Neural networks

Enric Gil Gallen, Yeray Boix Torner

*Dep. de Ingeniería y Ciencia de los Computadores*

*Universidad Jaume I, Spain  
Avinguda de Vicent Sos Baynat, s/n, 12071 Castelló de la Plana, Castelló, España*

al387320@uji.es

al376154@uji.es

***Abstract*— This document deals with the topics covered in the first two laboratory sessions "Perceptrons" and "Multilayer Perceptrons". The data obtained will be shown and analyzed and the knowledge necessary to carry out the practices will be deepened.**

1. Introduction

*“Machine learning is the study of computer algorithms that can improve automatically through experience and by the use of data”* **[1]**. These algorithms build models based on sample data in order to make predictions or decisions without being explicitly programmed to do it **[2]**.

Machine learning involves Artificial Intelligence , this term is referred to the intelligence provided by machines and nowadays it is everywhere. We can see Artificial Intelligence in topics like:

1. Virtual Personal Assistants
2. Predictions while Commuting
3. Videos Surveillance
4. Social Media Services
5. Search Engine Result Refining
6. Medicine

and many more applications. That means that Machine Learning with Artificial Intelligence will lead the way of the future helping human beings**[3]**.

In this report we are going to focus on Neural Networks specifically in Perceptrons but first we are going to take a look at what Neural Network means.

“*A neural network (NN) is a computing system loosely* *inspired by the structure of the human brain and it provides a framework for multiple Machine learning algorithms to work together to process complex data. A neural network can “learn” to perform tasks by analyzing examples, usually without task-specific instructions.”* **[4]**.

They have been used to process things like recognize handwriting for check processing, recognize traffic signs, resolve XOR, AND and OR problems, etc. We are going to explain these applications in detail in the different sections below.

1. Neuron Structure

Lets see the difference between a Biological Neuron and an Artificial Neuron:

