Neural Networks

Predicting student performance

Intermediate Report

Mestrado Integrado em Engenharia Informática e Computação

Inteligência Artificial

Group E5_1

Gonçalo Lopes
up201303198

Ivo Fernandes
up201303199

April 20, 2016

Contents

1 Objective

The purpose of this project is to train an Artificial Neural Network, using the Back-Propagation algorithm, in order to predict student performance based on attributes for each student, taking into account aspects of the student's private and work life and also attributes like their gender and age.

2 Description

2.1 Specification

2.1.1 Project phases

To complete this project the group will have to:

1. Understand the structure of a neural network
2. Understand the mechanism of the back-propagation algorithm
3. Choose a back-propagation framework and learn it
4. Learn how to train neural networks
5. Build different networks and train them
6. Compare the networks, try different training methods, handle the data-set
7. Construct a final network and evaluate its accuracy

2.1.2 Network Structure

A neural network is comprised of 3 types of layers, each layer containing nodes that represent neurons.

- Input Layer
- Hidden Layers
- Output Layer

Each layer can have a different number of nodes and all of a layer's nodes are connected to all of the next layer's nodes. There is only one input layer and one output layer. There can be any number of hidden layers, each with any number of nodes. To solve most problems, only one hidden layer is required. Usually, the output layer consists of a single node. A neuron consists of 5 main parts:

1. The **input**, through which the neuron receives information
2. The **connection weights**, that determine how much influence each input value has

3. The **combination function**, which is usually a sum of the input values times their respective weights

4. The **activation function**, which calculates the state of the neuron. The function used is usually the sigmoid function $\frac{1}{1+e^{-\alpha}}$ This function is used because it introduces non-linearity in the network, as opposed to using a step function. This function is also continuous and differentiable. Take $\beta = \frac{1}{1+e^{-\alpha}}$, $\frac{\partial\beta}{\partial\alpha} = \beta(1-\beta)$

5. The **output**, which is the result of the transfer function. If we didn't introduce non-linearity by using the sigmoid function, output $s_i \in \{0,1\}$. Since we introduced non-linearity, output $s_i \in [0,1]$.

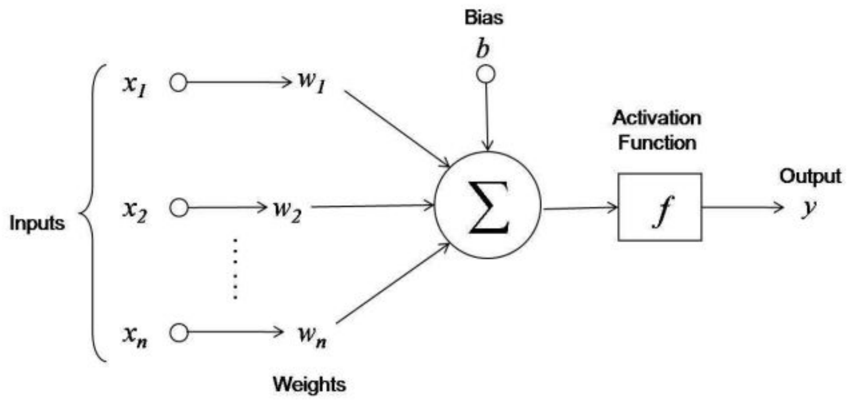

Figure 1: Example of a neuron, taken from `https://blogs.cornell.edu/info2040/2015/09/08/neural-networks-and-machine-learning/`

2.1.3 Back-propagation Algorithm

Let $\overrightarrow{z} = f(\overrightarrow{x}, \overrightarrow{w})$, $\overrightarrow{z}$representing the output of the neural network for the set of inputs $\overrightarrow{x}$ and set of weights $\overrightarrow{w}$ Let $\overrightarrow{d} = g(\overrightarrow{x})$, $\overrightarrow{d}$ representing the desired output of the neural network for the set of inputs $\overrightarrow{x}$ To calculate how well the network is doing we must calculate a function that includes the difference between $\overrightarrow{d}$ and $\overrightarrow{z}$. Let P$(\overrightarrow{d}, \overrightarrow{z}) = \frac{1}{2}||\overrightarrow{d} - \overrightarrow{z}||$, we will call this the Performance function.

We want to minimize the Performance function, minimizing it means minimizing the difference between the desired output and the real output, which means this is the direction to go, to make the network learn.

For explaining the back-propagation algorithm we will always refer to a simple network with 1 input node and 1 hidden node. For a network with 2 nodes, 2 weights, w1 and w2, $\Delta w = r(\frac{\partial P}{\partial w_1}, \frac{\partial P}{\partial w_2})$, r is the **learning rate**. As the name suggests, the learning rate will help the network learn faster, however, this rate cannot be too high. If the learning rate is too high, while trying to fit to the curve of the desired weights, the algorithm will get into a feedback loop, never converging to the minimum and learning nothing.

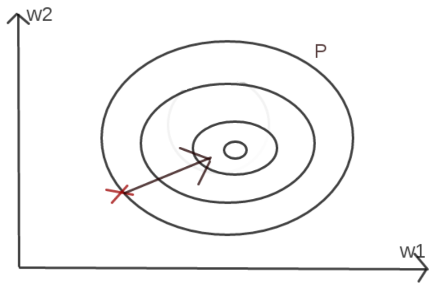

Figure 2: For 2 weights, w1 and w2, we are trying to move in the direction that minimizes the P function

Since we picked the sigmoid function for our activation function, calculating $\Delta\overrightarrow{w}$ becomes simple. Let:

- x be the input of node 1
- y be the output of node 1, input of node 2
- z be the output of node 2, the output of the network

$\frac{\partial P}{\partial w_2} = (d - z)z(1 - z)y$
$\frac{\partial P}{\partial w_1} = (d - z)z(1 - z)xw_2y(1 - y)$
But we already calculated $(d - z)z(1 - z)$.

