Linearisation of feed-forward artificial networks to study input importance.

Bachelor project Raoul Larsson **Supervisor** Mattias Ohlsson

August 19, 2013

Abstract

In the use of artificial neural networks (ANN) for real-life applications there is a need for determining the importance of each input variable for the decisions of the artificial neural network. By comparing more complex ANN:s with the simple perceptron, a basic method to determine some form of importance is found. This is tested on three different datasets of which one is based on a real life dataset of acute coronary syndrome (ACS). The results from these tests indicate that the method works.

Contents

1 Introduction

This project studies artificial neural networks (ANN), a type of learning algorithms with a wide variety of uses, e.g. recognition and data classification.

Amongst ANN:s there are different types of models, of which we focus on the single perceptron and the multiple-layer perceptron (MLP). The single perceptron is primary a linear classifier with an easy interpretation composed of several input-nodes and one output node, while the MLP have a similar but more complex organisation. This increase in complexity leads to the MLP being a non-linear classifier amongst other things [1][2].

While a single MLP can be used for classification by itself, it is often worthwhile to connect several of them into an ensemble, in this case taking the average over several independently trained networks [5]. This most often leaves us with a more secure basis for the classification for a not too heavy cost.

An example of the utility of ANN:s is for the usage of assisting in the classification of acute coronary syndrome (ACS), referring to any obstruction of the coronary arteries [4]. ANN:s have successfully assisted in determining if a patient is affected by ACS using the data from an electrocardiogram [7].

Of interest now, and also the purpose of this project, is to study the reason an ANN classified in the manner it did. One way to find this is by evaluating the importance of the input variables. The simple perceptron has a basic structure, making it easy to interpret the importance of the input variables by looking at the weights of the network. By linearising a MLP or an ensemble they can be given a structure similar to the simple perceptron.

The linearisation is done case-by-case, that is we take a trained network and linearise in the vicinity of a specific patient. This results in us knowing the importance of the input variables for that specific patient.

Having done the linearisation for the MLP and ensemble, these were implemented in Matlab, using its standard toolbox as a basis. Lastly, the implementation were tested using three different datasets with different purposes, of which one was a limited variant of the above mentioned ACS dataset.

2 Theory

2.1 Artificial neural networks and ensembles

As we utilize artificial neural networks (henceforth ANN) we begin by describing them [1].

ANN:s are a class of mathematical models that originated from the knowledge of the biological neural networks in the middle of the 20th century.

Although there is no general definition of an ANN, it usually involves nodes (neurons) connected to each other in some kind of network. These networks most often have some form of trainability, making them being able to solve a wide variety of problems. In this project we focus on a specific class of ANN:s, the feed-forward networks.

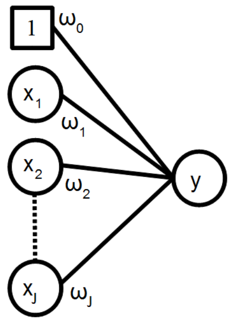

Figure 1: This figure illustrates how one can build up a simple perceptron. The x-values is the input-values, $\boldsymbol{\omega}$ the weights coupled to the x-values and y is the result. There is also an weight independent of the x-values, ω_0.

One common example of a simple feed-forward network is the single perceptron (see figure 1). The single perceptron has parameters $\boldsymbol{\omega}$, a vector of the weights building up the network. The weights define the network and are the trainable part of the network. Also often used is a bias, with $x = 1$, which in our case is included into the sum. Inputs are stored in a vector $\mathbf{x}$ and y is a function that maps the input to an output. We will in this project use the log-sig function below.

$$y(\mathbf{x}, \boldsymbol{\omega}) = \frac{1}{1 + e^{-f(\mathbf{x}, \boldsymbol{\omega})}}, \tag{1}$$

where

$$f(\mathbf{x}, \boldsymbol{\omega}) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_J x_J. \tag{2}$$

Here J is the number of input nodes.

The simple perceptron is a linear classifier, so that it can only classify linear separable patterns. While this limits its variety of uses, it sees lots of application around the world. An simple extension is the multiple layer perceptron (see figure 2) [2]. It of course resembles the simple perceptron, but introduces non-linearity. It can therefore be used for applications beyond separating linear patterns.

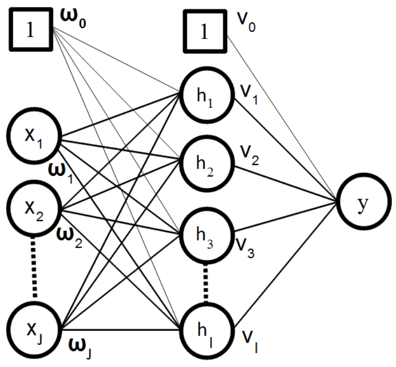

Figure 2: Multiple Layer Perceptron, this figure illustrates how a MLP can be constructed. There are inputs x_j with weight vectors $\boldsymbol{\omega}_j$ that links this output to nodes in the middle layer, or activation layer. These in turn are linked to the output node through the v_i weights. For each layer there is also a bias, with constant input 1 and weight $\boldsymbol{\omega}$ or $\mathbf{v}$.