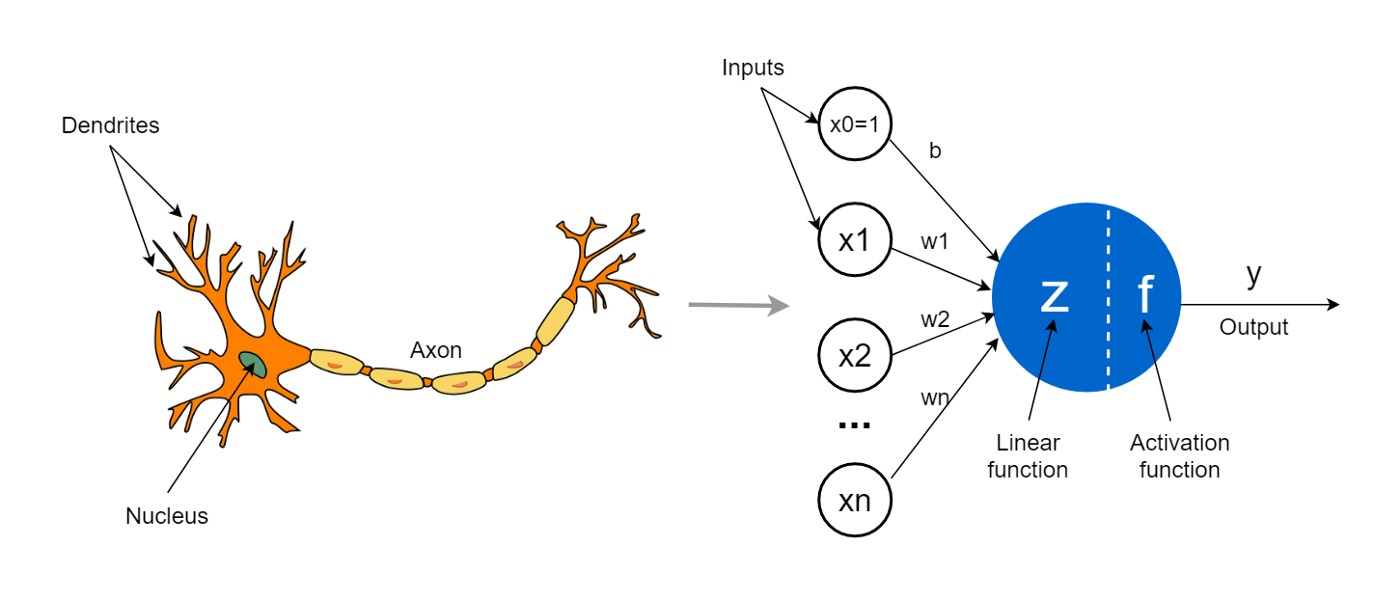


Fig. 1 Both Biological (lef) and Artificial Neuron (right) structures.

1. *Biological Neuron*

The neuron structure contains a data input (*dendrites)*, a computing organ ( *nucleus*) and an output channel (*axon*) (Fig. 1) **[5]**.

1. *Artificial Neuron*

We can see in the picture (Fig. 1) that the *Dendrites* correspond with the *Inputs* of the Artificial Neuron. The *Linear function* and the *Activation Function* correspond with the *Nucleus* and the *Output* with the *Axon.*

* **Inputs**: The input and output data can be binary or continuous.
* **Weights**: Interaction intensity of each neuron.
* **Propagation rules**: Potential value of the neuron.
* **Activation Function**: Gives actual activation state of the neuron depending on the previous state.
* **Output Function**: Actual output of the neuron depending on the actual activation state

1. Neural network types

We had a look on how Artificial Neurons are, now we are going to make networks combining neurons.

1. *Single layer Neural Network*

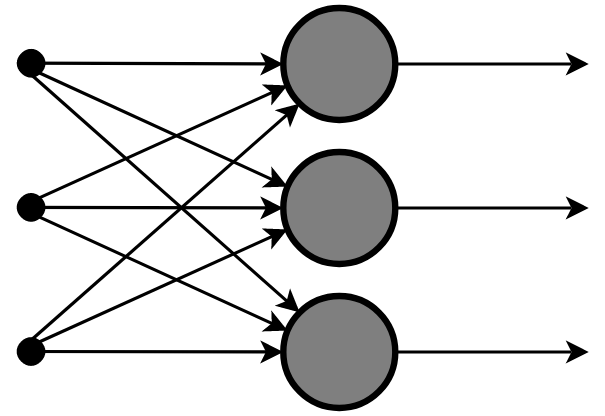


Fig. 2 Single layer Network

This is the simplest network (Fig. 3), we can see that we only have one layer of neurons (*grey circles*) where all of them receive input (*black circles*) data and make different calculations.

“*The system performs a mapping from the n-dimensional input space to the m-dimensional output space”* **[6]** and we can use the same algorithms used on one neuron for all of them.

This layer type is used to solve linear separable problems but it is not able to resolve non linear separable data sets **[6]**.

1. *Multilayer Neural Network*

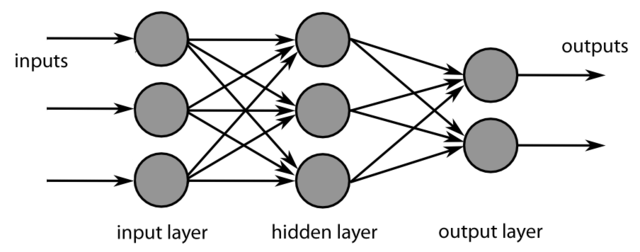


Fig. 3 Single layer Network

Here we can see the difference between both single (Fig 2.) and multilayer (Fig 3.).

“*Typically, they have at least one input layer, which sends weighted inputs to a series of hidden layers, and an output layer at the end*” **[7]**.

In the beginning of the AI evolution *Single layer networks* were the way to go but nowadays the majority of networks have a *Multilayer* model.

They are used to solve non linear classification problems employing hidden layers (Fig 4.).

We are going to have a look at some examples in detail for both single and multilayer networks below. But first let's talk about Perceptrons.

