Intelligent Systems

Neural Networks

Miguel Jiménez Benajes
Computer Engineering, UJI
Castellón, España
migueljimenezbenajes@gmail.com

***Abstract*— Neural networks are used to solve problems of classification and analysis of diverse nature that other algorithms can not solve. Voice recognition, data mining, quality control, process optimization, etc. This document is an introduction to neural networks. In addition, the work done in the various laboratory sessions of the subject EI1028 of Computer Engineering and the personal experimental results with the development environment of UNITY 3D are exposed.**

I. INTRODUCTION

"The brain is a far more open system than we ever imagined, and nature has gone every far to help us perceive and take in the world around us. It has given us a brain that survives in a changing world by changing itself". [1]

Neural networks first appeared in 1943, neurologist Warren McCulloch along with mathematician Walter Pitts wrote an article about how the neurons in our brain work. They created a simple neural network model using electronic circuits. During the following years, more advanced models of neural networks were developed, But over time the idea fell into disuse. [2]

In the early 70s the interest in the field was renewed. The first multi-layered neural network was developed in 1975, an unsupervised network. In 1986 a group of researchers developed the concept of "back-propagation" (The most commonly used process in training the neural network). However, this form of learning was very slow and required a large number of iterations. In those days was not too viable. [3]

In the following years the advance in computing has allowed to accelerate this process. In the 90s it began to be applied in different areas. For example, "Deep Blue", the machine that managed to win chess to Kaspárov in 1997.

In the last years the neural networks have returned to be in vogue. Thanks to the high speed of the current processors and to the new systems of computation in the cloud anyone can from its house prove and investigate about them.

II. THE PERCEPTRON

The perceptron is the most basic unit within neural networks. It is the simplest artificial model that exists of a neuron. Its unitary use is limited only to linearly separable problems.

Figure 1 illustrates the natural neuron model, composed of dendrites, nucleus and axon. Figure 2 illustrates the artificial model, composed of inputs, weights, a processing equation, an activation function and the output.

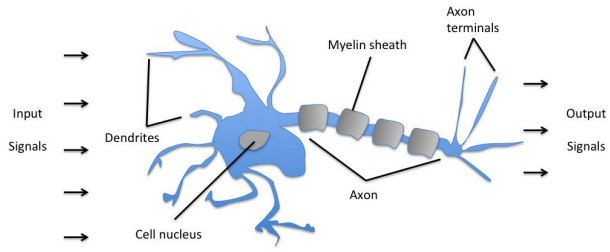

Figure 1. Neuron, Biology [4]

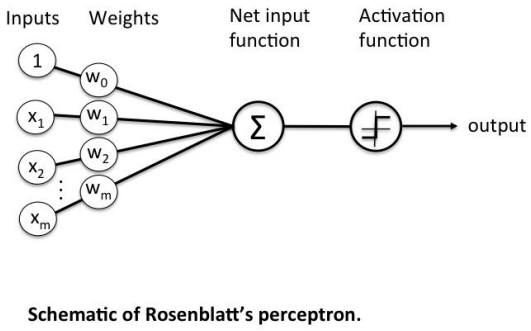

Figure 2. Neuron, Artifical [4]

The perceptron is composed of "weights". They are a series of numbers whose quantity agrees with the number of inputs. They are used in the processing stage, where the outputs are calculated from a series of inputs. The inputs, the weights and the outputs usually have values between 0 and 1 or -1 and 1.

In order to calculate the outputs, a value is obtained by the transfer function Eq. (1), "x" are the inputs, "w" the weights and "n" the number of inputs and weights. Subsequently, it is processed through an activation function that is usually the binary step function Eq. (2).

$$\alpha = x_1 w_1 + x_1 w_1 + ... + x_n w_n \;\; ; \;\; \alpha = \sum_{i=0}^{i=n} x_i w_i \quad (1)$$

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (2)$$

III. Neural Networks Training

Training is crucial in the process of creating the neural network. The product of the training process is the adjustment of the weights to obtain better results and approximate the desired function. Without a training the neural network would always give the same meaningless answers.

There are a lot of training algorithms that can be divided into three main blocks. [5]

- Supervised learning. It is the most common in perceptrones. The neural network receives two parameters, a list of inputs and a list of expected outputs. Thanks to the output and the expected output the error can be calculated. Then the weights are adjusted based on the error.
- Unsupervised learning. The network only receives the inputs and is in charge of adjusting its weights to find some pattern. Its use is widespread in the field of data mining.
- Reinforced learning. It is similar to supervised learning but instead of providing an expected output, a reward is given based on system performance. In this way it is intended to maximize the reward obtained.

