LU TP 16-25
May 2016

Applying the Maxout Model to Increase the Performance of the Multilayer Perceptron in Shallow Networks

Edvin Jakobsson

Department of Astronomy and Theoretical Physics, Lund University

Bachelor thesis supervised by Mattias Ohlsson

LUND
UNIVERSITY

Abstract

The Maxout network is an alternative to artificial neural networks that use fixed activation functions. By adding extra layers of linear nodes the Maxout network is able to learn both the relationship between the hidden nodes and the activation function that they use. The idea was presented as a natural companion to the dropout technique when training large convolutional networks, showing state-of-the-art results on several benchmark datasets. In this project we apply the Maxout method, without dropout, as a substitute to a sigmoidal activation function used in small networks with only one hidden layer. We show that the Maxout method can improve the performance of some classification problems while also decreasing the computer run-time needed for training. The classification problems used were artificially created to be non-linear multi-dimensional problems. The Maxout method was also tested on a highly complex regression problem and showed to yield as good results as the sigmoidal activation function, while taking a lot shorter time to train.

Populärvetenskaplig sammanfattning

Den mänskliga hjärnan är kanske det mest avancerade och komplexa system som vi känner till. Den är kapabel både till att skapa kreativa tankar och idéer och också till att skapa och minnas erfarenheter och lära sig från dem. Vetenskapen har kommit långt i utforskningen av hjärnans struktur, och idéer om hur vi kan ta till oss och använda den kunskapen är många. I hjärnan finns ett nätverk av nervceller som kommunicerar med varandra genom elektriska impulser. Idéen om att använda ett kommunicerande nätverk för att analysera och upptäcka mönster har väckt stort intresse i forskarvärlden. Konstruktionen av artificiella neurala nätverk har visat sig möjlig, och deras användbarhet tycks sakna gränser. Idag används de i allt från diagnosering av patienter till ansiktsidentifiering och självkörande bilar. Strukturen av sådana nätverk förändras konstant och det forskas intensivt på nya förbättringar för att göra framförallt stora nätverk snabbare och mer pålitliga. En ny struktur kallad Maxout har lagt grunden för det här projektet, som handlar om att analysera hur Maxout metoden presterar i små nätverk och på relativt simpla problem. Maxout metoden har visats vara användbar för djupa nätverk och komplexa problem som till exempel bildanalys och röstigenkänning, men dess brukbarhet för mindre nätverk är också relevant. Små nätverk är enklare att implementera, går fortare att träna och kan fortfarande vara mycket användbara när det gäller till exempel diagnostisering av patienter.

Contents

1 Introduction

The idea of the artificial neural network is based on the network of a brain. The simple perceptron is a start to modeling a neuron, receiving a series of signals and forming a signal to send forth. The multilayer perceptron is more complex, containing a network of neurons working together to perform more complicated tasks. This introduction attempts to give an understandable idea of how to artificially create a neural network.

1.1 Simple perceptron

1.1.1 Structure

In order to mimic the neuron in the human brain we start with a mathematical model of how we want to create an artificial neural network. We first need a series of input nodes, representing the data that we wish to analyze. We connect each input node to an output neuron through individual amplifiers that we call weights. When we feed this network with a set of input values the output neuron will summarize the product of the values and their respective weight. A bias term is added to the sum and it is then run through a chosen function we call activation function, in order to produce a result of the network in form of a number, see figure 1.

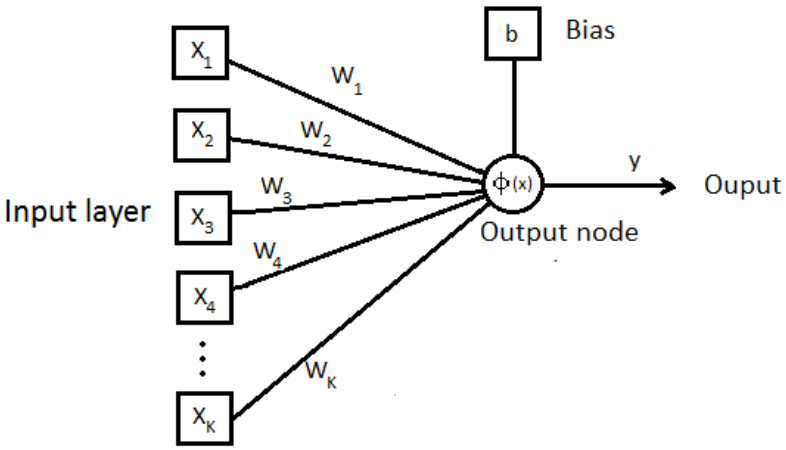

Figure 1: Structure of a simple perceptron with K number of input nodes. Signals are sent from left to right.

The output y is hence calculated as:

$$y = \phi(\sum_{k=1}^{K} w_k x_k + b)$$

We can simplify the equation by calling the bias term w_0 and adding an input value x_0 that is always 1. Also, if we write the input values $\{x_0, x_1, x_2, ..., x_k\}$ as a vector $\mathbf{x}$, and the weights as $\mathbf{w}$, this will enable us to rewrite the above equation as:

$$y = \phi(\sum_{k=1}^{K} w_k x_k + w_0 x_0) = \phi(\sum_{k=0}^{K} w_k x_k) = \phi(\mathbf{w}^T \mathbf{x})$$

The activation function $\phi(x)$ depends on the kind of problem we wish to solve, for example if we use the logistic function

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

the value of y will always be between zero and one, see figure 2.

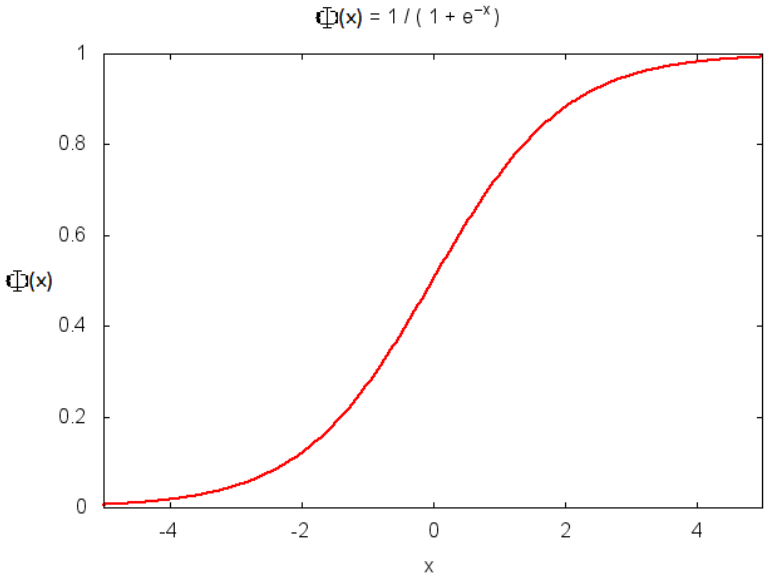

Figure 2: The sigmoidal function $\phi(x) = \frac{1}{1+e^{-x}}$. The value of $\phi(x)$ is bounded between 0 and 1.

The network can then by used as a classifier simply by dividing the output into different regions. If we decide to classify each data point that gives a result lower than a specific value, say $y = 0.5$, as belonging to class 0, and each data point that produces a higher value as class 1, the network will label any data point as either class zero or one, based on its input values. If we think of each input as a coordinate in a dimension we realize that the simple perceptron only divides the input space into two regions using a K-dimensional plane, where K is the number of input-nodes used. The properties of this plane are dependent on the amplifying weights of each of the connections, and by changing these weights we can change the decision plane in the input space as we please [1].

It is possible to use more than one output node in order to produce a multi-dimensional result from a single data point, however the datasets analyzed in this project are all using a one-dimensional target, hence I will not discuss multiple output nodes further.

1.1.2 Training the network

By using existing data we can train the perceptron into correctly classifying new data. Say we are given N input patterns that are already classified, each containing an input-vector $\mathbf{x}(n)$ and a class-label $d(n)$. This target $d(n)$ is either zero or one, depending on which class the pattern belongs to. Since we want our network to divide the input space so that the highest amount of patterns are correctly classified, we create an error-function:

$$E(\mathbf{w}) = \sum_{n=1}^{N} \Big(d(n) - y(n)\Big)^2 = \sum_{n=1}^{N} \Big(d(n) - \phi(\mathbf{w}^T\mathbf{x}(n))\Big)^2 \tag{1.1}$$

where $y(n)$ is the network output when fed the pattern $\mathbf{x}(n)$. The error E is dependent on the weight vector $\mathbf{w}$ and is decreasing as more patterns are classified correctly. By looking at the derivative of the error function with respect to $\mathbf{w}$ we can find out how we need to change the weights in order to decrease the total error of the network. The update of each weight during training can be calculated as:

$$\Delta w_k = -\eta \frac{\partial E}{\partial w_k} \tag{1.2}$$

where η is a constant called learning rate, and decides the size of the weight-change per update. This minimization method is called Gradient descent.