1. Perceptron

“*The perceptron is an algorithm for supervised learning of binary classifiers (functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not). It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time”.*

“*The perceptron algorithm dates back to the late 1950s, its first implementation, in custom hardware, was one of the first artificial neural networks to be produced”* **[8]**.

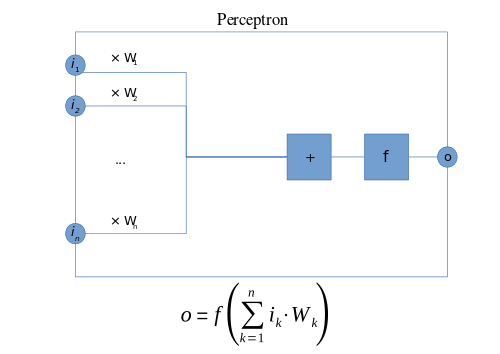


Fig. 4 Perceptron structure

The steps Perceptrons follow to learn are similar to how Neural Networks learn, which is as follows:

* Initialize weight values and bias
* Forward Propagate
* Check the error
* Backpropagate and Adjust weights and bias
* Repeat for all training examples

1. Perceptron ALGORITHM

The Perceptron algorithm states that:

**if Wx+b > 0 :**

y' = 1

**else:**

y' = 0

Knowing the learn steps and the algorithm we can see how Perceptrons solve problems like AND, OR gates and linearly separable classifications.

1. AND PROBLEM WITH A PERCEPTRON

First of all let's see how an AND logic table looks (Fig 7.) **[9]**

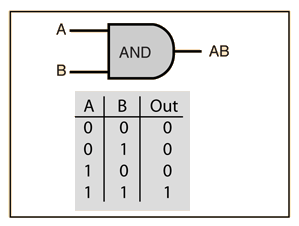


Fig. 5 AND gate

As we can see we have two inputs (A and B) and one output (AB) so we need to adapt the Perceptron algorithm to our case.

First of all we need to define the weight wich in this case we have two inputs, so our weight its composed by w1 and w2:

so we need to redefine the Perceptron Function as:

where b is the **bias**.

“*The bias shifts the decision boundary away from the origin and does not depend on any input value”* [8].

For the AND function we can define both **weights** as 1 and **bias** of -1 to start with.

1. DATA

We need to define the Target which represents the output we want, in this case [0, 0, 0, 1] and the input data as a matrix representing both A and B like [[0, 0], [0, 1], [1, 0], [1, 1]].

We can plot the data to see what we are doing, Zeros represented by red dots and ones with Black dots (Fig. 6).

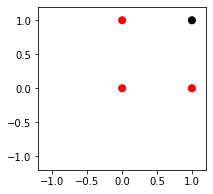


Fig. 6 AND plot

With that in mind we can start training the model until it converges.

1. *Train*

Now with all the data and functions defined we can start training the model until it converges.

In each iteration it will return an output data like [1, 0, 0, 1] which in this case it's wrong. So it is going to compare the output returned by the function with the target output defined before which is [0, 0, 0, 1].

Every time the model gets wrong it is going to adjust both **bias** and **weights** and repeat all the process until it converges.

When the model converges the perceptron boundary completely separates the samples of each class (0’s and 1’s) and it will look like this (Fig. 7):

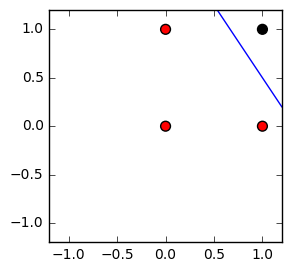


Fig. 7 AND plot converged

1. OR PROBLEM WITH A PERCEPTRON

Here we have an OR gate (Fig. 8)

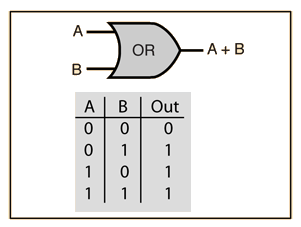


Fig. 8 OR gate

We can repeat the same process adjusting the input and output data with the OR values.