As seen in figure 2 there is an additional layer of nodes between the input and output nodes. This layer is called a hidden layer and the nodes in this layer are called activation nodes. We have the following equation for the MLP:

$$y(\mathbf{x}, \mathbf{v}, \boldsymbol{\omega}) = \frac{1}{1 + e^{-f(\mathbf{x}, \mathbf{v}, \boldsymbol{\omega})}}, \tag{3}$$

with

$$f(\mathbf{x}, \mathbf{v}, \boldsymbol{\omega}) = \sum_i v_i h(\sum_j x_j \omega_{ij}) \equiv \sum_i v_i h(b_i). \tag{4}$$

Here h is the activation function. $\mathbf{v}$ is the weights between the activation nodes and the output nodes while $\boldsymbol{\omega}$ like for the simple perceptron is between the input nodes and the activation nodes. Index i loops over the hidden nodes and j over the input nodes.

While a MLP in general can have any amount of hidden layers and multiple output nodes we in this project focus on one hidden layer and one output.

Because the MLP is constructed of nodes with non-linear activation functions it is a non-linear classifier and if h is an sigmoidal function there exists a theorem, the universal approximation theorem, that states that the MLP can be used to approximate any continuous that maps an interval of real input values to an interval of output values [3].

It is mentioned above that the MLP can solve more problems than the simple perceptron, including non-linear classification. Adding more nodes can increase the computational cost to a problematic degree and one way to improve the performance of the algorithm is by independently training N number of networks, and add their result. The corresponding network is an ensemble [5]. The ensemble gives us better classification at the cost of having to train more networks. The ensemble of MLP:s has the following equation

$$y_{ens}(\mathbf{x}, \mathbf{v}, \boldsymbol{\omega}) = \frac{1}{N} \sum_{n=1}^{N} f(\sum_i v_{ni} h(\sum_j x_j \omega_{nij})), \tag{5}$$

where index n indicates ensemble member.

We will use both single MLP:s and ensembles.

For a two-class classification problem we can define a decision boundary, a hyper surface that divides the input space into the two different classes.

2.2 Linearisation of the networks

It is possible to directly interpret the importance of the variables for the simple perceptron because of the simple structure.

It is clear from equations (1) and (2) that $y(\mathbf{x}, \boldsymbol{\omega})$ only depends on $f(\mathbf{x}, \boldsymbol{\omega})$ monotonically. If f is high then that is true for y. This means that we can study f to understand the importances of each variable. For f it is easy to see that each variable only occur once, coupled with its relevant weight. This means that we can interpret the weight as the importance parameter of the single perceptron.

By doing a Taylor expansion for both the MLP and the ensemble for some case (or patient) and keeping first order terms only we should for both of them get expressions that mimics that of the simple perceptron.

For the MLP this can be done directly. We take one of the inputs and expand around that. Following the thorough calculation in appendix A we know that we can approximate $f(\mathbf{x}, \mathbf{v}, \boldsymbol{\omega})$, defined in equation (4), as

$$f(\mathbf{x}, \mathbf{v}, \boldsymbol{\omega}) \approx \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \cdots \tag{6}$$

where

$$\alpha_j = \sum_i B_i v_i \omega_{ij},\ j \neq 0 \tag{7}$$

and

$$\alpha_0 = \sum_i v_i \tanh b_i - \sum_j \alpha_j x_{0j}. \tag{8}$$

Here,

$$b_i = \sum_j x_{0j} \omega_{ij}, \tag{9}$$

$$B_i = \frac{1}{cosh^2 b_i} \tag{10}$$

and x_0 represents the inputs for the case we are linearising around.

For the ensemble there is some more tricks at hand. Equation (5) would be quite complex to linearise, so we write it in the form

$$y_{ens}(\mathbf{x}, \mathbf{v}, \boldsymbol{\omega}) = \frac{1}{1 + e^{-g(\mathbf{x}, \mathbf{v})}} \tag{11}$$

for some g. Solving for g and Taylor expanding that for one set of inputs (case), we for the ensemble get

$$g(\mathbf{x}, \mathbf{v}, \boldsymbol{\omega}) \approx \alpha_0 + \alpha_1 x_1 + \cdots . \tag{12}$$

According to the linearisation presented in appendix B, we have

$$\alpha_j = \frac{1}{m} \frac{\sum_n \frac{e^{-f_n}}{(e^{-f_n}+1)^2} \sum_i v_{ni} B_{ni} \omega_{nij}}{y_{ens}(1 - y_{ens})},\ j \neq 0 \tag{13}$$

and

$$\alpha_0 = g(\mathbf{x}_0, \mathbf{v}) - \sum_j \alpha_j x_0. \tag{14}$$

Here B_{ni} and y_{ens} are the results of the MLP and the ensemble for the case used.

We now have a form similar to the simple perceptron for both the ensemble and the MLP. Note that the intent is not for the linearisations to replace the ordinary network output, but to be able to compare input values in the same way we can for the simple perceptron.

