Recent Trends in Communication and Computer Networks , ComNet

An Insight into the Fundamentals of Neural Networks and Working of Back Propagation Algorithm

Srishti P Mishra
ABB GISPL, Bangalore, India
shiny.srishti@gmail.com

***Abstract*—This research paper aims at explaining the fundamentals of neural network with a very different approach by first taking a real life example to bring simplicity. Also, beginning by explaining a biological neuron which is the real working model of an artificial neuron and diving in to discuss the structural aspects of a neural network along with the activation functions that act as decision makers for an input to a neuron. Further it covers the mathematical aspects of training a neural network by discussing the back propagation algorithm which has been widely accepted as the best known method to learn a neural network. Although a theoretical insight into the concept of neural networks, this paper is a precursor to an implementation of Financial Predictor which is still in progress.**

***Index Terms*— activation function, backpropagation, hidden unit, normalizing function, perceptron, synapses, spikes.**

I. INTRODUCTION

Each year, the field of computer science becomes more sophisticated as new types of technologies hit the market. Despite that, the problem of developing intelligent agents that will precisely simulate human brain activity is still unsolved. One of the most prominent models of intelligent agents built in computer memory is represented by neural networks (NN). Neural networks have been used with computers since the 1950s. Through the years, many different models have been presented. The Perceptron is one of the earliest neural networks. It was an attempt to understand human memory, learning and cognitive processes, to construct a computer capable of "human-like thought".

II. NEURON IN HUMAN BRAIN

A neuron consists of a cell body, with various extensions from it. Most of these are branches called dendrites. There is one much longer process (possibly also branching) called the axon. The dashed line shows the axon hillock, where transmission of signals starts.
The boundary of the neuron is known as the cell membrane. There is a voltage difference (the membrane potential) between the inside and outside of the membrane.
If the input is large enough, an action potential is then generated. The action potential (neuronal spike) then travels down the axon, away from the cell body. The connections between one neuron and another are called **synapses**. Information always leaves a neuron via its axon (Figure 1 below), and is then transmitted across a synapse to the receiving neuron. Neurons only fire when input is bigger than some threshold. It should,

Grenze ID: 02.COMNET.2015.3.19

however, be noted that firing doesn't get bigger as the stimulus increases, it’s an all or nothing arrangement.

III. Input to a neuron

Synapses can be excitatory or inhibitory.
Spikes (signals) arriving at an excitatory synapse tend to cause the receiving neuron to fire. Spikes (signals) arriving at an inhibitory synapse tend to inhibit the receiving neuron from firing.
The cell body and synapses essentially compute (by a complicated chemical/electrical process) the difference between the incoming excitatory and inhibitory inputs (spatial and temporal summation). When this difference is large enough (compared to the neuron's threshold) then the neuron will fire. Roughly speaking, the faster excitatory spikes arrive at its synapses the faster it will fire (similarly for inhibitory spikes).