We can plot the OR gate and it will look like this:

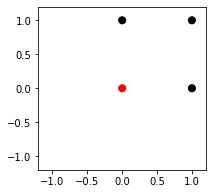


Fig. 9 OR plot

Red dots for 0’s and Black dots for 1’s.

So if we train the model it will converge and will look like this:

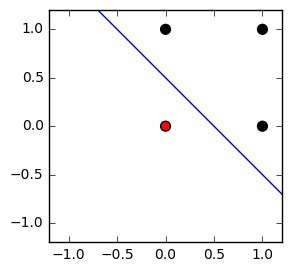


Fig. 10 OR plot converged

So we can see that Perceptrons can solve OR and AND gates easily because they are linearly separable problems.

But what happens with the XOR problem? Let's have a look.

1. XOR PROBLEM WITH A PERCEPTRON

First of all let's see how a XOR gate looks (Fig. 11):

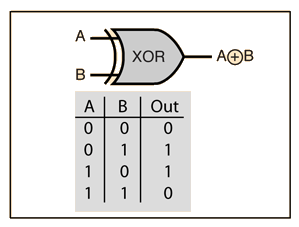


Fig. 11 XOR gate

With a naked eye we could say that as the OR and AND problems we have the same inputs and outputs so what is really going on here? Let's have a look plotting the values:

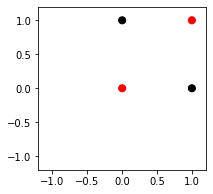


Fig. 12 XOR plot

We can see now that unlike OR and AND problems the XOR problem is not linearly separable so a single Perceptron can not solve this problem.

1. LINEARLY SEPARABLE CLASSIFICATION WITH SINGLE PERCEPTRON

Lets see another sample of how a single Perceptron can work well for linearly separable problems.

1. *Problem*

Let’s imagine that there are two classes of dots (red and black). Each dot is defined by two features. The dataset consists of a matrix X with as many rows as dots, and two columns, and the vector Y with as many elements as dots. The value of y[i] is 0 for red and 1 for black dots.

So with that in mind we build a dataset with at least 5 or 9 red dots and a total of 14 dots and define a Target to aim the model to converge.

It could be a model like this:

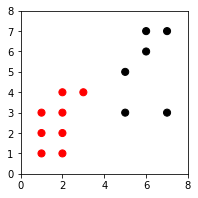


Fig. 12 Data plot

We can see that the model is linearly separable, so let's train a model with a single Perceptron until it converges.

As the AND and OR problems it will adjust the weights and de bias everytime it fails on his prediction until it converges. It will look something like this:

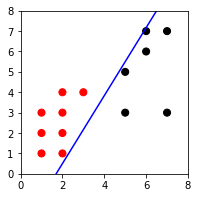


Fig. 13 Data plot converged

So once more we can see a single Perceptron working on linearly separable classes.

1. INTRODUCTION MULTIPLAYER PERCEPTRON
2. *Introduction*

The multilayer perceptron (MLP), unlike most common perceptrons, classifies data that are not linearly separable. This is because they use a more robust and difficult architecture to learn regression and classification models for difficult data sets. The multilayer perceptron (MPL) has at least three layers of nodes. 1st input layer, 2nd hidden layer, 3rd output layer. The second and third node layer is a neuron using a nonlinear activation function.

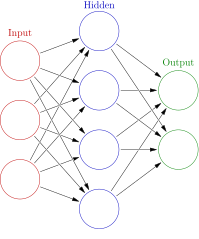


Fig. 14 Example of an image with Multiplayer receptor

Multilayer perceptron may have other names:

*“ The term MLP is used ambiguously, sometimes loosely to mean any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptrons (with threshold activation); see § Terminolog*[*y*](https://en.wikipedia.org/wiki/Multilayer_perceptron#Terminology)*. Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.”* **[10]**.

1. TRAINING MULTILAYER NEURAL NETWORK
2. *Forward propagation*

MPLs have an arbitrary number of hidden layers located between the input layers as well as between the output layers.

To begin with, weights are assigned to each input layer depending on its characteristics flowing down to the hidden layers.