1.1.3 Limitations

Since the boundary between the class-regions created by the simple perceptron will always be linear, its uses are limited. The network will be inadequate when trying to solve a problem which requires a more complex boundary to separate the classes. A common example of this is the XOR problem [1], a two-dimensional classifying-problem containing only four data points. Class 0 contains two points located at $(-1,-1)$ and $(1,1)$, while class 1 contains the two points $(-1,1)$ and $(1,-1)$, see figure 3.

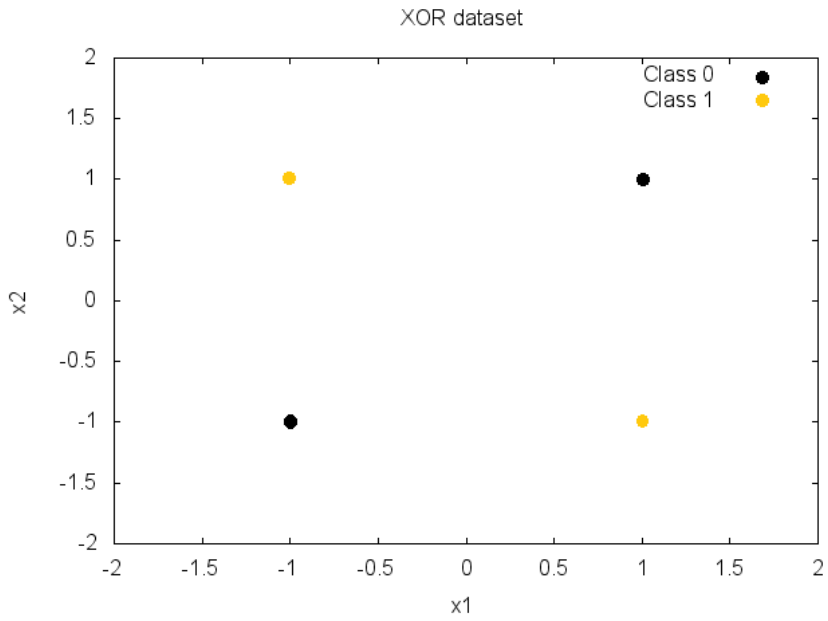

Figure 3: The XOR dataset. There are only two datapoints in each class, but it is clear that they can not be correctly separated using only one straight line.

Since the problem is only two-dimensional the dividing plane created by the simple perceptron will only be a one-dimensional line. It is obvious that the two classes can not be separated by a single straight line in the input space, hence the simple perceptron will never be able to classify all four data points correctly.

In 1969, fundamental limitations to the simple perceptron were mathematically proven which meant a setback for the research of artificial neural networks [3], and it probably killed the dreams of many scientists at the time. It was however soon discovered that complex multilayer structures did not have to be bound by the same limitations.

1.2 Multilayer perceptron

1.2.1 Structure

To solve the XOR problem we need a more complex network, such as the multilayer perceptron. By adding a hidden layer of nodes in between the input nodes and the output we equip the network with several independent weight vectors that can all contribute to the function of the network, see figure 4. The network will be more complex, and will be able to analyze more complex datasets.

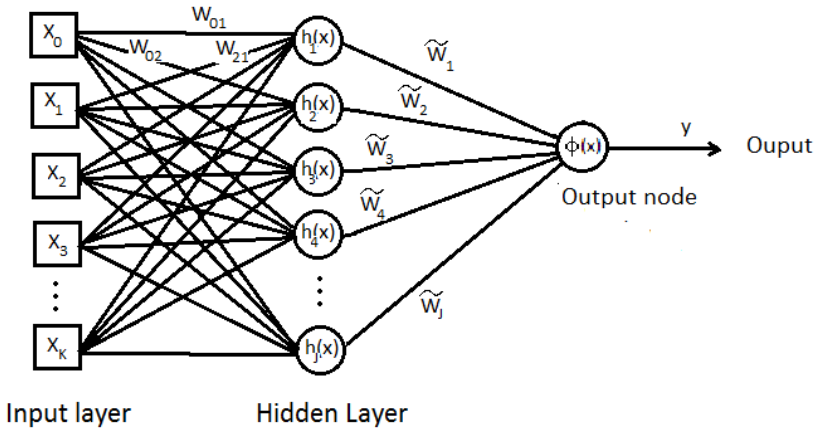

Figure 4: Structure of a multilayer perceptron with K input nodes, J hidden nodes and one output node.

The hidden nodes work in similar ways to the output node, as it summarizes the product of each input and a weight, and runs the result through a hidden function $h_i(x)$.

$$h_j(x) = h_j\big(\sum_k^K (w_{kj}x_k + b)\big) = h_j(\mathbf{w}_j^T\mathbf{x})$$

It is common to use a sigmoidal function as activation function for the hidden nodes [1]. In this project it was chosen as $tanh(x)$. There is, in fact, no use in having a linear hidden function as it would result in the same output as that of the simple perceptron [1]. Using a sigmoidal hidden function each hidden node will divide the input space into two parts and their combined contribution will make for a non-linear boundary [1]. Training the multilayer perceptron is done in the same way as the simple perceptron, see equations 1.1 and 1.2.

1.2.2 Solving the XOR problem

A network with only two hidden nodes is theoretically adequate to correctly classify all four points in the XOR problem, creating two decision-lines that can separate the two classes, see figure 5.

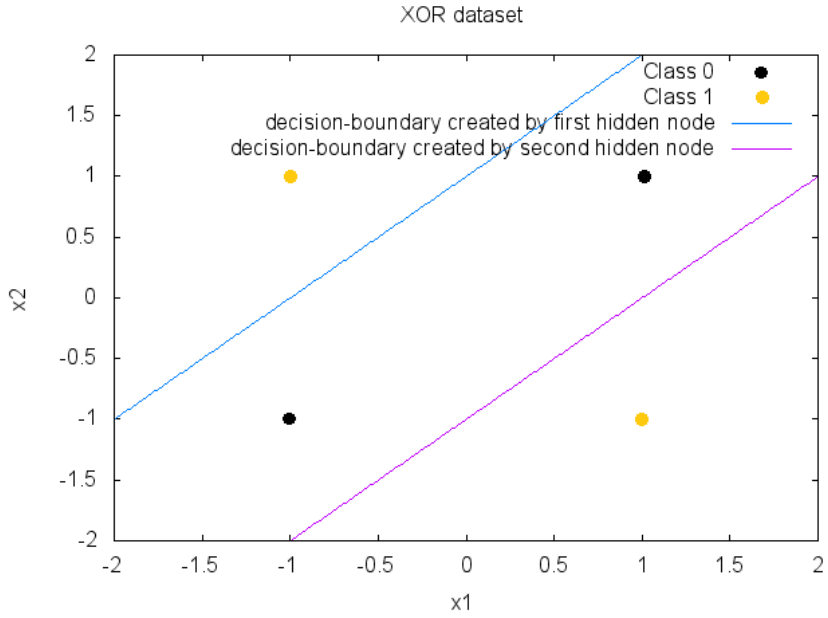

Figure 5: The XOR problem, solved by a multilayer perceptron using two hidden nodes.

1.2.3 Deep networks

In theory a multilayer perceptron with one hidden layer can, with a sufficient amount of hidden nodes, approximate any continuous function arbitrarily well [4]. However in practice it is common, when undertaking more complex problems, to expand networks rather by adding more hidden layers instead, see figure 6.