What we want from the linearised ANN:s and the ensembles are the alpha values, or in some cases the product $x_j \cdot \alpha_j$. These correspond to the weights of the simple perceptron (or the weight-input product), and the importance of each linked input variable.

2.3 Datasets

Having done the linearisation we want to evaluate our approach of studying the importance of the input variables for both the MLP and ensemble. To accomplish this we utilize three different datasets, of which one is based on real life data.

2.3.1 Linear dataset

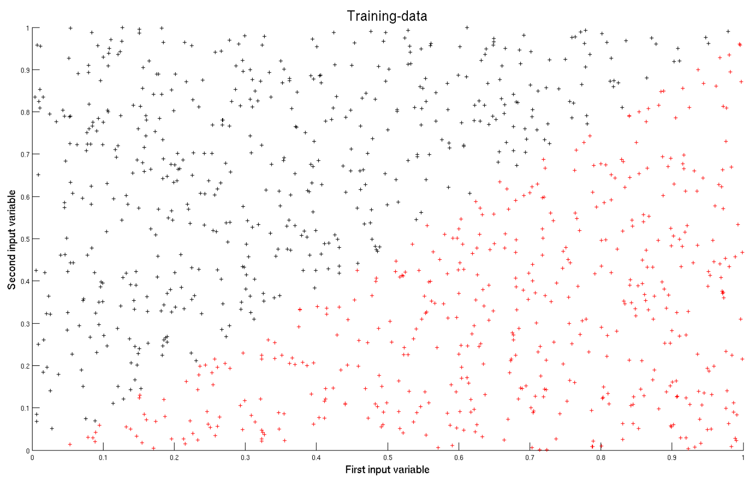

Figure 3: Linear dataset of 1000 data points, this figure shows the distribution of two different classes, black and red, over a two variable dataset. The black class is above the line $y = x$ and the red below.

The first dataset we utilise, using 1000 data points, is illustrated in figure 3. It is constructed using two input variables (being easy to create visualisation from), where class 0 is above the line $y = x$ and class 1 below it. This is a simple classification, easily solved by the simple perceptron, but it serves as an important first check for this project.

Remembering from the theory section of this project that we linearise to the form $f(\mathbf{x}, \boldsymbol{\omega}) \approx \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \cdots$, the dataset above having two input variables means that there is also two linearisation constants α_1 and α_2. With our straight linear line $y = x$ we expect a sufficiently trained network to also be linear or close to linear. This should lead to a constant set of alphas. We can also draw a conclusion from $y = x$ that both the inputs should have the same importance, meaning that α_1 and α_2 should have the same absolute value.

This leaves us a way to verify the linearisation by using the dataset above and checking if the linearisation of the dataset gives us the same alpha-values for the set.

2.3.2 Non-linear dataset

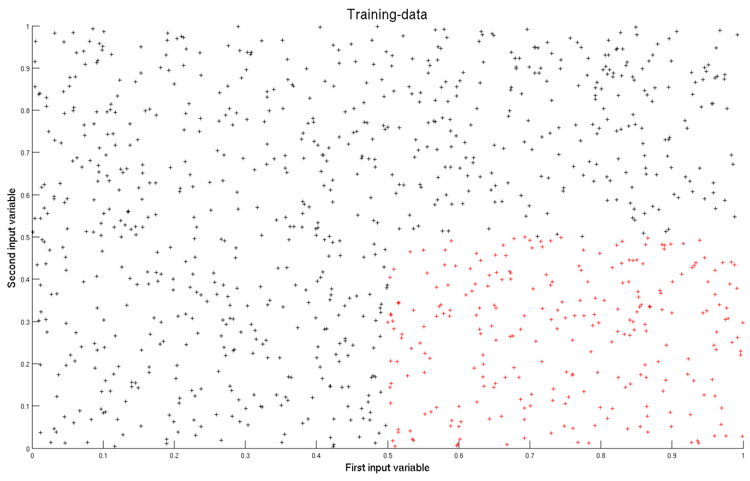

Figure 4: Non-Linear dataset, the distribution of 1000 data points divided into two classes, red and black. The red class is defined for the interval $x > 0.5$ and $y < 0.5$, and black for the rest $(0 < x, y < 1)$.

As seen in figure 4 the second dataset similar to the first only has two classes, two variables and 1000 data points. This non-linear dataset differs from the linear dataset by instead having class 1 in the interval $x > 0.5$ and $y < 0.5$ and class 0 for the rest $(0 < x, y < 1)$.

The difference in structure compared to the linear dataset gives us different expectations for the result. There are two major areas of the graph, the vertical decision boundary and the horizontal decision boundary. For the vertical part it is easy to see that movement from the boundary to the left or right changes the output while movement up and down changes nothing. The horizontal part in comparison works the other way, movement vertically changes the output and movement horizontally does nothing.

For our linearisation we can therefore expect that the relative value of the two αs change as we move along the decision boundary. As each input variable is more important in different parts of the decision boundary, the related α values should therefore also be of different sizes in the two areas.