The weights are then joined with the requirements of the inputs (these requirements can be anything such as arithmetic operations: addition, subtraction, multiplication, etc.) to pass to the activation function, the most commonly used of which are sigmoid functions. The result of the hidden layer becomes the input of the next hidden layer.

This process will be repeated as many times as we have programmed, so far we have only moved in the forward direction.

This is known as forward propagation.

This system has a problem that error in one neuron is due to the weights of its previous neurons. To minimize the error you have to adjust the learning rate.

Learning rate: It is the percentage of change with which the weights are updated in each iteration, in other words, each time an iteration is performed in the training process the input weights must be updated in order to give a better approximation each time. It should be noted that:

If small you have to iterate many epochs.

If it is very large it may not converge **[11]**

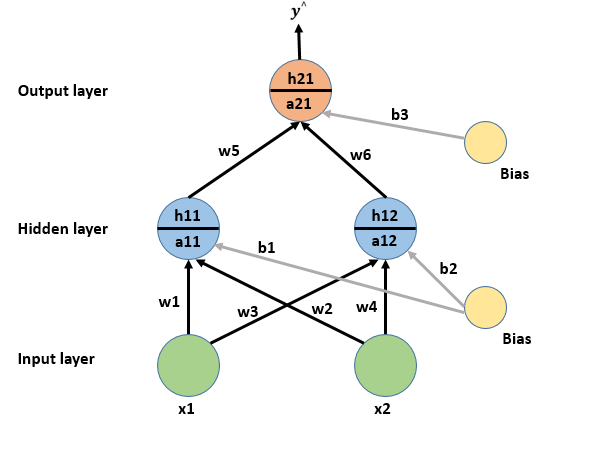


Fig 15. Example of Forward propagation.

### 

1. *Backward propagation*

Now, it is time to move on to backpropagation (back propagation) in order to minimise the value of the loss.

The weights need to be adjusted to be close to the predicted output. Then, during training the weights will be updated so that the difference is decreasing. One way to do this is to use gradient decreases.

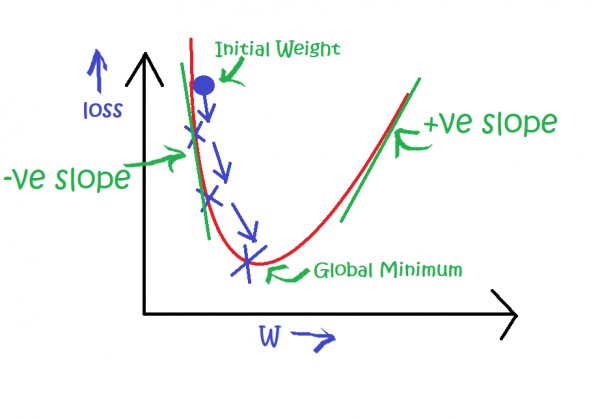


Fig 16: Example of the use of Gradient Descent

1. APPLICATIONS OF MULTILAYER PERCEPTRONS

Some examples of Applications of Multi-Layer Perceptrons **[13]**

*1. Types of Feed-Forward Neural Network Applications*

*2. Brain Modelling*

* *Development, Adult Performance,*
* *Neuropsychology Analysis of Hidden Unit Representations*

*3. Real World Applications*

* *Data Compression - PCA*
* *Time Series Prediction*
* *Character Recognition and What-Where*
* *Autonomous Driving - ALVINN*

1. REAL WORK APPLICATION [13]

*1. Airline Marketing Tactician (Beale & Jackson, Sect. 4.13.2)*

*2. Backgammon (Hertz et al., Sect. 6.3)*

*3. Data Compression – PCA (Hertz et al., Sect. 6.3; Bishop, Sect. 8.6) •*

*4. Driving – ALVINN (Hertz et al., Sect. 6.3) •*

*5. ECG Noise Filtering (Beale & Jackson, Sect. 4.13.3)*

*6. Financial Prediction (Beale & Jackson, Sect. 4.13.3; Gurney, Sect. 6.11.2) •*

*7. Hand-written Character Recognition (Hertz et al., Sect. 6.3; Fausett, Sect. 7.4) •*