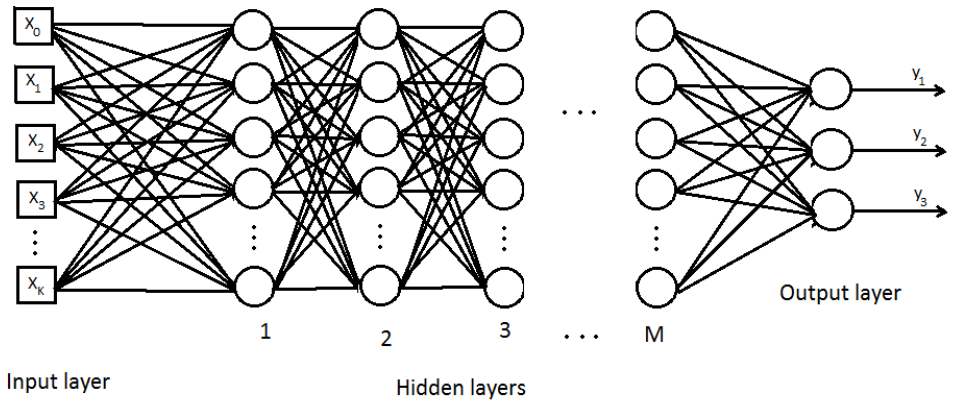

Figure 6: Structure of a deep multilayer perceptron with M hidden layers and three output nodes.

Deep neural networks are the frontier of artificial intelligence research and they have

proven to be extremely resourceful when dealing with large input data such as picture analysis and voice recognition [5]. The aim of this project however is to test if the *maxout unit* can improve the performance of shallow networks, hence the networks used in this project are all consisting of only one layer of hidden neurons.

1.3 Maxout networks

1.3.1 Structure

There are several different ideas as to what activation function $h(x)$ to use for the hidden neurons. Since functions can perform differently on different datasets the choice of function can become a parameter in the network itself. One recent contribution called the Maxout network was presented by Ian Goodfellow among others in 2013 [2]. A layer of linear nodes are added to every hidden unit, each connected to all the input nodes, see figure 7. These nodes use the activation function $y(x) = x$ and send a signal forward to the hidden node, which uses a new type of activation function: the maxout unit. The hidden node simply chooses the largest of the values produced by the linear nodes and sends it forward to the output node. The output will hence be calculated as:

$$y = \phi\big(\sum_{j=1}^{J} h_j(\mathbf{x})\big) = \phi\big(\sum_{j=1}^{J} \max_{l\in[1,L]}(\mathbf{w}_{jl}^T\mathbf{x})\big)$$

where J is the total number of hidden nodes and L is the number of linear nodes for each hidden node, while $\mathbf{w}_{jl}$ is the weight vector connecting the linear node l in maxout unit j to the input nodes.

Aside from training the weights between the hidden layer and the output node we will, in the maxout network, also train the weights connecting the input layer with the linear nodes. The structure of each hidden node's activation function can hence be altered during training. Note that there are no trainable weights in between the linear nodes and the hidden nodes. The function of such weights is already compensated for by the weights from the input layer, so they would be of no use to the network.

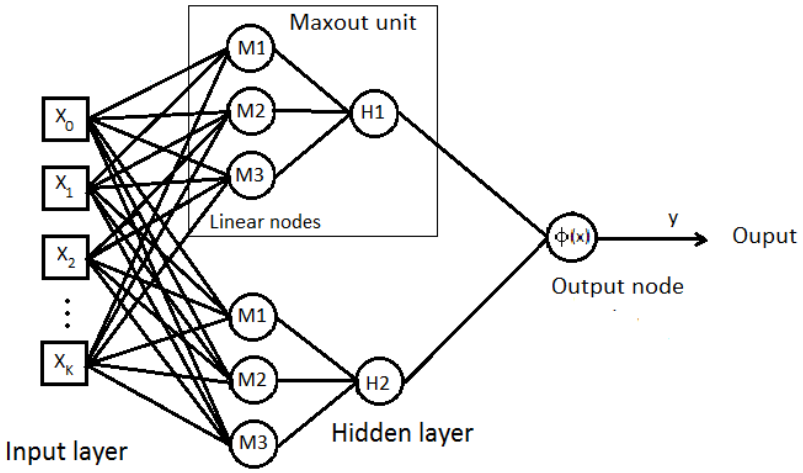

Figure 7: Structure of a maxout network with two maxout units each containing three linear nodes.

1.3.2 Interpretation

Each linear node can be interpreted as making a piecewise linear approximation to an arbitrary convex function [2]. This function will work as the new activation function for the hidden node. For each data point fed to the network the maxout node with the highest output value will be used, and its weight vector will be updated. The decision plane created by the maxout node will hence be altered, and the activation function for the hidden node will change. The multilayer perceptron is only focused on finding the proper relationship between its hidden nodes, while the Maxout network is also learning the activation function for each hidden unit, see figure 8.

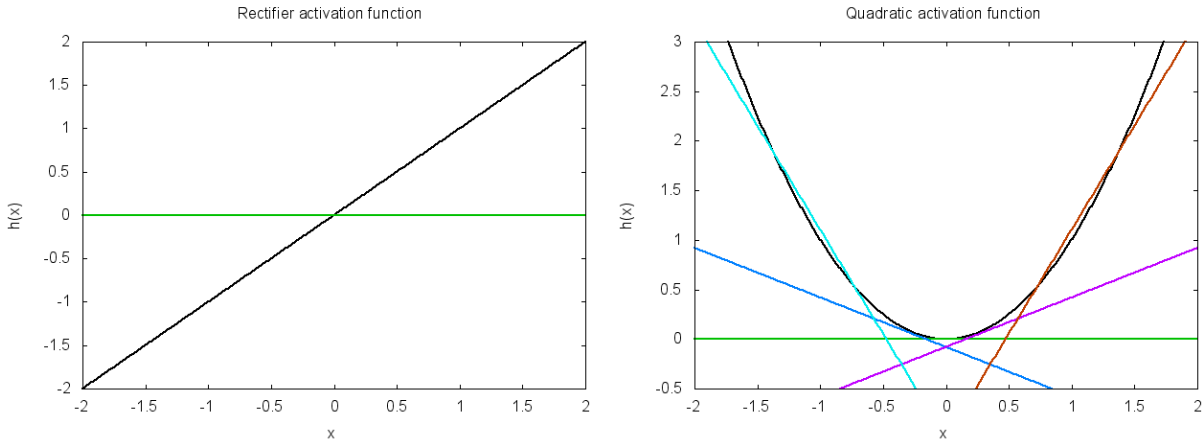

Figure 8: Examples of activation functions created by the Maxout unit. The highest value of the linear functions is always chosen, making the Maxout unit able to mimic the rectifier activation function (left figure), or a quadratic activation function (right figure). Note that the black quadratic function pictured in the right figure is not created by the Maxout unit, it is only pictured to show what the maxout activation function can approximate when using five linear nodes. The diagram is two dimensional and only shows how the Maxout activation function behaves using a one dimensional input.

The activation function created by the maxout nodes will always be a convex function, however with enough maxout nodes it can approximate any convex function arbitrary well. It is therefore also possible, only using two hidden units, to approximate any function [2].

Goodfellow shows that the Maxout network used with the dropout technique in deep networks can perform exceptionally well on a series of benchmark datasets, outperforming many previous techniques [2].

This project is focused on implementing the Maxout method on shallow networks, and no dropout has been used. The Maxout activation function was instead compared to the sigmoidal function used in ordinary multilayer perceptrons.

2 Methods

2.1 How to validate a network

2.1.1 Mean Squared Error

After a network has been trained its performance needs to be tested in order to evaluate its efficiency. We would like to know how well the network performs when analyzing data it has never seen before, hence we need a validation dataset. One easy way to evaluate the network is to measure the average squared error that it makes on the validation set, called E^{MSE}:

$$E^{MSE} = \frac{1}{N}\sum_{n}^{N}\big(d(n) - y(n)\big)^2$$

This will give us a number that can be used to compare the efficiency between different models, to see which type of network is most suitable for the problem.

2.1.2 Regression

When using a network for function approximation problems there may not be a limit as to what values the target data can take. This will make the mean squared error very dependent on the characteristics of the dataset. It can therefore be useful to look at the normalized mean summed squared error E^{NMSE}, calculated as the mean square error divided by the variance of the target data:

$$E^{NMSE} = \frac{\sum_{n}^{N}\big(d(n) - y(n)\big)^2}{\sum_{n}^{N}\big(d(n) - < d >\big)^2}$$

where $< d >$ is given by:

$$< d >= \frac{1}{N}\sum_{n}^{N} d(n)$$

2.1.3 Specificity and Sensitivity

When trying to create a good network for classification problems the mean squared error is not always a suitable measurement of the networks performance. Often it is more interesting to look at how many data points that were miss-classified, as it will give a better understanding of how useful the network is. A problem here is that the different classes can contain different amounts of data points, which will make them unequal in importance. For example if the validation set contains 90% data points belonging to class 1, a network that classifies every data point, no matter its features, as class 1 will still have a success-rate of 90%. It is therefore better to look at the average of the miss-classification in all the classes. Usually we are only trying to separate the data into two classes, and we call the success-rates of the two classifications specificity and sensitivity respectively. The performance of the network used for comparison hence becomes:

$$balanced\ \ accuracy = \frac{specificity + sensitivity}{2}$$

The sensitivity, or true positive rate, is named so for referring to the proportion of positives that are correctly classified as such, for example correctly identified people with a certain illness, and is usually class 1 in the target data. The specificity, or true negative rate, is the proportion of negatives that are correctly classified as such, for example people who were correctly classified as not having the illness [6].