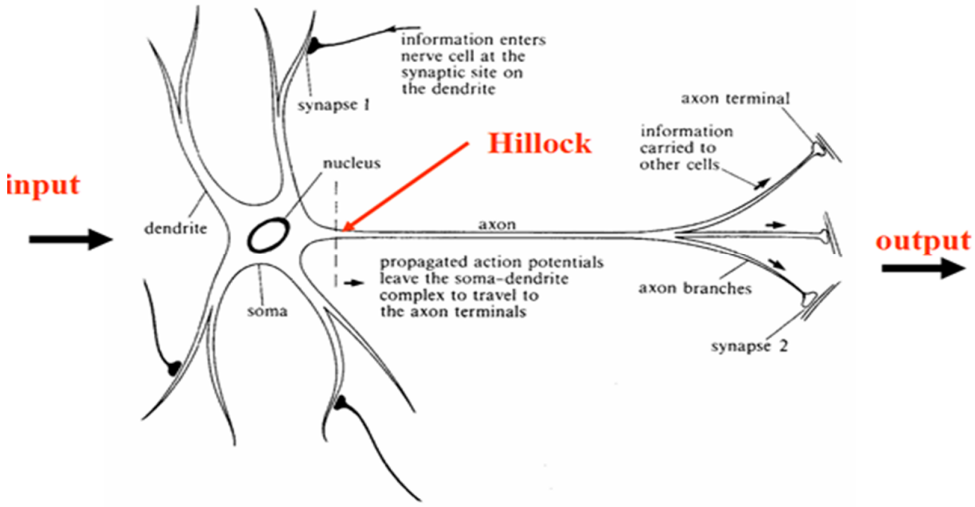

Figure 1 - A Biological Neuron

IV. Neural Network-A real life analogy

A way you can think about the perceptron is that it's a device that makes decisions by weighing up evidence. Suppose the weekend is coming up, and you've heard that there's going to be a cheese festival in your city. You like cheese, and are trying to decide whether or not to go to the festival. You might make your decision by weighing up three factors:

1. Is the weather good?
2. Does your friend want to accompany you?
3. Is the festival near public transit? (You don't own a car).

We can represent these three factors by corresponding binary variables a, b and c, and for instance, we'd have a=1 if the weather is good, and a=0 if the weather is bad. Similarly, if your friend wants to go b=1, and b=0 if not. And similarly again for and public transit.
Now, suppose you absolutely adore cheese, so much so that you're happy to go to the festival even if your friend is uninterested and the festival is hard to get to. But perhaps you really loathe bad weather, and there's no way you'd go to the festival if the weather is bad. You can use perceptron to model this kind of decision-making. One way to do this is to choose a weight w1=6 for the weather, and w2=2 and w3=2 for the other two conditions. The larger value indicates that the weather matters a lot to you, much more than whether your friend joins you, or the nearness of public transit. Finally, suppose you choose a threshold of 6 for the perceptron. With these choices, the perceptron implements the desired decision-making model, outputting a 1 whenever the weather is good, and 0 whenever the weather is bad. It makes no difference to the output whether your friend wants to go, or whether public transit is nearby.
By varying the weights and the ***threshold***, we can get different models of decision-making. For example, suppose we instead chose a threshold of 3. Then the perceptron would decide that you should go to the festival whenever the weather was good or when both the festival was near public transit and your friend was willing to join you. In other words, it'd be a different model of decision-making. Dropping the threshold

means you're more willing to go to the festival. This example illustrates how a perceptron can weigh up different kinds of evidence in order to make decisions. And it should seem plausible that a complex network of perceptrons could make quite subtle decisions.