2.3.3 Acute Coronary Syndrome

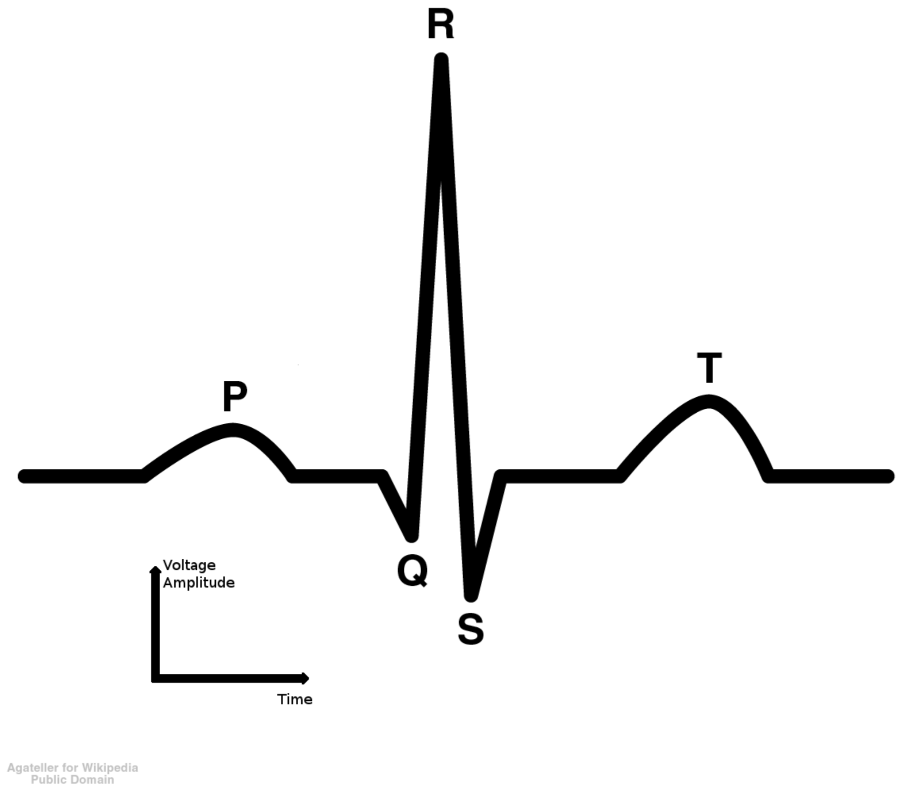

Figure 5: This figure illustrates how a typical ECG curve looks like

Having these more simplified datasets for testing implementation and overall function of the linearisation we want some way to correlate this to more real data [4]. It has already been shown that an ANN can successfully be used as a decision support in some cases relating to ACS, see [7], why we choose to use ACS to study the linearisation.

ACS refers to all syndromes that arise from obstruction of the coronary arteries. There are several ways of determining if a patient has ACS, of which one is using electrocardiogram (ECG). This is a way of measuring the electrical activity of the heart of a patient by putting up several nodes across the body (with more of them close to the heart) and measuring the electric activity between the different nodes. This measurements gives a large amount data in the form of several voltage amplitudes and durations. See figure 5 for a picture of a typical ECG graph.

The full dataset uses 3000 patients with 144 input variables [6]. In this project we are because of computational constraint limited to 17 input variables, see appendix C for a list.

3 Results

To implement the linearisation we used Matlab and its inbuilt toolbox for artificial neural networks. That is to say that the MLP, the functions and the training method used the toolbox while the ensemble, data-classes and the linearisation was implemented by hand. We used the hyperbolic tangent function as weight function and the log-sig activation function (as seen in section 2.1) as output function, these two are regularly used and work fine for the purposes of this project.

We use a decision boundary to check the trained networks for some datasets in this project. This is the region of the data for which the trained network results in a result close to 0.5. Our decision boundary is found by brute force, using a grid between 0 and 1 with 0.01 in distance between the grid points. This is fed into the already trained network and results between 0.45 and 0.55 are ruled as belonging to the decision line.

For the results we use 1000 data points for the linear and non-linear datasets. These two datasets are trained using the gradient descent method with momentum and adaptive learning rate backpropagation, a basic training algorithm.

The training of the ensemble uses 10 networks and uses the Baggings training method [5]. This is constructed by training each network on a bootstrap sample.

For the following subsections we use two kinds of figures, the first type shows the decision curve of the trained (non-linearised) network/ensemble and the second type shows the alpha values of the linearised network for each point on the decision boundary.

3.1 Linear dataset

For both the single MLP and the ensemble we used one single node in the hidden layer for the linear dataset, to constraint the networks to be linear.