2.1.4 K-fold Cross Validation

When validating the performance of a network it is usually tested on new data that has not been used for training. However the performance of a network is dependent on the amount of data that has been used for training, and it can be costly to set aside a part of the available data for a validation set, that could otherwise have been used to improve the network. The reliability of the validation performance is also dependent on the size of the validation set, meaning this set should be as large as possible.

A common method to find a better estimation of the performance is to divide all the available data into K parts, usually around five or ten, and create K different networks all using the structure we wish to validate. The K:th network then uses the K:th part of the dataset as a validation set while training on all the other parts, see figure 9.

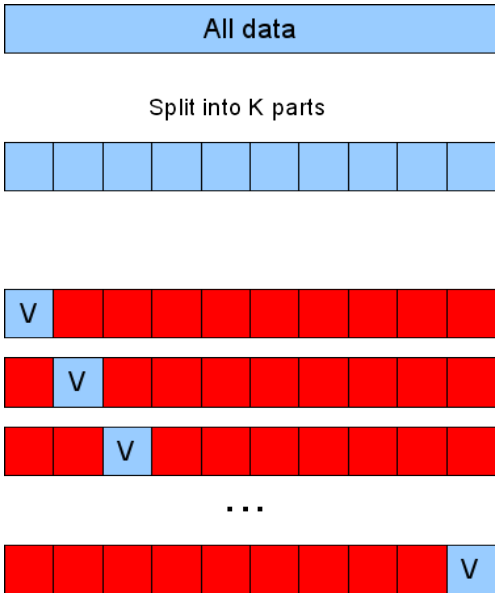

Figure 9: K-fold cross validation. For each network a different part (V) of the dataset is used for validation while the red part is used for training.

All the available data has hence been used both for training and for validation, while no single network has been validated on the same data that it used for training. The average performance of the K networks will provide a good estimation of the usefulness of the structure.

For the datasets used in this project a k-fold validation with ten splits was used and repeated 100 times, and the average result was used as an estimation of the validation.

2.2 Methods to increase performance

2.2.1 Winning bias

When training a Maxout network only the "winning" linear nodes, the ones creating the highest value for each hidden node, are updated. In deep networks the amount of linear nodes and the dataset used are both so large that this may not be a problem, but a concern was raised that a small network could get stuck in a local minimum of the error function. Since the area occupied by data points in the input space is very limited it is possible for a linear node to be outperformed by the others on every data point, and it will hence never be updated, a "dead" neuron. In order to prevent this a winning bias was added to every linear node that would increase if the node did not win, and reset as soon as it did, as following:

A winning bias constant α is added to amplify the signal of every linear node that did not win, making them more likely to do so during the next update. For a winning linear node the total bias is reset to zero. This would make all the nodes win over time, and force the network into using all its linear nodes. The winning bias constant α would become a parameter in the network that could be tuned in order to find its most useful value.

During the training the new equation for calculating the output of the network would become:

$$y = \phi\big(\sum_{j=1}^{J} \max_{l\in[1,L]}(\mathbf{w}_{jl}^T\mathbf{x})\big) \qquad \rightarrow \qquad y = \phi\big(\sum_{j=1}^{J} \max_{l\in[1,L]}(\mathbf{w}_{jl}^T\mathbf{x} + n_{jl}\alpha)\big)$$

where n_{jl} is the number of updates since the linear node l in maxout unit j last won.

When using the network for validation however the winning bias is always set to zero for every node.

2.2.2 Momentum

Gradient descent is a simple update-function that will stop as soon as it finds a local minimum of the error function. In order to increase efficiency and make the update-function able to pass local minimum in pursuit of better ones a momentum term was added. For each update of the weights a contribution from the previous update was added to the weight-change. The size of the contribution is decided by a momentum term β, which is a parameter that can be tuned to find the best performing update-function. The contribution will accelerate the minimization of the error-function as long as the updates has the same sign and decelerate it if they have the opposite sign. The enhanced gradient decent then becomes:

$$\Delta w_k = -\eta\frac{\partial E}{\partial w_k} \qquad \rightarrow \qquad \Delta w_k(t+1) = -\eta\frac{\partial E}{\partial w_k} + \beta\Delta w_k(t)$$

2.2.3 Regularization

Overtraining is usually a problem when training artificial neuron networks. The amount of data used for training is always limited, and a network that is allowed enough time to train might be able to learn the dataset completely. Since there is usually some sort of interfering noise in each data point an overtrained network will perform poorly when presented with new data. Regularization is the process to try and prevent a network from overfitting to the training data. The dropout method is a natural regularization method, but since no dropout was used in the project another type of regularization had to be implemented.

A method called max-norm regularization was used, which is named so because it restricts the norm of the weight vectors to a maximum value c [7]. We demand that

$$|\mathbf{w}_{jl}| \leq c$$

for every weight vector. If a vector is above the max-norm constant c its total length is shortened to the value c, without changing the direction of the vector.

An allowed maximum range c was set and after each update of the weights the length of each weight-vector was calculated and adjusted accordingly. This showed to prevent overtraining of the networks used.

2.3 Implementation details

The networks and training procedure in this project were implemented in Java from scratch. All the code used was created during the project, hence a complete understanding of the concept and mathematics of the multilayer perceptron was vital.

2.4 Datasets

The Maxout method was tested on five different datasets, and its performance was compared to that of the standard multilayer perceptron. The first four datasets are all classification problems with increasing complexity and dimensionality. The fifth dataset is a regression problem, in which the network is supposed to find the correlation between the method used to create cement and the cement's compressive strength.

2.4.1 Simple two-dimensional classification problem

The dataset contains 120 points divided evenly among two classes. It requires at least two linear class-boundaries in input space to be solved completely, see figure 10. The main idea is to see if a maxout network with only two linear nodes and one maxout unit is better at solving this problem than a multilayer network with two hidden nodes.

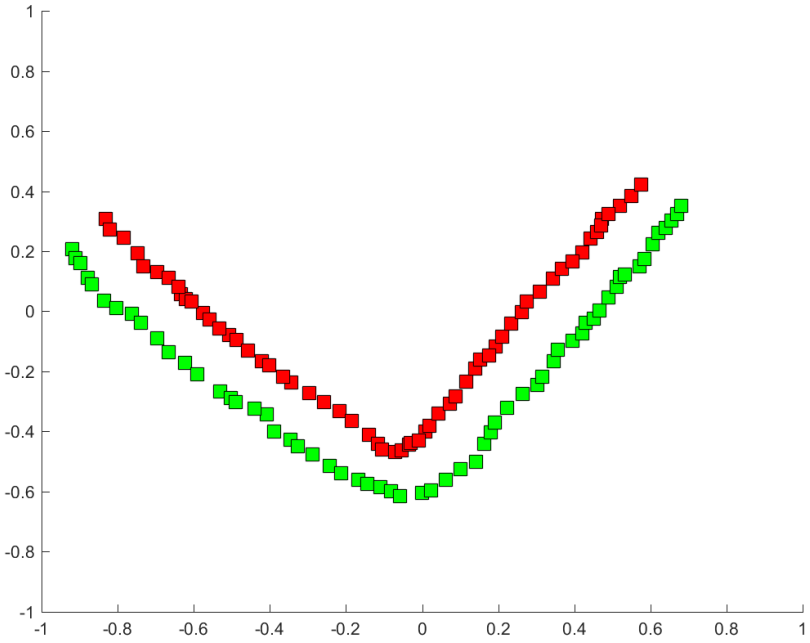

Figure 10: The complete Simple1 dataset. This is an artificially created dataset used to test the Maxout method. The top layer in red is class 0 and the bottom green layer is class 1.