*8. Pattern Recognition/Computer Vision (Beale & Jackson, Sect. 4.13.5) •*

*9. Protein Secondary Structure (Hertz et al., Sect. 6.3)*

*10. Psychiatric Patient Length of Stay (Gurney, Sect. 6.11.1)*

*11. Sonar Target Recognition (Hertz et al., Sect. 6.3)*

*12. Speech Recognition (Hertz et al., Sect. 6.3)*

*13. Text to Phoneme Mapping (Beale & Jackson, Sect. 4.13.1; Bullinaria, 2011) •*

1. XOR PROBLEM WITH MULTIPLAYER PERCEPTRON

We had a look at how the XOR problem is and we saw a plot of the data, but we could resolve the problem using a single Perceptron. Let's see how a Multilayer Perceptron can outperform a Single Layer Perceptron and resolve the XOR problem..

Building the model

When we set the data we need to build the model in that case we are going to use a stochastic gradient descent solver (standard technique in backpropagation), one hidden layer with 5 neurons and we are going to set the maximum iterations until converge to 4000 maximum.

Train

Then we are going to train the network until it converges. Lets see how the result looks.

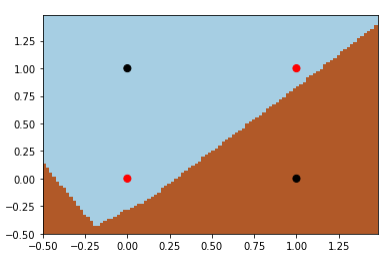


Fig 17 non-converged model

It seems that our MLP failed the percentage of correct classification of the training data. It is 75.0 and it did 1039 iterations during the training. The convergence stops because the loss or score were not improving by at least tol (default 1e-4) for two consecutive iterations and when it happens the convergence is considered to be reached and training stopps. Lets see the curve in this plot:

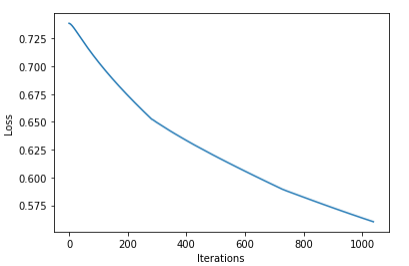


Fig 18 loss curve non-converged model

We can see that the Loss value is not changing at all so the model stopps the training.

Lets see the difference with a model that reaches the 100.0 score:

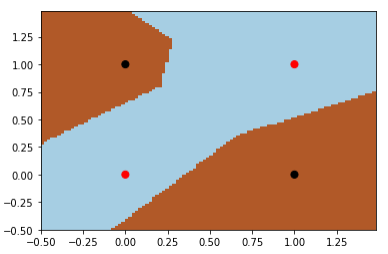


Fig 19 converged model

We can see the model converges very well and separates the non-linear classification problem. In that case it takes 2930 iterations to solve the problem as we can see in this plot:

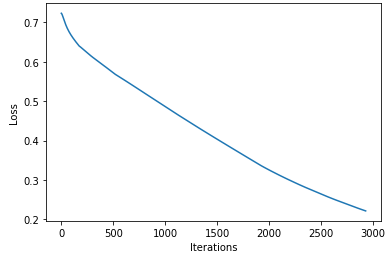


Fig 20 loss curve converged model

We can see the line of the plot is not different from the non-converged model and it continues because the iterations until the model reaches the score of 100.0 points. Does not stop because the Loss value because it was changing with a difference bigger than 1e-4.

The model could converge in different solutions such as:

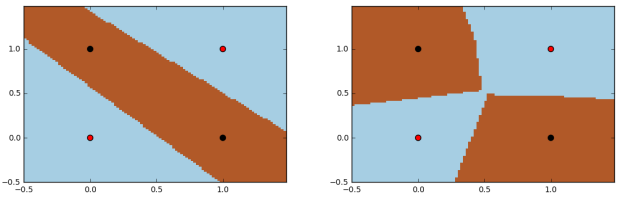


Fig 20 converged samples

With different iterations and different loss curves but the same score (100.0).