3.1.1 One MLP

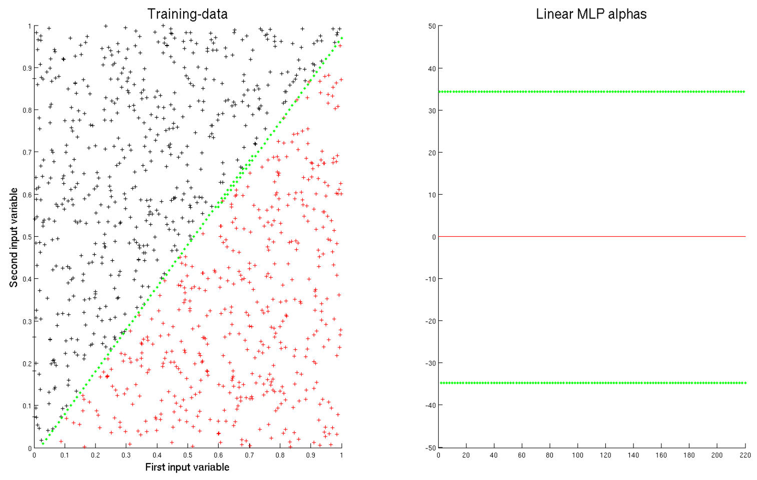

Figure 6: Results of linear MLP, figure (a) is the training result from the ordinary MLP on the dataset. (b) is the related alpha values going from down-right to up-left, with each point on (a) corresponding to two different alpha values in (b).

Figure 6 shows us the decision boundary of the trained MLP on the linear dataset. We clearly see that the decision boundary very closely follows that of $y = x$, the line separating the classes. This shows us that the MLP is linear and therefore that we can use the training and linearise the network to evaluate our conclusions from the theory section.

From figure 6 we see that the alpha-values are constant, as we move along the decision boundary they stay the same. It is also clear that the absolute value of both α_1 and α_2 are the same, meaning that the input variables are equally important to the decision. This means that our implementation is correct following the theory in section 2.3.1.

3.1.2 Ensemble

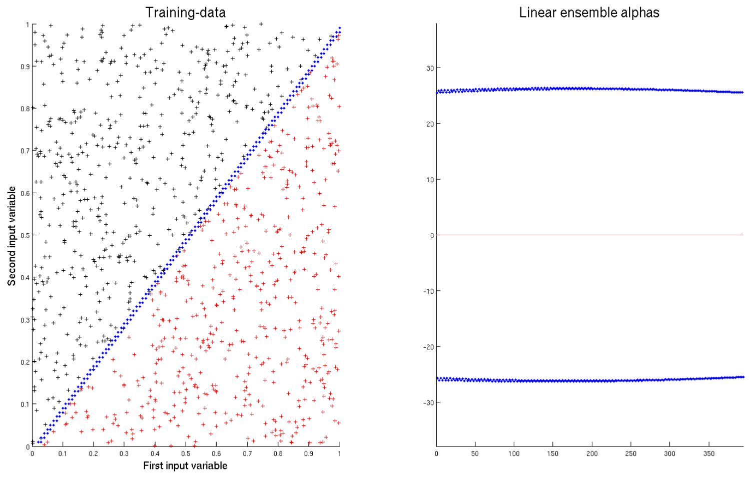

Figure 7: Results of linear Ensemble, figure (a) is the resulting training from the ordinary MLP on the dataset. (b) is the related alpha values going from down-right to up-left, with each point on (a) corresponding to two different alpha values in (b).

The result from the ensemble differs little compared to the MLP according to figure 7. The decision boundary follows the dividing line $y = x$. Notably is that compared to the MLP the ensemble is not linear, which can be seen barely from figure 7a. It is still clear that the ensemble is close to linear, giving us a result where the alphas are very close in value. It is also clear that the absolute value for the two α are the same for each point.

3.2 Non-linear dataset

The non-linear dataset is trained using 6 nodes in the hidden layer.

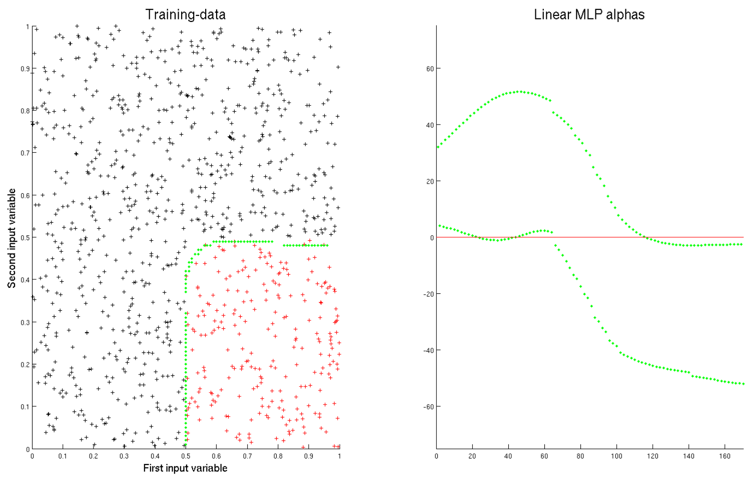

Figure 8: Results of non-linear MLP, figure (a) is the resulting training from the ordinary MLP on the dataset. (b) is the related alpha values going from down-right to up-left, with each point on (a) being enclosed in a red line on (b).