2.4.2 Artificial non-linear classification problem

A dataset was created using the formula:

$$y = x_1 * x_2 + x_3 * x_4 + 2x_1^2 + 2x_2^2 + 2\sin(2\pi x_3) + b_1 * b_2 + 3b_3 * b_4 \tag{2.1}$$

where x_1-x_4 are normally distributed variables and b_1-b_4 are binary randomized variables. 1000 datapoints were created and the 500 with the highest y-value was labeled as class 1 while the other 500 as class 0. The function is chosen so that a multidimensional solution is necessary, and a network with only one or two hidden nodes is expected to show a weak performance on the dataset.

2.4.3 Altered non-linear classification problem

This is an altered version of the above dataset, where four extra input values has been added. The class assigned to a data point is however completely independent of these extra input values, their only purpose is to throw the network off, making it harder to correctly classify the data points. Another 1000 datapoints were created using equation 2.1, and the four extra input values were added.

2.4.4 Concrete data

The strength of concrete created is based on several factors such as time and many ingredients, making the process very complex. This dataset contains the strength found in concrete created when experimenting with eight different parameters, making it an eight-dimensional regression problem [8]. The dataset used contained 700 datapoints.

3 Results

The datasets were tested using many differently structured networks, and average performances over 100 models were used for comparison. Most results are presented by plotting the validation of different networks while the amount of hidden nodes/maxout units is increased.

For the problems that were analyzed by performance rather than success-rate, such as the two artificial classification problems and the regression problem, each model used a k-fold cross validation with ten splits to estimate the validation of the network modeled. For the XOR problem and the two-dimensional classification problem k-fold cross validation was unnecessary, since the networks were valued based on their chance to find a correct solution.

The momentum term was tuned separately for all problems, but a value of 1 showed to be the optimal choice for all of them. This means that the complete step in weight-space made by the previous update was added to every new update.

The max-norm constant c was not used in the XOR or the two-dimensional problem since they can be solved completely. For the last three datasets overtraining could be an issue, hence a max-norm constant had to be tuned. For the two classification problems a max-norm value of 2.5 was found to prevent the networks from overtraining, while not hindering the networks from producing the optimal results. For the regression problem no overtraining was visible, and so no regularization

3.1 XOR

The XOR problem is theoretically solvable with a multilayer perceptron using only two hidden nodes, however the success-rate of such a network was below 50 % over 1000 trials. When increasing the amount of hidden nodes the success-rate of the networks rapidly approached 100 %. The multilayer perceptron was however greatly outperformed by the Maxout method, even when a fewer number of hidden nodes were used in the Maxout network, see figure 11, left picture.

When a winning bias was implemented the Maxout networks success-rate was improved even further, see figure 11, right picture. The winning bias parameter was tuned and the best value found was 0.35. With this winning bias implemented even the smallest possible Maxout network, using one maxout unit with two linear units, was able to solve the XOR problem almost every time.

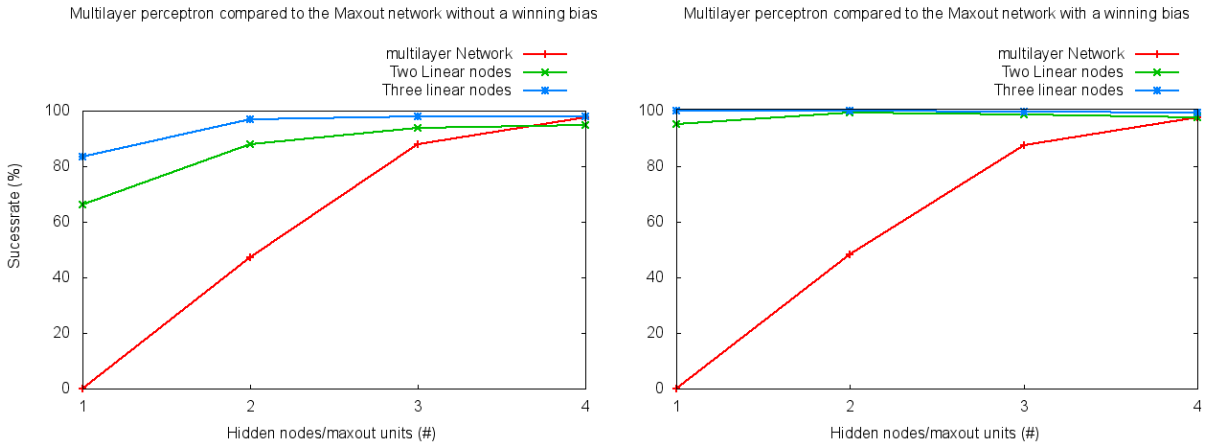

Figure 11: Success-rate comparison between the MLP and the Maxout method on the XOR problem. The Maxout networks clearly outperforms the multilayer perceptrons when a small number of hidden nodes are used. The left graph pictures the success-rates for the Maxout networks with no implementation of the winning bias for the linear nodes (see 2.2.1), while the right graph shows Maxout networks with a tuned winning bias of 0.35.

3.2 Simple two-dimensional classification problem

This simple problem was easily solved by a multilayer perceptron with only two hidden nodes, with a success-rate above 99%. The Maxout network with only one maxout unit and two linear nodes was also able to solve the problem, although the chance of success was dropped to about 64%. This problem was partly included for the purpose of creating a visual interpretation of how the Maxout network was being trained, and a picture of the decision lines in the input space created by the two linear nodes were created for each update. The tuning of the maxout activation function could then be studied, see figure 12.

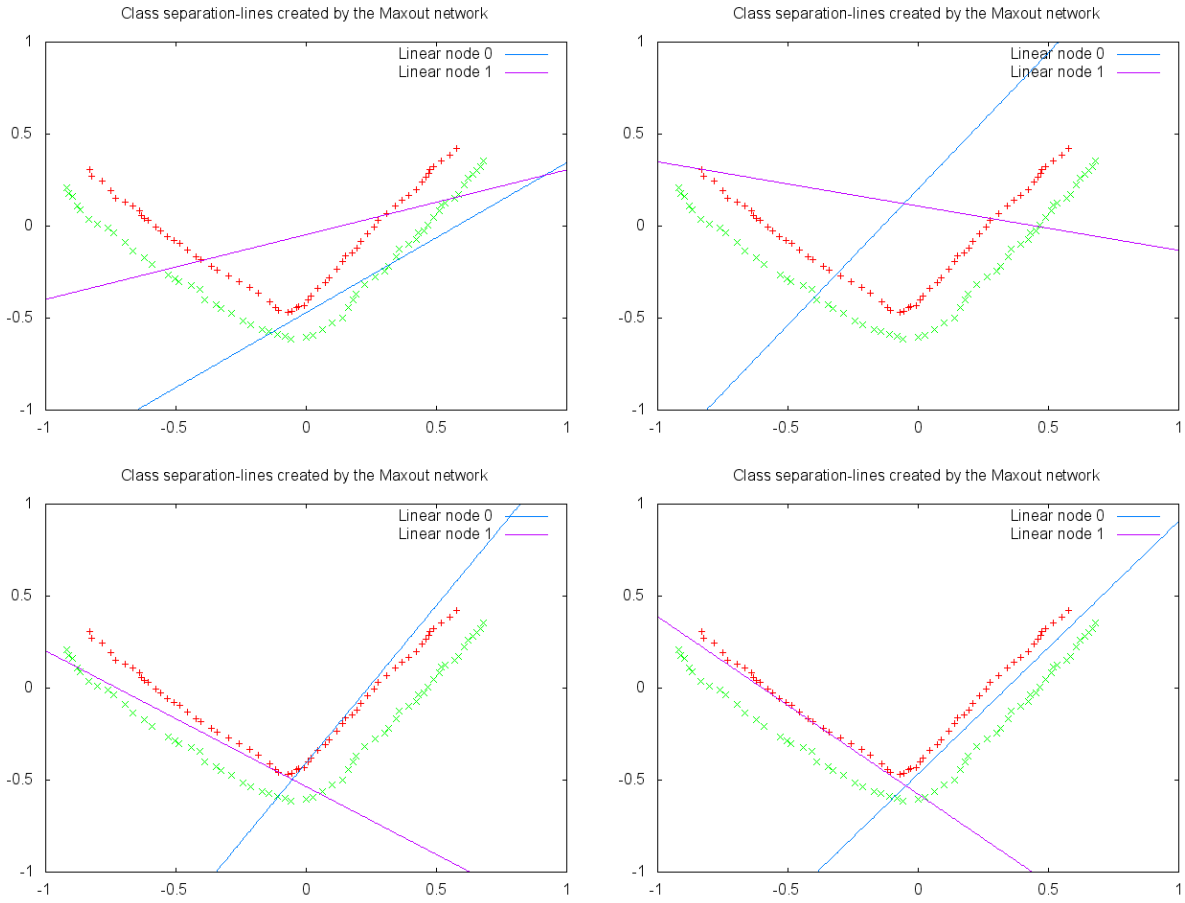

Figure 12: A Maxout network finding where to separate the input space in order to divide between the two classes. The dataset used is the simple two-dimensional classification problem and the pictures show the separation lines created by each linear output at different times during training. In the fourth picture the network has classified all the datapoints correctly, leaving one class above the lines and the other beneath one or both of them.