1. CLASSIFICATION EXAMPLE WITH MPL

In these kinds of classification problems we first need to preprocess the input data, that means that the data rarely comes in the form and shape that is necessary for the optimal performance of an algorithm. The first steps are the preprocessing tasks: loading the data, scaling it and splitting the data into training and testing sets.

In our case the predictive model is the MLP and its parameters need to be set before starting to train. We are going to use “lbfgs” solver and one hidden layer with 5 neurons with a maximum of 4000 iterations.

After we select the model that has been fitted on the training data set, we can use the test data set to estimate how well it performs on this unseen data to estimate the generalization error.

1. *Evaluating network performance*

Learning the parameters and testing it on the same data is a methodological mistake, a model that would just repeat the labels of the sample that it has just seen would have a perfect score but would fail to predict anything useful on unseen data.

So we need to store the data set we use as a test set like X\_test and Y\_test to train with them first.

1. *Data scaling*

We need standardization of the datasets because it could behave badly if the individual features do not look like standard distributed data.

With all of this clear we can work with our training data set:

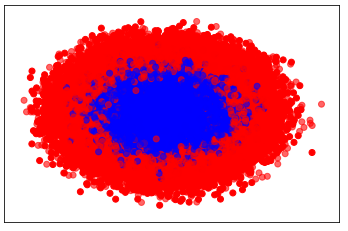


Fig 21 test sample

and see the result of training with this test data:

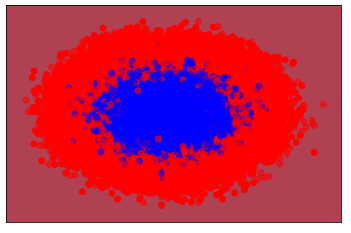


Fig 22 test sample solved

The percentage of correct classification of the test data is 88.75 which is very good.

1. USING THE TRAINED MODEL WITH A REAL PROBLEM

Now we can use our data set to solve classification problems such as the previous

1. RECOGNIZE ANY NUMBER

Another use of Multilayer Feedforward Networks is to recognize any number in an image.

To solve this problem we will need a database with different types of numbers as varied as possible (the more varied in shape and size the numbers are, the better it will be able to recognize different formats). The numbers will be obtained from images with the same resolution 8 X 8 pixels.



Fig 23: Analysis images

Next we train the network and test it with different parameters.

We can make different reports of the results to see how effective they are

1. *Classification report*

It will measure:

* **precision**: The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
* **recall**: The recall is intuitively the ability of the classifier to find all the positive samples.
* **f1-score**: a weighted average of the precision and recall, where an 𝐹1F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the 𝐹1
* fupport: The support is the number of occurrences of each class

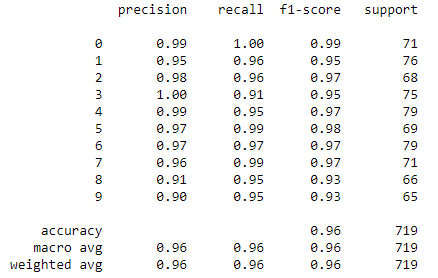


Fig 24: Classification report

1. *Confusion matrix*

By definition, a confusion matrix 𝐶 is such that 𝐶𝑖,𝑗 is equal to the number of observations known to be in group 𝑖 but predicted to be in group 𝑗 . Thus an optimal classification result would produce a diagonal matrix.

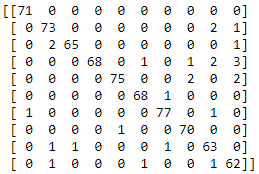


Fig 25: Confusion matrix

1. *Loss curve*

Number of iteration: 694

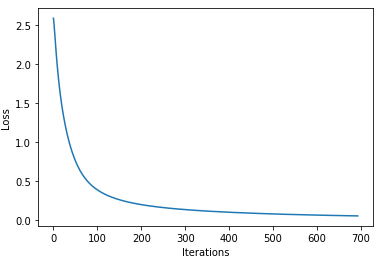
****

Fig 26: Loss curve

We can observe that from iteration 600 the variation of the loss is so small that the model converges.