Looking at figure 8 and figure 9 we see that the trained MLP and the ensemble both creates a decision boundary which at large follows the rectangular division of the non-linear dataset. It is clear that the decision boundary has two major parts and one minor. There is a large horizontal part, one large vertical part and one where the two large parts meet. Because these decision boundaries follows the same basic feature as the true data, we can use the MLP and the ensemble for the linearisation.

It is clear from the figure that the alpha values follow the expectation from section 2.3.2. One of the alpha values dominate in the vertical part of the decision boundary, where only one of the input-variables really matters for the output, and the other alpha value dominate in the horizontal part. Differing from the true dataset used to train the network we for both the MLP and the ensemble get an intermittent part where both the input variables are important.

From this we can draw the conclusion that our method work for the non-linear dataset for both the MLP and ensemble.

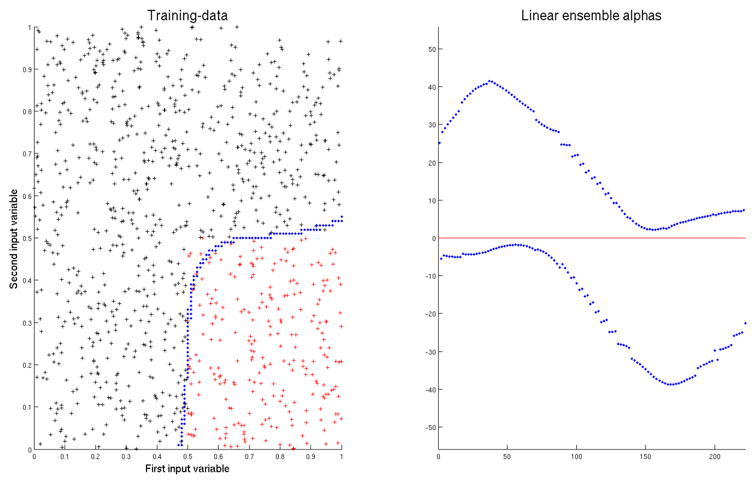

Figure 9: Results of non-linear ensemble, figure (a) is the resulting training from the ordinary MLP on the dataset. (b) is the related alpha values going from down-right to up-left, with each point on (a) being enclosed in a red line on (b).

3.3 ACS

The full dataset utilises 3000 patients with 144 input variables. We are as mentioned above limited to 17 input variables, but still the entire set of patients. See appendix C for a list of the inputs used.

The ACS-data uses the Levenberg-Marquardt backpropagation algorithm, one of the better algorithms for the environment used in this project [8].

We have too many variables to show the results as in earlier parts and instead we show the spread of alphas for every single patient. This lets us show that the alpha values are different for the various patients and that distinct input variables are not equally valuable.

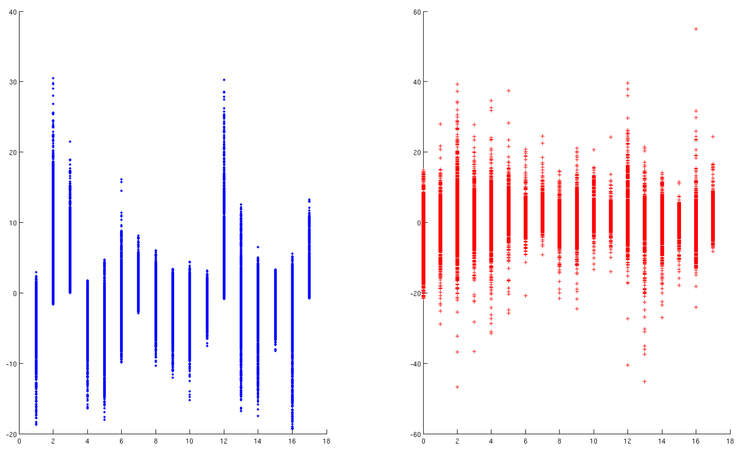

Figure 10: Graph of the results of the ensemble trained on the data on ACS, first graph shows the spread of the alphas and second shows the spread of the alpha x product

3.3.1 One MLP

Training the MLP using 6 hidden nodes we for the network get a classification performance of 90.27% for the training data and 87.90% for the validation data. From this we can be assured that while our MLP is not optimally trained, it is still a good result. We can therefore use this network for the linearisation purposes.

We clearly see from figure 10 that for both the different alpha values and the product $x_j \cdot \alpha_j$ there is a large spread of alpha values and $\alpha_j \cdot x_j$ products. The αs clearly are not the same for each input variable and some of the input variables has a smaller spread of α with the values closer to zero which shows that these generally are less important. This would suggest that the linearisation works for finding the importance of the input.

3.3.2 Ensemble

Like the MLP we trained the ensemble with 6 hidden nodes, giving a classification performance of 90.87% for the training data and 89.23% for the validation. This is obviously a better result than for the MLP, which is expected for an ensemble of MLP:s. As we use ten MLP:s in the ensemble the computational time takes about a magnitude more, which for our project is not really worthwhile (because we aren't focusing on optimising the networks).

The ensemble, like the MLP, shows a spread in the alphas according to figure 11. We accordingly can draw the same conclusion as for the MLP.