3.3 Artificial non-linear classification problem

The artificially created dataset was tested using many different structures of networks. As expected a linear boundary was not enough for satisfactory results, and every extra hidden node up to some degree would increase the performance of the network. The Maxout network only required four hidden nodes to find its top result while the multilayer perceptron needed at least six hidden nodes before the performance-increase became negligible. A maxnorm-regularization value of 2.5 was used in order to prevent the networks from overtraining, see figure 13.

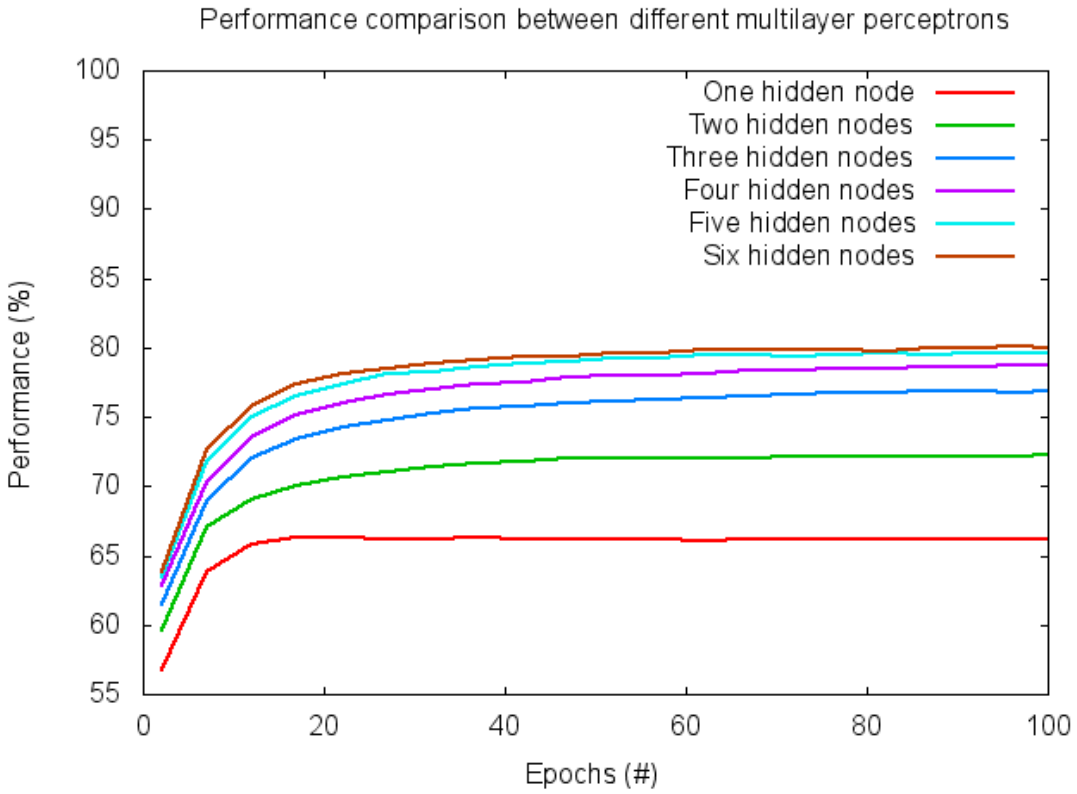

Figure 13: The performance of the multilayer perceptron on the Artificial non-linear classification problem. The performance is increased up to a certain value with every added hidden node to the structure of the network. The training curves are shown to view a proper convergence.

When implementing the Maxout method the overall performance of the networks increased, and the computer run-time needed to train the networks diminished significantly. For networks using three hidden nodes, a Maxout network with four linear nodes only took two thirds the time to implement of that of the multilayer perceptron, and it averaged above a 83 % correct classification, compared to the 80 % which was that of the multilayer perceptron, see figure 14.

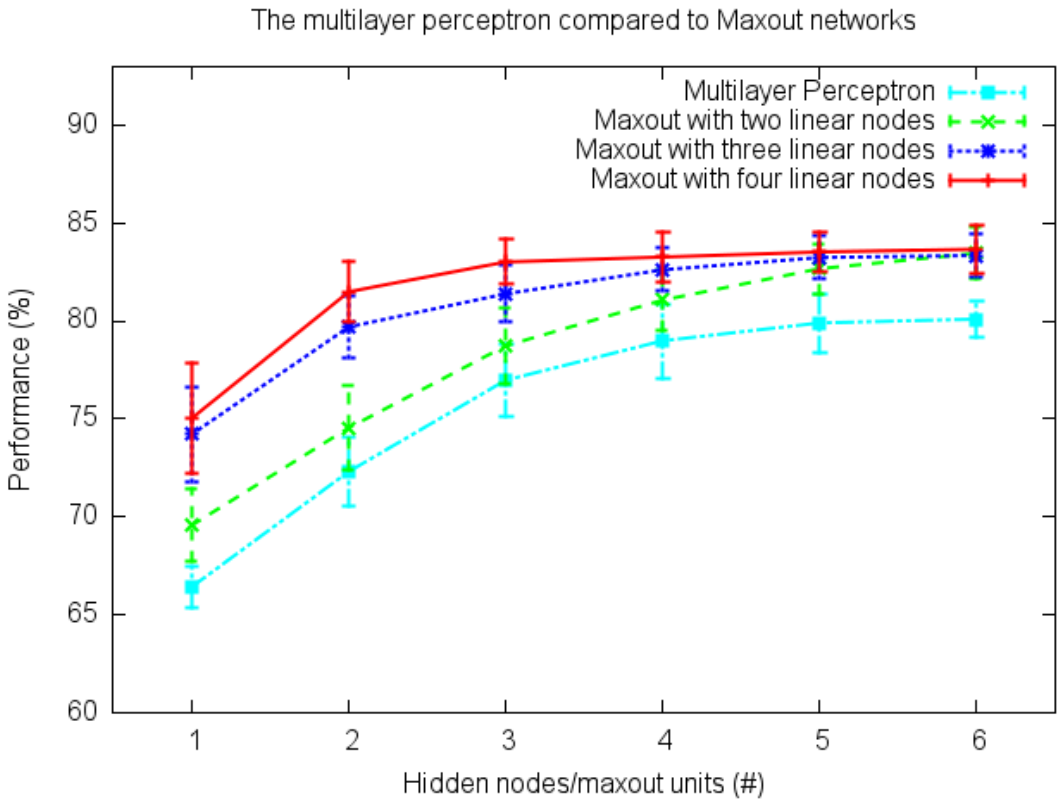

Figure 14: The average performance of different networks on the artificial non-linear classification dataset. The error-bars shows the standard deviation of the 100 networks that were used to create the average performance. The multilayer perceptron performs poorly compared to the Maxout network, it is even outperformed by the network using only two maxout nodes, and the MLP took more than twice as long to train.

A network using three hidden nodes becomes faster to train with an increased performance when the Maxout method is implemented, however the complexity of the network also increases since the number of trainable weight-parameters increases with every linear node. It is therefore interesting to compare the Maxout network with a multilayer perceptron with the same total amount of weights. Since every linear node is connected to every input node a Maxout network with three maxout units, each with four linear nodes, will have the same amount of weight parameters as a multilayer perceptron with one hidden layer containing twelve hidden nodes.

However when compared to the large multilayer perceptron the small Maxout network is still the better classifier, see figure 15. The MLP was outperformed by about two percent units and the computer time needed to train it per epoch was almost six times higher than that of the Maxout network. Increasing the size of the multilayer perceptron any further would also not increase its performance significantly, meaning it would never reach the potential of the Maxout network.