1. *Samples of predictions*

Finally, the target and prediction can be displayed for some images of the data set.

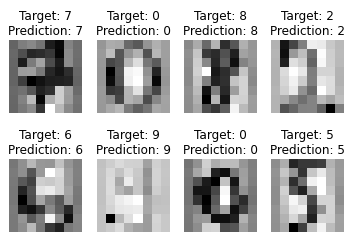


Fig 27: Predictions

1. CLASSIFICATION WITH THE GERMAN TRAFFIC SIGN RECOGNITION BENCHMARK

Another use of Multilayer Feedforward Networks is Classification with the German Traffic Sign Recognition Benchmark.

An automatic road sign recognition system first locates road signs within images captured by an imaging sensor on-board of a vehicle, and then identifies road signs assisting the driver to properly operate the vehicle.

Automated road sign recognition is a difficult task.

The first thing to do is to standardise the data to be used for training.

In this case we have a large database in which the different types of signals are separated and each type has many different shapes.

By training the network with so much variety, it will allow the network to recognize new signals or signal variations more efficiently when it is already trained.

1. *Database*

* Single-image, multi-class classification problem
* More than 40 classes
* More than 50,000 images in total
* Large, lifelike database

The training set archive is structured as follows:

* One directory per class
* Each directory contains one CSV file with annotations ("GT-.csv") and the training images
* Training images are grouped by tracks
* Each track contains 30 images of one single physical traffic sign

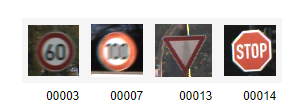


Fig 28

To improve the training of the network, it is sometimes convenient to treat the data. In this case, since the unmatched signals are in the same position in the images, we will crop the images (all with the same format) so that the network focuses on what is important and so that it is not distracted by the colours and focuses only on the shapes, we will convert them to black and white.

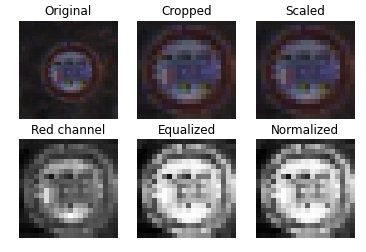


Fig 29

1. *Loss curve*

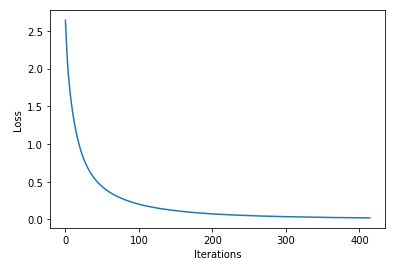
****

Fig 30

1. *Result of a different set of tests than the trained one*

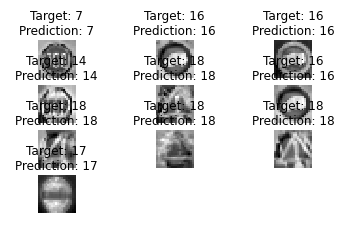
****

Fig 31

REFERENCES

[1] https://en.wikipedia.org/wiki/Machine\_learning

[2] Arthur Sammuel

[3]https://medium.com/app-affairs/9-applications-of-machine-learning-from-day-to-day-life112a47a429d0

[4] https://www.arm.com/glossary/artificial-neural-network

[5] http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm#introduccion

[6] https://www.teco.edu/~albrecht/neuro/html/node17.html

[7] https://www.techopedia.com/definition/33268/multi-layer-neural-network

[8] https://en.wikipedia.org/wiki/Perceptron

[9] <https://www.geeksforgeeks.org/implementation-of-perceptron-algorithm-for-or-logic-gatewith-2-bit-bnary-input/>

[10] https://en.wikipedia.org/wiki/Multilayer\_perceptron

[11] https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250

[12] https://www.goeduhub.com/10389/how-to-train-a-multilayer-perceptron

[13] https://www.cs.bham.ac.uk/~jxb/INC/l11.pdf