This means that a given node's weight depends on the node's input, output, the weight of the next node and the partial derivative of the next node. This gives the back-propagation algorithm its name. We propagate the error, $\frac{\partial P}{\partial w_2}$, backwards, in order to calculate $\Delta\overrightarrow{w}$, the vector containing the error for each weight.

By using multiple iterations, the algorithm adjusts the weights of the network, minimizing the performance function.

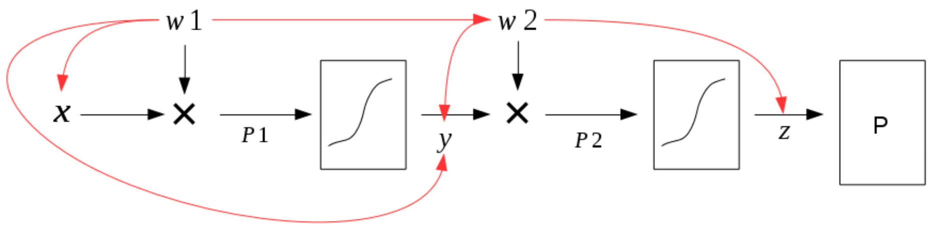

Figure 3: Example network with 1 input node and 1 hidden node, red arrows represent dependence.

The calculation of $\Delta\overrightarrow{w}$ may also be influenced by **momentum**. To incorporate momentum into the calculation, the new formula is:

$\Delta\overrightarrow{w}(t) = r(\frac{\partial P}{\partial w_1}, \frac{\partial P}{\partial w_2}) + m\Delta\overrightarrow{w}(t-1)$, t is the iteration number, m the momentum constant. The momentum is usually set to a high value between 0 and 1.

When we calculate Δw we are going along the gradient of the activation function, in the direction specified by the output error. If by going along this direction, we skip the ideal point for the weights, we must go back. Momentum helps with this situation, if the algorithm was moving in a direction and it suddenly changes directions, the first step will be small, since it takes into account the direction of the last step, which was opposite to the direction of the new step.

A factor that may lead to unsuccessfully training the network is the number of iterations. If the number of iterations is too high overfitting may start to happen. Over fitting is when the algorithm tries so hard to fit to the sample (training set) that values nearby the ones used may become distorted, and when faced with a similar situation the network will output a very different value.

The back-propagation algorithm must also shuffle the dataset instances before starting each iteration, otherwise it will be unable to successfully train. The chosen framework does this.

2.1.4 Training a series of Networks for the same dataset

The data-set to be used on the project has 33 attributes and 649 instances. The 33rd attribute of the data-set is the final grade, this will be our single output node. This means that, initially, there will be 32 input nodes. One of the main purposes of the project, is treating the dataset, reducing non-important attributes. This will make the network training more efficient, since there will be less nodes to weight.

The framework we will be using restricts the inputs and outputs to be in the range [0, 1], so to train and test our network we need to first parse the data and transform all the inputs into the corresponding value in the range [0, 1].

To train and test the network we will have to divide the dataset into two separate groups. The **Training Set** and the **Test Set**, these groups will be used to train different networks, however, they will be the same for all the networks. We could use a **Validation Set**, but since we only have 649 instances, having 3 disjoint sets would mean small sets, thus, we chose not to use the Validation Set, which may or may not be used in the training of a neural network.

We will keep the Training Set size at 2/3 of the number of instances and the Test Set at 1/3, as it is usual to follow this approach.

As described in the Network section, the network may have any number of hidden layers, each with any number of nodes. We must fiddle with these values while creating different networks, in order to create better networks with faster training times and better accuracy.

The network's learn rate and momentum may also be adjusted. We already discussed where these attributes affect the algorithm in the previous section.

We can also adjust the error threshold to be achieved, which means that when the back-propagation calculated error is below the threshold, the training of the network stops.

2.2 Work Done

From the project phases section, we already concluded the first 4 items:

1. Understand the structure of a neural network
2. Understand the mechanism of the back-propagation algorithm
3. Choose a back-propagation framework and learn it
4. Learn how to train neural networks

The rest of the work will lead to the conclusion of the project.

2.3 Expected Results and Tests

From training different networks, and fiddling with the attributes referenced in the training section, we expect to see different training times and network accuracies.

We are unable to set an accuracy goal for our final network, but we hope to see a noticeable increase when comparing our final network to our first.

3 Conclusions

So far we have learned how a neural network is structured, how the back-propagation algorithm works and how to differently train networks in order to achieve different results.

Learning this subject was fun because of how easy it is to be impressed by machines learning through training. And now we have some knowledge of how it works.

4 Resources

4.1 Bibliography

`https://web.fe.up.pt/~eol/IA/1516/APONTAMENTOS/7_RN.pdf`

Neural Nets, Back Propagation, Patrick Winston, M.I.T.

`http://www.faqs.org/faqs/ai-faq/neural-nets/`

`http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-la`

`http://www.akamaiuniversity.us/PJST9_1_72`

`https://takinginitiative.wordpress.com/2008/04/23/basic-neural-network-tutorial-c-`

`https://github.com/harthur/brain`

4.2 Software

To train the network, we will use the javascript framework **BrainJS**, an open source implementation of a neural network, using the back-propagation algorithm, implemented using the basic sigmoid function.