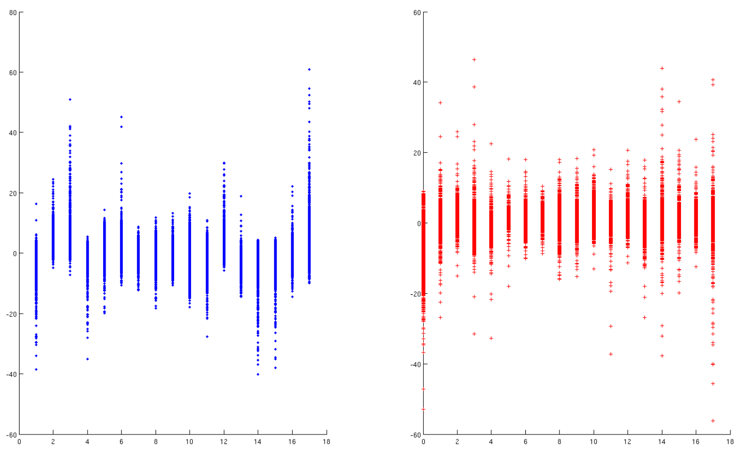

Figure 11: Graph of the results of the ensemble trained on the data on ACS, first graph shows the spread of the alphas and second shows the spread of the alpha x product

3.4 Discussion

All of the results for the three different datasets suggest that our method works. They all show that the different input-values have different alpha-values connected to them or a spread in the alpha-values. We can take a patient (or case), linearise about this patient and therefore get how important each variable is. Having this information is useful because then focus can be turned to the more important variables, and maybe even discarding less important ones.

Of note is that this method is not a replacement to the ordinary neural networks, but an add-on. It would not make sense to use the linearised networks instead of the standard networks, since the linearised networks require the results (including the weights) from the standard network. In addition for the linearisation we skip higher order terms, therefore introducing a large error to the result if we were to use the linearised network.

There is also the question of how true the alpha-values are. Do they represent the true relative importance of the input-values, or is the relationship between the alpha-values more complex. If we would keep higher order terms in the linearisation we would get cross terms of the input values. This of course mean that the linearisation would have a different interpretation.

We could use a different output and activation function. For the activation function this should trivially only introduce a slight change for the constants in the equations. Changing the output function, if we keep to other functions used for feed-forward networks, does not alter our interpretation of the single perceptron. More complexity is expected if we add additional hidden layers to the network, for which we can see an example if we compare the linearisation of the MLP and the ensemble.

A In-depth calculation of the linearisation for the MLP

Using the equations introduced in section 2.1 we here do the full calculation for the linearisation of the MLP. In this section n loops over all networks, i over output nodes and j over input nodes and for both i and j the bias-weights.

We have

$$f(\mathbf{x}, \boldsymbol{\omega}) = \sum_i v_i h(\sum_j x_j \omega_{ij}) = \sum_i v_i h(b_i), \tag{15}$$

where h is the hyperbolic tan-function, $\mathbf{x}$ is the input and $\boldsymbol{\omega}$ are the weights.

We Taylor-expand around $\mathbf{x}_0$, where $\mathbf{x}_0 = (x_1, x_2, \cdots)$:

$$f(\mathbf{x}, \boldsymbol{\omega}) \approx f(\mathbf{x}_0, \boldsymbol{\omega}) + (\mathbf{x} - \mathbf{x}_0)^T \cdot \nabla f(\mathbf{x}, \boldsymbol{\omega})|_{\mathbf{x}=\mathbf{x}_0} + \cdots, \tag{16}$$

where

$$\nabla f(\mathbf{x}, \boldsymbol{\omega}) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots) \tag{17}$$

and

$$f(\mathbf{x}_0, \boldsymbol{\omega}) = const. \tag{18}$$

We make the partial derivate for each element

$$\frac{\partial f}{\partial x_j} = \sum_i v_i \frac{\partial h(b_i)}{\partial x_j} \cdot \frac{\partial b_i}{\partial x_j} = \sum_i v_i \omega_{ij} \frac{1}{cosh^2 b_i} \tag{19}$$

and see that the full gradient can be written as

$$\nabla f(\mathbf{x}, \boldsymbol{\omega}) = \left(\sum_i \omega_i \omega_{i1} \frac{1}{cosh^2 b_i}, \sum_i \omega_i \omega_{i2} \frac{1}{cosh^2 b_i}, \cdots \right). \tag{20}$$

When the gradient is inserted into the Taylor-expansion, we notice, thanks to the scalar product, that we get an expression in the form

$$f(\mathbf{x}, \boldsymbol{\omega}) \approx \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \cdots \tag{21}$$

where

$$\alpha_j = \sum_i B_i v_i \omega_{ij}, \; j \neq 0 \tag{22}$$

and

$$\alpha_0 = \sum_i v_i \tanh b_i - \sum_j \alpha_j x_j. \tag{23}$$

Above expressions where made more non-cluttered by the use of below:

$$b_i = \sum_j x_j \omega_{ij} \tag{24}$$

and

$$B_i = \frac{1}{cosh^2 b_i}. \tag{25}$$

B In-depth calculation for the linearisation of the ensemble

For the ensemble we have

$$y_{ens} = \frac{1}{N}\sum_n y_n(\mathbf{x}) = \frac{1}{N}\sum_n \frac{1}{1+e^{-f(\mathbf{x},\mathbf{v})}}, \tag{26}$$

where N is the total number of independent networks in the ensemble.