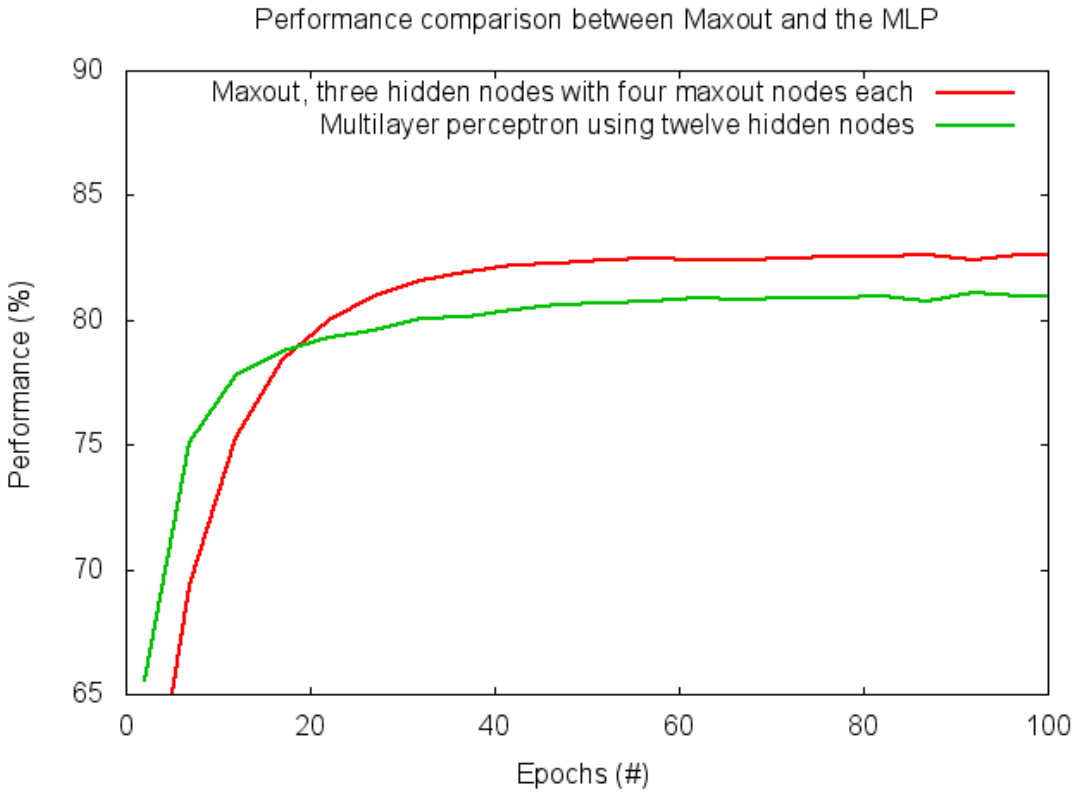

Figure 15: Comparison between a small Maxout network and a large multilayer perceptron, trained on the artificial non-linear classification problem. Despite the large amount of hidden nodes in the multilayer perceptron the Maxout network still outperforms it, both in performance and in computer-time used to train.

3.4 Altered non-linear classification problem

When adding four extra random inputs to the artificial dataset to make the problem more difficult the Maxout method is still the better classifier, see figure 16. The difference in performance between the multilayer perceptron and the Maxout networks were even greater when switching to the more complex dataset, making the advantages of the Maxout method clearer. When an equal amount of hidden nodes were used the MLP took 60% longer time to train compared to the Maxout network, even though the same amount of updates were needed.

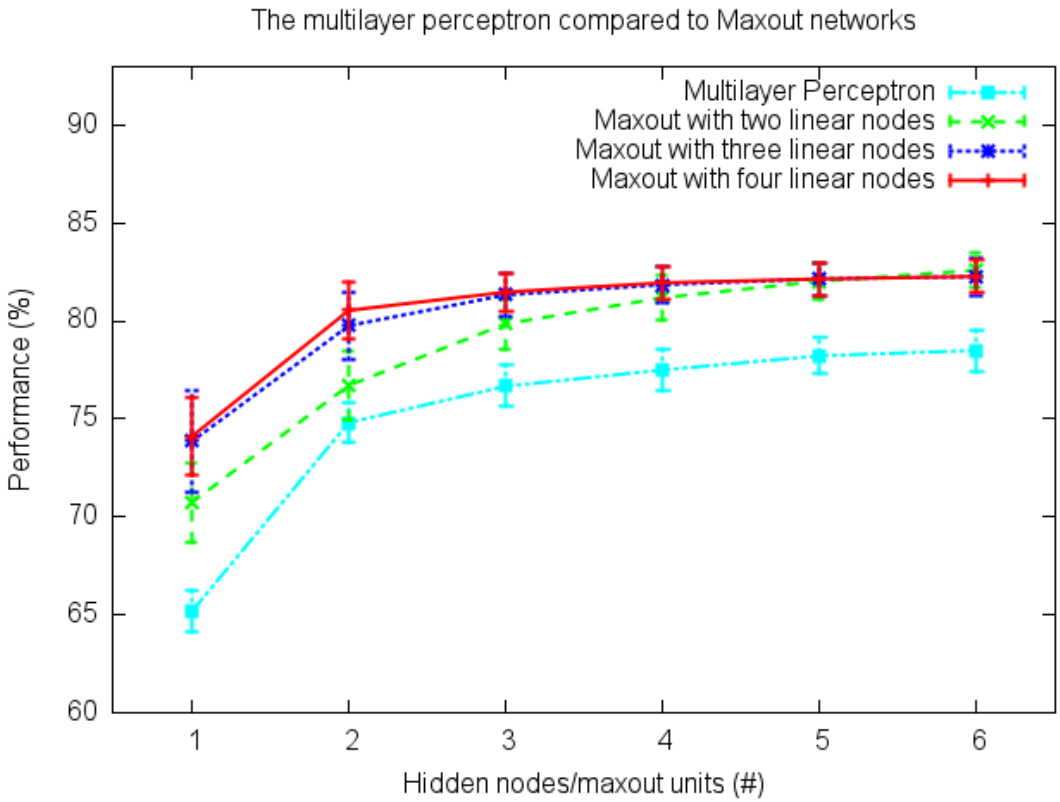

Figure 16: The performance of different networks trained on the altered artificial non-linear classification problem. The error-bars shows the standard deviation of the 100 networks that were used to create the average performance. The Maxout networks clearly outperforms the multilayer perceptron when the same amount of hidden nodes are used.

The multilayer perceptron was not able to reach as high performance as the small Maxout network regardless of number of hidden nodes used, and it took four times as long to train a multilayer perceptron with the same amount of weight parameters as the Maxout network, see figure 17.

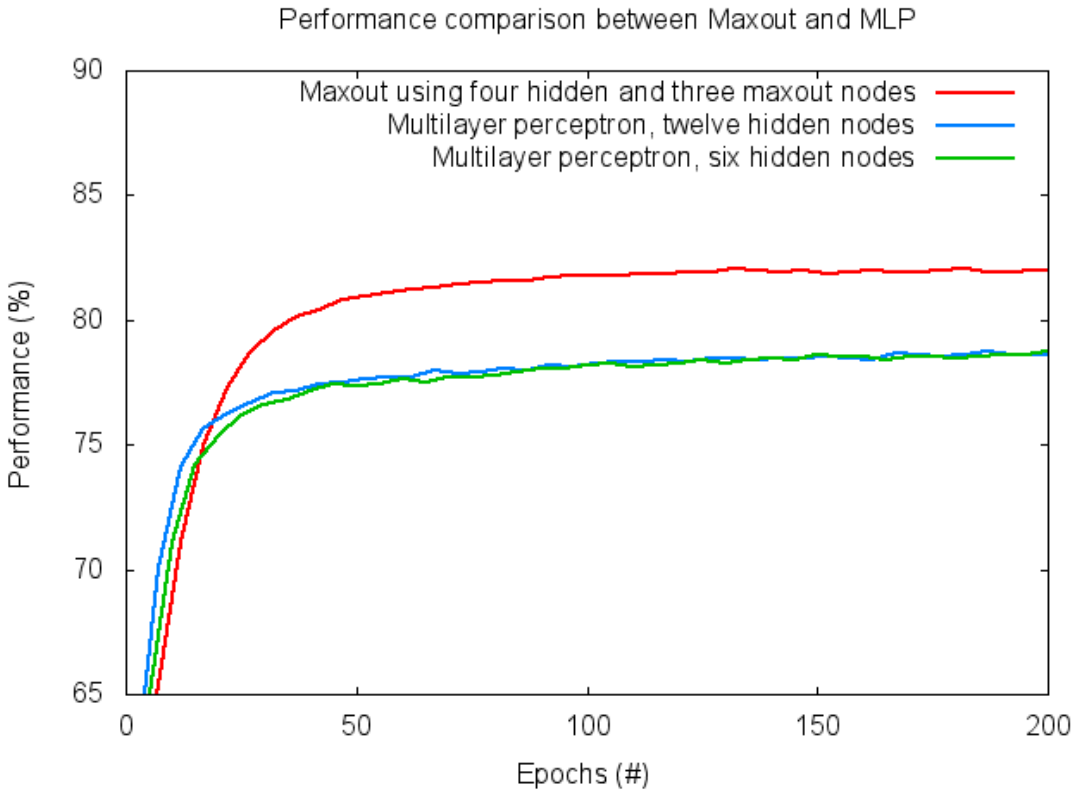

Figure 17: The performance of a small Maxout network compared to larger multilayer perceptrons using the altered artificial non-linear classification problem. The performance of the multilayer perceptrons does not increase beyond six hidden nodes, and it is clearly outperformed by the Maxout network. It is also visible that due to the max-norm regularization method the networks are not being overtrained.