V. Neural Network Structure and Working

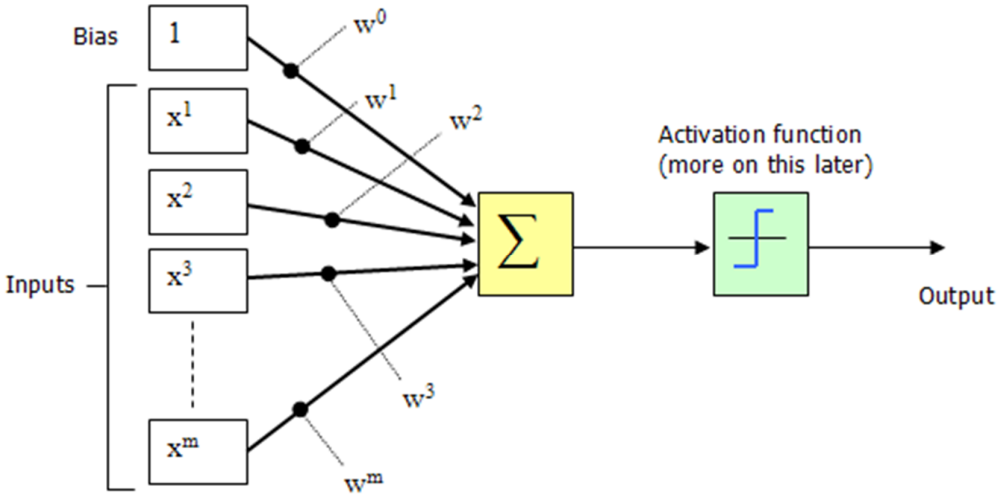

Figure 2- Artificial Neuron with Bias as additional Unit

This configuration is actually called a **Perceptron** (an invention of Rosenblatt [1962]). It consists of neurons. Some of them, known as **input units**, are designed to receive various forms of information from the outside world that the network will attempt to learn about, recognize, or otherwise process. Other units sit on the opposite side of the network and signal how it responds to the information it's learned; those are known as **output units**. In between the input units and output units are one or more layers of **hidden units**, which, together, form the majority of the artificial brain. The connections between one unit and another are represented by a number called a **weight**, which can be either positive (if one unit excites another) or negative (if one unit suppresses or inhibits another). The higher the weight, the more influence one unit has on another.

The hidden unit is responsible for **Summing** and **Activation Functions**. The information sent to the neuron is multiplied by corresponding weights and then is added together and used as a parameter within an activation function. In a biological context, a neuron becomes activated when it detects electrical signals from the neurons it is connected to. If these signals are sufficient, the neuron will become "activated" - it will send electrical signals to the neurons connected to it. An activation function is similar: the artificial neuron will output a value based on these inputs. It is almost always the case that a neuron will output a value between [0, 1] or [-1, 1], and this normalization of data occurs by using the summed inputs as a parameter to a normalizing function, called an "activation function".

There are three kinds of Activation Functions generally used.

1. Threshold Function: a simple function that compares the summed inputs to a constant, and depending on the result, may return a -1, 0, or 1. Specifically, for summed input

$$f(x)=\begin{cases} 1 & if\ x>0 \\ 0 & if\ x=0 \\ -1 & if\ x<0 \end{cases}$$

2. Piecewise-Linear Function: if the summed inputs are in the range [-0.5, 0.5], the value is kept as before, while anything else will return a -1 or 1:

$$f(x)=\begin{cases} 1 & if\ x\geq\frac{1}{2} \\ 0 & if\ -\frac{1}{2}<x<\frac{1}{2}=0 \\ -1 & if\ x\leq-\frac{1}{2} \end{cases}$$

3. Hyperbolic Tangent Function: a continuous function with a domain of (-∞,∞) and a range of (-1, 1)

$$\tanh(x)=\frac{e^{2x}-1}{e^{2x}+1}$$

VI. Back Propagation

In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. The back propagation algorithm is useful in that it generalizes the Delta Rule for single-layer feed forward neural networks to multilayer feed forward ones. One of the biggest problems in training neural networks is the Credit Assignment Problem: how does one assign credit to individual neurons for errors in the final outputs of a neural network?

This process requires that the neural network compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back propagation algorithm is the most widely used method for determining the EW.

The back-propagation algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each EW by first computing the EA, the rate at which the error changes as the activity level of a unit is changed. For output units, the EA is simply the difference between the actual and the desired output. To compute the EA for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the EAs of those output units and add the products. This sum equals the EA for the chosen hidden unit. After calculating all the EAs in the hidden layer just before the output layer, we can compute in like fashion the EAs for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the EA has been computed for a unit, it is straight forward to compute the EW for each incoming connection of the unit. The EW is the product of the EA and the activity through the incoming connection.