IV. Multilayer Neural Networks

Many times, in order to deal with more complicated tasks or classification problems that are not linearly separable we need to make use of a network with more than one perceptron and several layers. These types of networks called multi-layer ANNs are composed of hidden layers between input and output, figure 3.

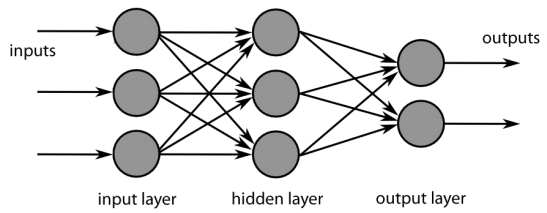

Figure 3. Multil-layer Artificial Neural Network with 1 hidden layer, 3 inputs and 2 outputs. [6]

Such networks need a more complex learning algorithm. Among all the existing ones the most outstanding is "backpropagation". Basically the weights are adjusted by calculating the error from front to back. To obtain the new weights it is necessary to make use of differential calculations, which is why a continuous activation function is necessary (the step function is not). Among all, the most important are the sigmoid function (Eq.3) and the hyperbolic tangent. (Eq. 4).

$$f(x) = \frac{1}{1+e^{-x}} \quad (3) \qquad f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (4)$$

V. First Lab Session: Perceptrons

In the first laboratory session, the basic concepts of the perceptron and its individual functioning as a linear classifier were seen. The development environment of Jupyter and Scikit-learn (a tool for machine learning in python) was also announced. Three typical classification problems were performed.

- The AND function. Through supervised learning the perceptron was taught to evaluate the function. Usually about 10 iterations were needed until the model converged.
- The OR function. Very similar to the AND, it had to be written completely.
- A classification problem of dots (red and black). A linearly separable dataset (fig. 4) formed by two elements had to be designed. The "x" element was an array of as many rows as points and two columns. The element "y" was a list of the color of the dots (0 red and 1 black).

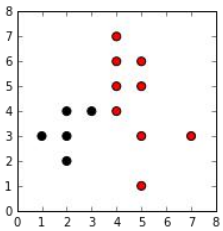

Figure 4. Dataset created on the lab session.

VI. Second And Third Lab Sessions: Multilayer Perceptrons

Throughout two laboratory sessions four problems of classification and recognition of patterns were realized. In order to solve this type of problems (non-separable linearly) multi-layer neural networks are required.

- The XOR function. The first and classic linear non-separable function problem is the XOR function. By its nature, a single perceptron can not converge correctly. The network consisted of 5 hidden neurons, 2 inputs and 1 output (fig. 5). To solve the problem we made use of stochastic gradient descent (very used in the technique of backpropagation). Sometimes the network does not converge because it stays stuck in a local minimum (fig. 6), This can happen when using a Gradient decent-type learning algorithm [7]. For better results and avoid the local minimum it is better to make use of an optimization algorithm in the learning process.
- The second problem was an example of classification. First a dot dataset was generated in a 2D space with some noise (fig. 7). Subsequently the dataset was divided into two parts, one dedicated to the training phase and another dedicated to the test phase. Finally the neural network was trained and the percentage of

successes in the test was verified. Thanks to the "plot" module of the environment, it was possible to visualize the area of probability generated by the neural network (fig. 8). For this problem was made use of the 'lbfgs' solver of sklearn. This solver implements an optimization algorithm belonging to the family of quasi-Newton methods. Its name is because it is not a complete Newton method, this allows to find the global minimum without consuming much time. Its use is highly recommended for small-scale problems that need to converge quickly [8].

- The third problem was to recognize handwritten digits. A dataset consisting of 1797 images of digits in size 8x8 pixels grayscale was provided. Subsequently it was divided into two sets, the training set and the test set, this process is called cross-validation. Then the model was created using the stochastic gradient descent used in the XOR problem and the network was trained with the training set. The last phase was the analysis of the results. The accuracy, recall and f1-score of each number were obtained along with the mean of all. In addition, the accuracy of the function was evaluated through the confusion matrix and the training curve was generated. Finally some numbers were visualized along with their prediction and their real value (fig. 9). As work, this process had to be iterated 100 times, each time generating a new test set and calculating the mean and standard deviation of the values.

