csv file provided me with an overview of how different changes affect the performance of the algorithm.

Another challenge that I choose to pursuit was evolving the neural network structure. In literature many methods are used but very few work on a general set of data.

After succeeding to implement the above mentioned solutions, the result that I obtained is an algorithm that is able to classify a heart diseases with an accuracy of 74%.

8.2 Observations

From the results of the parameter evaluation it can be observed the the parameters that affect the neural networks performance the most are the activation function and the loss function.

Another takeaway from the parameter evaluation is the fact that initialising the weights with zeros, leads to an algorithm that quickly kills all the neurones which has as final effect a system that doesn't learn anything.

8.3 Issues

When designing a neural network there are many issues that can be encountered. For most of them I succeeded in finding a satisfactory solution, but there still are some that do not depend on the system or the implementation but more on the logistics.

1. The biggest problem that I ran into is the dataset. Not having a large dataset in a field where connections are not easily observable can result in reaching a point where no more improvements can be made without having more data.

2. Another problem occurs when designing a genetic algorithm, this will not find the best solution unless the population size is very big. The problem of having such a population is the fact that the designed algorithm evaluates each algorithm using k-fold for 1000 epochs per fold. Therefore we can see how the evaluation of thousand of neural networks will take quite a long time.

Similar issues were encountered by researches at Stanford [40] university, that developed an algorithm capable of classifying 13 different types of heart arrhythmias. Having over 30,000 of clips as a training set still lead to a lot of cases where the algorithm failed, due to the fact that when talking about diagnosis there are a lot of particular situations that can ”break the rule”. We can therefore see how the algorithms accuracy is strongly affected by the lack of data, having access to only 300 data entries that were available for free.

8.4 Further development

Neural networks in health care is a subject that is currently under intensive development and therefore there are still a lot of changes that can be made in order to improve the system.

The first step in further developing this system is getting access to a larger dataset. Due to the area of the system, heart diseases, an interesting approach is enhancing the system with knowledge. Such systems are called Knowledge based system, and they have the advantage of being less susceptible to big mistakes.

After finalising the algorithm the next step would be incorporating the algorithm into smart gadgets that monitor individuals health activity in real time and could give real-time warnings when heart problems are detected.

8.5 Conclusion

I consider the results obtained to be very good, especially when classifying the presence or absence of a heart disease case in which the algorithm performs very well. When faced with the problem of observing the differences between the levels of gravity of

the heart diseases the algorithm still has some issues which are mainly due to insufficient training data. Having chosen to combine neural networks with genetic algorithms has opened my interest for hybrid algorithms, which I now consider to be the final point in solving most of the unresolved machine learning problems.

Bibliography

[1] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, 1995, vol. 2.

[2] G. B. Orr and K.-R. Müller, *Neural networks: tricks of the trade*. Springer, 2003.

[3] M. A. Nielsen, *Neural networks and deep learning*. Determination Press USA, 2015.

[4] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*. Springer, 2014, vol. 72.

[5] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016. [Online]. Available: http://arxiv.org/abs/1609.04747

[6] S. K. Kenue, "Efficient activation functions for the back-propagation neural network," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, vol. 2. IEEE, 1991, pp. 946–vol.

[7] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International journal of forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[8] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015. [Online]. Available: http://arxiv.org/abs/1511.07289

[9] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, vol. abs/1206.5533, 2012. [Online]. Available: http://arxiv.org/abs/1206.5533

[10] J. D. Olden and D. A. Jackson, "Illuminating the ?black box?: a randomization approach for understanding variable contributions in artificial neural networks," *Ecological modelling*, vol. 154, no. 1, pp. 135–150, 2002.

[11] F. Gers, "Long short-term memory in recurrent neural networks," *Unpublished PhD dissertation, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland*, 2001.

[12] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[13] D. Hunter, H. Yu, M. S. Pukish III, J. Kolbusz, and B. M. Wilamowski, "Selection of proper neural network sizes and architectures?a comparative study," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 2, pp. 228–240, 2012.

[14] S. Xu and L. Chen, "A novel approach for determining the optimal number of hidden layer neurons for fnn?s and its application in data mining," 2008.

[15] J. Sun, "Learning algorithm and hidden node selection scheme for local coupled feedforward neural network classifier," *Neurocomputing*, vol. 79, pp. 158–163, 2012.