We want the form

$$y_{ens}(\mathbf{x}) \approx \frac{1}{1+e^{-g(\mathbf{x},\mathbf{v})}}, \tag{27}$$

to be able to compare to the single-perceptron case. We solve for g:

$$e^{-g} = \frac{1}{y_{ens}} - 1 = \frac{y_{ens}}{1-y_{ens}} => \tag{28}$$

$$g = \ln\left(\frac{y_{ens}}{1-y_{ens}}\right) = \ln(Y) \tag{29}$$

Taylor-expansion around $\mathbf{x}_0$:

$$g(\mathbf{x},\mathbf{v}) = g(\mathbf{x}_0,\mathbf{v}) + (\mathbf{x}-\mathbf{x}_0)\cdot\nabla g(\mathbf{x},\mathbf{v})|_{x=x_0} + \cdots \tag{30}$$

$$\frac{\partial g}{\partial x_j} = \frac{1}{Y}\frac{\partial Y}{\partial x_j} = \frac{1}{Y}\cdot\frac{\partial y_{ens}}{\partial x_j}\frac{1}{(y_{ens}+1)^2} = \frac{1}{y_{ens}(1-y_{ens})}\frac{\partial y_{ens}}{\partial x_j} \tag{31}$$

$$\frac{\partial y_{ens}}{\partial x_j} = \frac{1}{M}\sum_n \frac{\partial}{\partial x_j}\left(\frac{1}{1+e^{-f_n}}\right) = \frac{1}{M}\sum_n \frac{e^{-f_n}\frac{\partial f_n}{\partial x_j}}{(e^{-f_n}+1)^2} \tag{32}$$

As $f_n = \sum_i v_{ni}h(b_{ni})$ we get

$$\frac{\partial f_n}{\partial x_j} = \sum_i v_{ni}\frac{1}{\cosh^2 b_{ni}}\omega_{nij} \tag{33}$$

If we enter above expressions in the Taylor expansion and make the correct insertions (eg. $\mathbf{x} = \mathbf{x}_0$) we note that we can write

$$g(\mathbf{x},\mathbf{v}) \approx \alpha_0 + \alpha_1 x_1 + \cdots. \tag{34}$$

Where

$$\alpha_j = \frac{1}{N}\frac{\sum_n \frac{e^{-f_n}}{(e^{-f_n}+1)^2}\sum_i v_{ni}B_{ni}\omega_{nij}}{y_{ens}(1-y_{ens})},\ j \neq 0 \tag{35}$$

and

$$\alpha_0 = g(\mathbf{x}_0,\mathbf{v}) - \sum_j \alpha_j x_0. \tag{36}$$

C Used variables for the ACS dataset

This section shortly describes the used data for this project. It is taken from [6].

Index of original dataset	Type
77	ST Amp II
78	ST 28 Amp II
79	ST 38 Amp II
80	ST Slope II
82	ST 28 Amp III
83	ST 38 Amp III
86	ST 28 Amp aVR
87	ST 38 Amp aVR
88	ST Slope aVR
93	ST Amp aVF
94	ST 28 Amp aVF
95	ST 38 Amp aVF
96	ST Slope aVF
123	T Pl Amp II
125	T Pl Amp III
130	T Mi Amp aVL
131	T Pl Amp aVF

ST refers to a specific segments of the ECG graphs while T stands for the T wave. Amp is the voltage amplitude of a segment while slope is the inclination. The last part is the lead, where the lead is defined as the output between two electrodes.

D Sources

[1] S, O, Haykins. *Neural Networks: A Comprehensive Foundation (2rd Edition).* (1999) p117-p150.

[2] S, O, Haykins. *Neural Networks: A Comprehensive Foundation (2rd Edition).* (1999) p156-p161.

[3] S, O, Haykins. *Neural Networks: A Comprehensive Foundation (2rd Edition).* (1999) p208-p209.

[4] `http://en.wikipedia.org/wiki/Acute_coronary_syndrome`

[5] D, Opitz. R, Maclin. *Popular Ensemble Methods: An Empirical Study* (Aug 1999)

[6] SE, Olsson. M, Ohlsson. H, Ohlin. S, Dzaferagic. ML, Nilsson. P, Sandkull. L, Edenbrandt. *Decision support for the initial triage of patients with acute coronary syndromes.* (2006)

[7] J, L, Forberg. M, Green. J, Bjork. M, Ohlsson. L, Edenbrandt. H, Ohlin. U, Ekelund. *In search of the best method to predict acute coronary syndrome using only the electrocardiogram from the emergency department.* (2008)

[8] W, Press. S, Teukolsky. W, Vetterling. B, Flannery. *NUMERICAL RECIPIES: The Art of Scientific Computing (3rd Edition)* (2007) p801-p805.