- The fourth and final problem was the recognition of traffic signs German roads. Today's high-tech cars like the Audi A8 or the Tesla Model S [9] are able to detect signals and warn us when we are committing a violation or assisting in driving. This task is not easy, many environmental factors have to be taken into account to obtain good results. The dataset contained more than 40 types of signals with more than 50,000 images. It was obtained from the INI (Institute for Neural Computation) of Germany. For this case (as it usually happens in real datasets), the images had to be processed to cut them, to scale them (they all have to have the same number of pixels) and to eliminate the green and blue channels (red is the one that provides the most information in this case) (fig. 10). Following the same steps as in the problem of digits we had to create a neural network capable of identifying up to 10 classes of signals with 15 tracks each, making a total of 4500 images. Finally, we had to do the analysis of the data obtained (classification report, confusion matrix and loss curve).

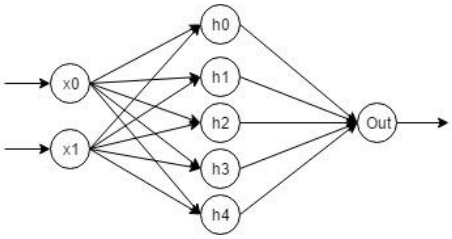

Figure 5. Neural network with 2 inputs, 5 hidden neurons and 1 output.

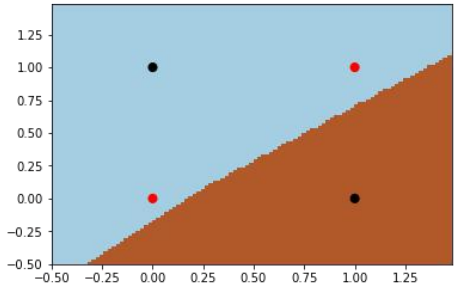

Figure 6. Example algorithm result for a non-converging network.

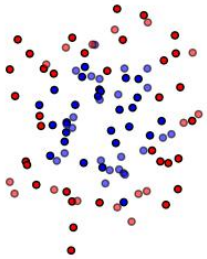

Figure 7. Example dataset generated on the classification problem.

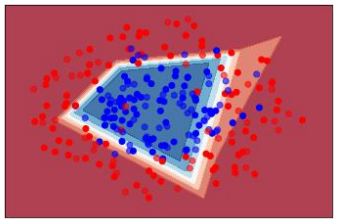

Figure 8. Decision boundary area generated on the neural network.

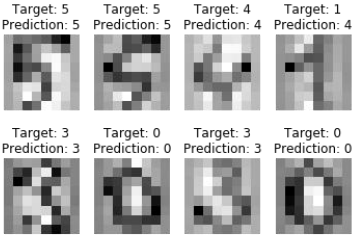

Figure 9. The target and prediction for a few images of the dataset.

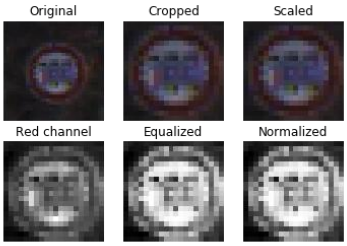

Figure 10. Image processing steps.

VII. Neural Network in Unity 3D

As an experiment and to learn more about neural networks it has been developed in C # a neural network model in the Unity 3D video game engine [10]. This multilayer model is general purpose and is intended to mimic the downward stochastic gradient used in practices. It makes use of the sigmoid activation function and its derivative. Currently, the model only supports a hidden layer in backpropagation learning. The most important methods are described below.

- `UpdateNeuronal(List<double> inputs)`. This method is used to evaluate the network. It receives a list of inputs and returns a list of outputs.
- `train(List<double> inp, List<double> outp)`. This method is used at each iteration to train the neural network. It receives a list of inputs, a list of expected outputs and returns all outputs on the network.

When you create the network, you can specify a series of parameters to fit as well as possible to our problem. Number of inputs, number of outputs, number of hidden layers, number of neurons per hidden layer, bias and learning rate (fig. 11).

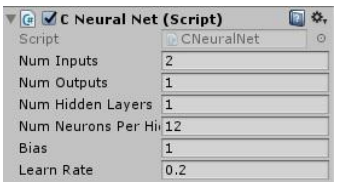

Figure 11. Neural Net with some typical parameters.

VIII. XOR Problem in Unity 3D

In order to check the proper functioning of the neural network described above it has been created a test environment for the XOR function [11]. The script is responsible for creating and executing neural network training and test. It is possible to specify the number of iterations used in the training and also to add noise to the test to check its tolerance (fig. 12). At the end of the run, the percentage of hits obtained by the neural network in the test is displayed on the screen (fig. 13).