3.5 Function Approximation

When using an artificial network to find an underlying function in the formula for creating cement the Maxout network was outperformed by the multilayer perceptron. The normalized mean squared error was compared between the networks and the multilayer perceptron had a smaller error than the Maxout method regardless of the number of hidden nodes used. The standard deviation of the different models were also smaller, making it a more reliable network, see figure 18. A further increase of the amount of linear nodes in the Maxout network did not have any major impact on the performance of the network. It is however worth mentioning that the multilayer perceptron took twice as long computer-time to train as the Maxout networks.

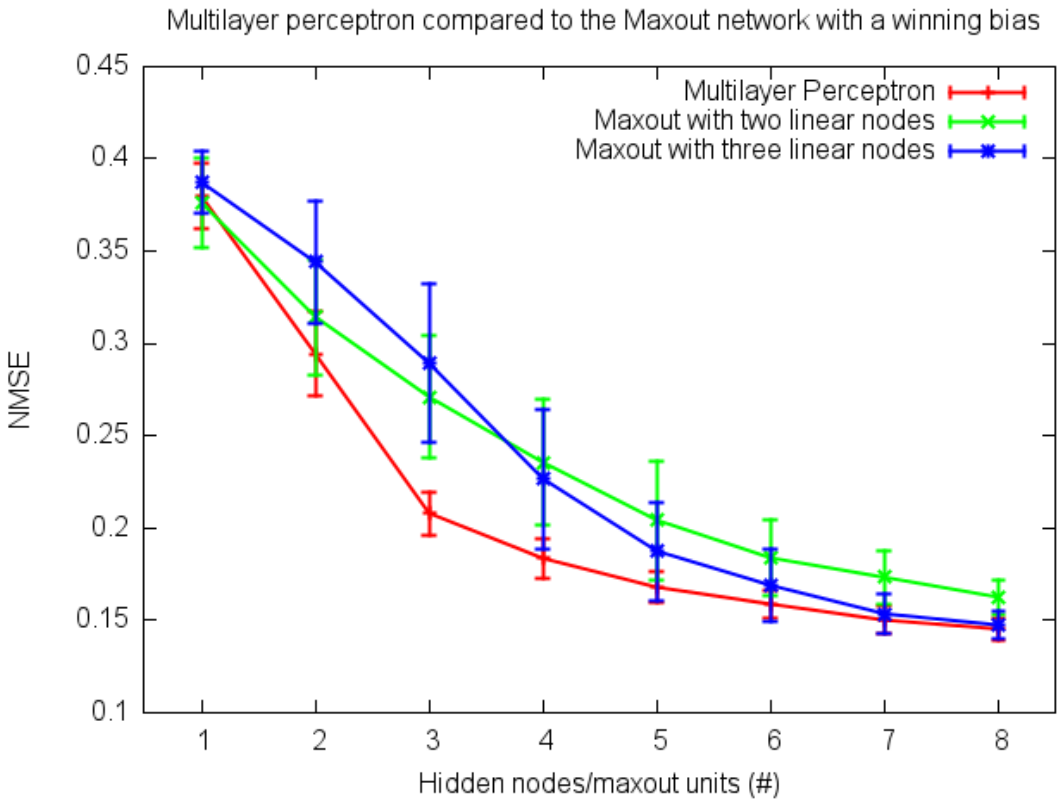

Figure 18: The normalized mean squared error for the multilayer perceptron and the Maxout network on the concrete dataset. The error-bars shows the standard deviation of the 100 networks that were used to create the average performance. The MLP is finding a better solution than the Maxout networks when few hidden nodes are used, but for larger networks they have similar performance. The Maxout networks however take much shorter time to train than the MLP.

4 Conclusion and Discussion

The Maxout network is a genuine alternative to the multilayer perceptron when using shallow networks. For multi-dimensional classification problems it is able to reach results not obtainable for the multilayer perceptron. It is also much faster to train due to the piecewise linear activation function used, which saves the network update from time-consuming computations of derivatives.

4.0.1 Classification

When solving the XOR problem using a shallow network the Maxout method showed a clear increase in the success-rate of the network. The implemented winning bias, hindering "dead" nodes, increased the success-rate even further, making even the smallest possible

Maxout network able to solve XOR problem almost every time. When increasing the number of hidden nodes above three the multilayer perceptrons success-rate would also approach 100%, however it took a longer time to train than the Maxout Network.

As the complexity of the problems increased the Maxout method showed to be an obvious improvement, both in reaching an overall higher classification-performance and also in greatly decreasing the computation-time needed for training.

4.0.2 Regression

The regression problem used, the concrete dataset, proved to be a highly complex problem, creating a need for a larger network in order to find the best results.

Using small networks the sigmoidal activation function of the multilayer perceptron resulted in a lower normalized mean squared error than that of the small Maxout networks. The differences between the two models however diminished as the size of the models were increased, making the Maxout method useful for regression problems as well.

The time-difference in training the multilayer perceptron compared to the Maxout network increases with the size of the networks, making the advantages of the Maxout method larger for deeper networks. It also seems to be correlated to the complexity of the dataset used, giving the Maxout method an advantage when trying to minimize the mean summed squared error for the concrete dataset.

4.0.3 Outlook

The science of artificial neural networks is still in its infancy, and it is difficult to tell where the journey will take us. We have begun the creation of artificial brains that are capable of analyzing, memorizing and acknowledging its surroundings and learning from it. Yet there is still so much more to discover and so much more to learn. What will happen as these artificial brains grow more and more complex, eventually maybe even past or own brains? Our dreams to understand what consciousness is, to understand what we are, such questions may one day be answered. I think that we have finally found the path that just might take us there.

5 References

References

[1] Mattias Ohlsson, “Lecture Notes on Artificial Neural Networks (FYTN06)” Computational Biology and Biological Physics Department of Astronomy and Theoretical Physics Lund University, 2015

[2] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio, “Maxout Networks” Département d’Informatique et de Recherche Opérationelle,

Université de Montréal 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8; 2013

[3] Simon Haykin. Neural Networks: a comprehensive foundation. 2. uppl. Upper Saddle River, New Jersey: 1999

[4] Andrew R. Barron. Universal Approximation Bounds for Superpositions of a Sigmoidal Function. IEEE Transactions on information theory; 1993. Available from:

http://www.stat.yale.edu/ arb4/publications_files /UniversalApproximationBoundsForSuperpositionsOfASigmoidalFunction.pdf

[5] Yann LeCun, Yoshua Bengio, Geoffrey Hinton. Deep learning. Nature; 2015. Available from:

http://www.nature.com/nature/journal/v521/n7553/full/nature14539.html

[6] Tom Fawcett. An introduction to ROC analysis. Volume 27, Issue 8, June 2006, Pages 861–874. Available from:

http://www.sciencedirect.com/science/article/pii/S016786550500303X

[7] Nitish Srivastava, Geoffrey Hinton, lex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 2014.

https://www.cs.toronto.edu/ hinton/absps/JMLRdropout.pdf

[8] Yeh, I-Cheng Cheng. Modeling of strength of high-performance concrete using artificial neural networks. Volume 28, Issue 12, December 1998, Pages 1797–1808. Available from:

http://www.sciencedirect.com/science/article/pii/S0008884698001653