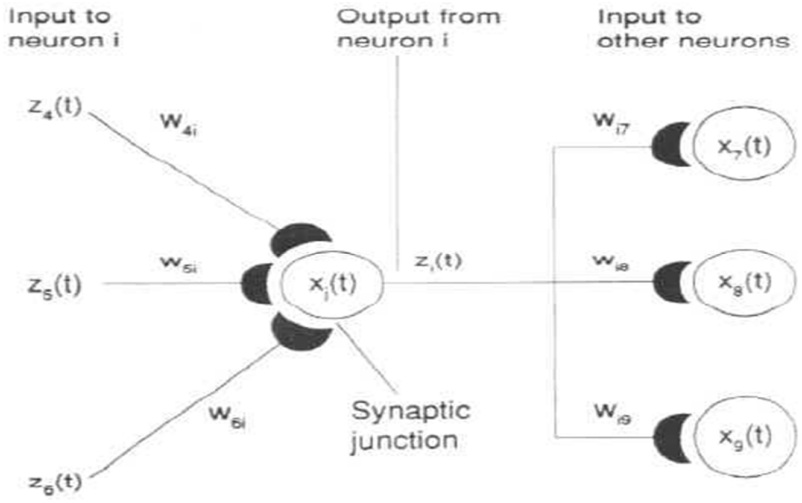

Figure 3 – Back Propagation Algorithm

VII. Step by Step procedure of Back Propagation Algorithm

A unit in the output layer determines its activity by following a two step procedure.
First, it computes the total weighted input xj, using the formula:

$$X_j = \sum_i y_i W_{ij}$$

where yi is the activity level of the jth unit in the previous layer and Wij is the weight of the connection between the ith and the jth unit.
Next, the unit calculates the activity yj using some function of the total weighted input. Typically, we use the sigmoid function:

$$y_j = \frac{1}{1+e^{-x_j}}$$

Once the activities of all output units have been determined, the network computes the error E, which is defined by the expression:

$$E = \frac{1}{2}\sum_i \left(y_i - d_i\right)^2$$

where yj is the activity level of the jth unit in the top layer and dj is the desired output of the jth unit.
The back-propagation algorithm consists of four steps:
1. Compute how fast the error changes as the activity of an output unit is changed. This error derivative (EA) is the difference between the actual and the desired activity.

$$\mathrm{EA}_j = \frac{\partial E}{\partial y_j} = y_j - d_j$$

2. Compute how fast the error changes as the total input received by an output unit is changed. This quantity (EI) is the answer from step 1 multiplied by the rate at which the output of a unit changes as its total input is changed.

$$\mathrm{EI}_j = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \times \frac{dy_j}{dx_j} = \mathrm{EA}_j y_j \left(1 - y_j\right)$$

3. Compute how fast the error changes as a weight on the connection into an output unit is changed. This quantity (EW) is the answer from step 2 multiplied by the activity level of the unit from which the connection emanates.

$$\mathrm{EW}_{ij} = \frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial W_{ij}} = \mathrm{EI}_j y_i$$

4. Compute how fast the error changes as the activity of a unit in the previous layer is changed. This crucial step allows back propagation to be applied to multilayer networks. When the activity of a unit in the previous layer changes, it affects the activities of all the output units to which it is connected. So to compute the overall effect on the error, we add together all these separate effects on output units. But each effect is simple to calculate. It is the answer in step 2 multiplied by the weight on the connection to that output unit.

$$\mathrm{EA}_i = \frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial y_i} = \sum_j \mathrm{EI}_j W_{ij}$$

By using steps 2 and 4, we can convert the EAs of one layer of units into EAs for the previous layer. This procedure can be repeated to get the EAs for as many previous layers as desired. Once we know the EA of a unit, we can use steps 2 and 3 to compute the EWs on its incoming connections.

VIII. Conclusion and Future work

The computational systems we write are procedural; a program starts at the first line of code, executes it, and goes on to the next, following instructions in a linear fashion. A true neural network does not follow a linear path. Rather, information is processed collectively, in parallel throughout a network of nodes. The computing world has a lot to gain from neural networks. Their ability to learn by example makes them very flexible and

powerful. Furthermore, there is no need to devise an algorithm in order to perform a specific task; i.e. there is no need to understand the internal mechanisms of that task.
The aim of this paper was to understand the fundamentals of Neural Networks in order to implement it for predicting stock prices which is still under progress.

REFERENCES

[1] Wojciech Gryc , Neural Networks in "Neural Network Predictions of Stock Price Fluctuations" .
[2] Salim Lahmiri in "Neural Networks and Investor Sentiment Measures for Stock Market Trend Prediction".
[3] Ciumac Sergiu in "Financial Predictor via Neural Network" from http://codeproject.com.
[4] "Application of Neural Network in Analysis of Stock Market" from *http://www.ijcset.com*.
[5] Ms Sonali B Maind, in "Research Paper on Basic of Artificial Neural Network".