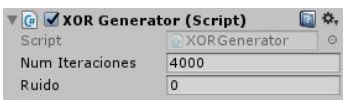

Figure 12. XOR generator parameters

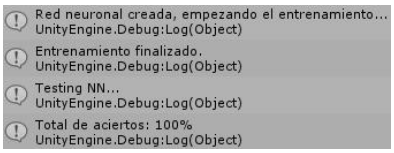

Figure 13. Example output from XOR generator

It seems that with a network composed of 5 neurons in the hidden layer and a learning rate of 0.5 converges correctly around the 4000 iterations (tab. 1).

Num iterations	*% success*
1	50%
100	50%
1000	75%
2000	50%
>4000	100%

Table 1. Num iterations vs %success on the XOR generator problem

IX. Classification Problem in Unity 3D.

In order to go deeper into the use of neural networks, an environment has been created to generate the problem of classification of points (red and blue) seen in the second laboratory session. Figure 14 shows the parameters that the script supports, and the dot space generated from these parameters is shown in Figure 15.

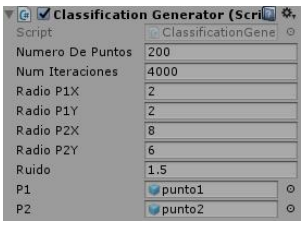

Figure 14. Parameters on the classification problem generator

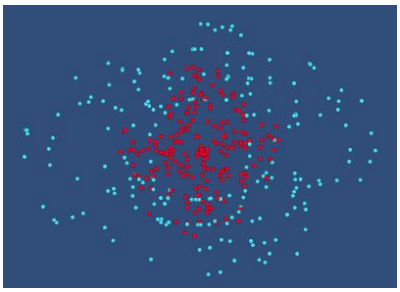

Figure 15. Result of the execution of the script with the parameters used in figure 14.

Table 2 shows the % of hits in relation to the noise and the number of iterations used. The neural network used in this example consists of a hidden layer with 12 neurons and 0.2 of learn rate. The training set consists of 300 dots and 100 dots for the test. It can be appreciated that increasing the number of iterations and reduce the noise increases the number of hits. In addition, with more iterations the results are more predictable.

numIterat\noise	*0*	*1*	*1.5*	*2*	*3*
1 x 300	62%	64%	41%	58%	58%
2 x 300	53%	58%	46%	60%	49%
5 x 300	72%	76%	76%	63%	54%
10 x 300	84%	79%	68%	63%	55%
50 x 300	96%	91%	82%	79%	64%

Table 2. Success rate between the number of iterations and the noise used.

X. CONCLUSIONS

Despite being more than 70 years old, neural networks are the order of the day. Thanks to the power of our processors and the latest advances by google and universities around the world, the neural networks have recovered from an old drawer that more than one looked like lost. Artificial intelligence begins to emerge from universities and laboratories to penetrate the business world. Some people call it the fourth industrial [13] revolution and the media are beginning to expose this issue because of the problems of unemployment that it entails. All this would be impossible without neural networks.

As the work developed in the laboratory sessions, it has helped me little to understand the subject. I have learned more in depth on the subject on my own and working to reproduce a network in the UNITY environment. However, together with the theory classes, they are a good introduction to the subject and serve to take some first steps in the area.

REFERENCES

[1] Norman Doidge. "The Brain That Changes Itself".

[2] https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html.

[3] https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history2.html.

[4] http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html.

[5] http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-2-learning/8.

[6] http://magizbox.com/training/deep_learning/site/introduction/.

[7] Atakulreka A., Sutivong D. (2007) Avoiding Local Minima in Feedforward Neural Networks by Simultaneous Learning. In: Orgun M.A., Thornton J. (eds) AI 2007: Advances in Artificial Intelligence. AI 2007. Lecture Notes in Computer Science, vol 4830. Springer, Berlin, Heidelberg

[8] https://optimization.mccormick.northwestern.edu/index.php/Quasi-Newton_methods

[9] Vehicles with tsr: https://en.wikipedia.org/wiki/Traffic_sign_recognition.

[10] https://gist.github.com/mijim/1082e852e0bd4eee5b846cb509fc0130.

[11] https://gist.github.com/mijim/c3469be1aef25d597ebd015c54e47fa6.

[12] https://gist.github.com/mijim/525adbc206073914976a7df686462671.

[13] https://www.weforum.org/about/the-fourth-industrial-revolution-by-klaus-schwab