[16] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[17] M. A. Jabbar, B. Deekshatulu, and P. Chandra, "Classification of heart disease using artificial neural network and feature subset selection," *Global Journal of Computer Science and Technology Neural & Artificial Intelligence*, vol. 13, no. 3, 2013.

[18] N. B. Amma, "Cardiovascular disease prediction system using genetic algorithm and neural network," in *Computing, Communication and Applications (ICCCA), 2012 International Conference on*. IEEE, 2012, pp. 1–5.

[19] M. Fathurachman, U. Kalsum, N. Safitri, and C. P. Utomo, "Heart disease diagnosis using extreme learning based neural networks," in *Advanced Informatics: Concept, Theory and Application (ICAICTA), 2014 International Conference of*. IEEE, 2014, pp. 23–27.

[20] I. S. F. Dessai, "Intelligent heart disease prediction system using probabilistic neural network," *International Journal on Advanced Computer Theory and Engineering (IJACTE)*, vol. 2, no. 3, pp. 2319–2526, 2013.

[21] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[22] C. Clabaugh, D. Myszewski, and J. Pang, "Feedforward networks," https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Architecture/feedforward.html.

[23] M. Bernacki, "Principles of training multi-layer neural network using backpropagation," http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html/.

[24] C. Stergiou and D. Siganos, "Neural networks," https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Contents.

[25] Wikipedia, "Cross-validation (statistics)," https://en.wikipedia.org/wiki/Cross-validation_%28statistics%29#k-fold_cross-validation.

[26] T. B.Arnold, "Weight initialization, momentum and learning rates," http://euler.stat.yale.edu/~tba3/stat665/lectures/lec15/lecture15.pdf.

[27] A. Karpathy, "Convolutional neural networks for visual recognition," http://cs231n.github.io/neural-networks-3/#baby.

[28] ——, "Convolutional neural networks for visual recognition," http://cs231n.github.io/neural-networks-1/.

[29] J. Brownlee, "Evaluate the performance of deep learning models in keras," http://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/.

[30] ——, "Overfitting and underfitting with machine learning algorithms," http://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/.

[31] Wikipedia, "Cross entropy," https://en.wikipedia.org/wiki/Cross_entropy/.

[32] C. Olah, "Understanding lstm networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[33] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks," http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[34] W. H. Organization, "Top 10 causes of death worldwide," http://www.who.int/mediacentre/factsheets/fs310/en/.

[35] B. Marr, "A short history of machine learning – every manager should read," https://www.forbes.com/sites/bernardmarr/2016/02/19/a-short-history-of-machine-learning-every-manager-should-read/#48a307915e78.

[36] J. R. Koza, "Genetic programming," http://geneticprogramming.com/tutorial/.

[37] "Loss and cost functions," http://image.diku.dk/shark/sphinx_pages/build/html/rest_sources/tutorials/concepts/library_design/losses.html.

[38] W. Anish Singh, "Types of optimization algorithms used in neural networks and ways to optimize gradient descent," https://medium.com/towards-data-science/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d3

[39] S. Ruder, "An overview of gradient descent optimization algorithms," http://ruder.io/optimizing-gradient-descent/.

[40] K. Brown, "Can an algorithm diagnose heart disease better than a person?" https://gizmodo.com/can-an-algorithm-diagnose-heart-disease-better-than-a-p-1796690061.

[41] N. N. G. S. GmbH, "Optimal network size," http://www.bestneural.net/Products/cVision/Technologies/NetworkSize.html.

[42] R. Gutierrez-Osuna, "Validation," http://research.cs.tamu.edu/prism/lectures/iss/iss_l13.pdf.

[43] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[44] D. Rucker, "Neurons and synapses," https://www.tes.com/lessons/zMaG-i_CdPxTNg/neurons-and-synapses.

[45] S. Miller, "Mind: How to build a neural network (part one)," https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/.

[46] "Scikit learn," http://scikit-learn.org.

[47] "Genetic algorithms - parent selection," https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm.

[48] M. Harvey, "Let?s evolve a neural network with a genetic algorithm," https://medium.com/@harvitronix/lets-evolve-a-neural-network-with-a-genetic-algorithm-code-included-8809bece164.