# The math behind ANN (ANN- Part 2)

## Introduction

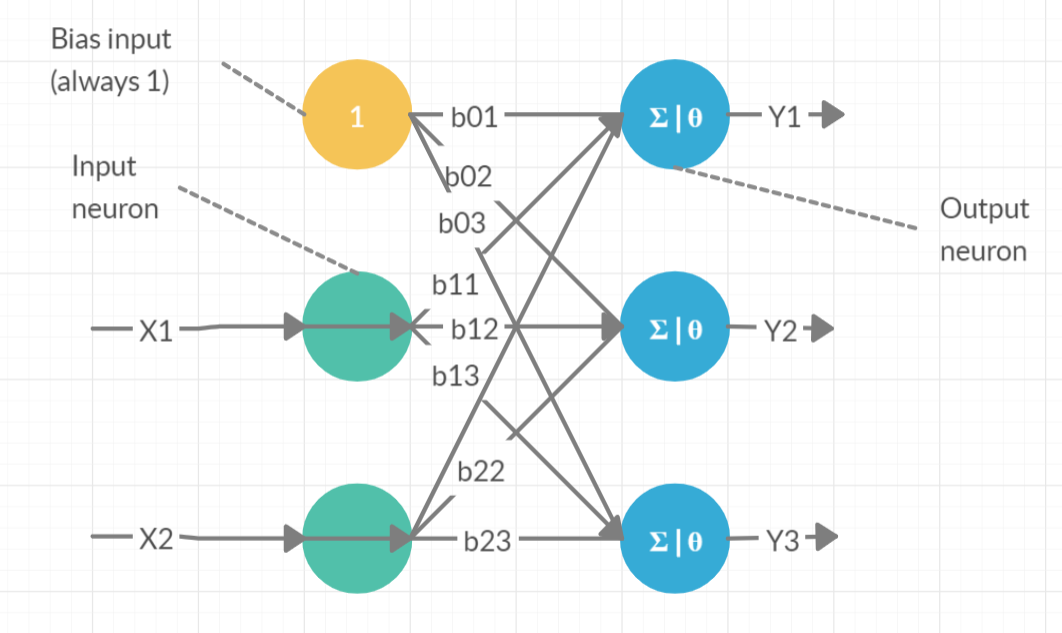
This is second in the series of blogs about neural networks. In this blog, we will discuss the backpropagation algorithm. In the previous blog, we have seen how a single perceptron works when the data is linearly separable. In this blog, we will look at the working of a multi-layered perceptron (with theory) and understand the math behind backpropagation.

## Multi-layer perceptron

An MLP is composed of one input layer, one or more layers of perceptrons called hidden layers, and one final perceptron layer called the output layer. Every layer except the input layer is connected with a bias neuron and is fully connected to the next layer.

### Perceptron

In the previous blog, we have seen a perceptron with a single TLU. A Perceptron with two inputs and three outputs is shown below. Generally, an extra bias feature is added as input. It represents a particular type of neuron called bias neuron. A bias neuron outputs one all the time. This layer of TLUs is called a perceptron.



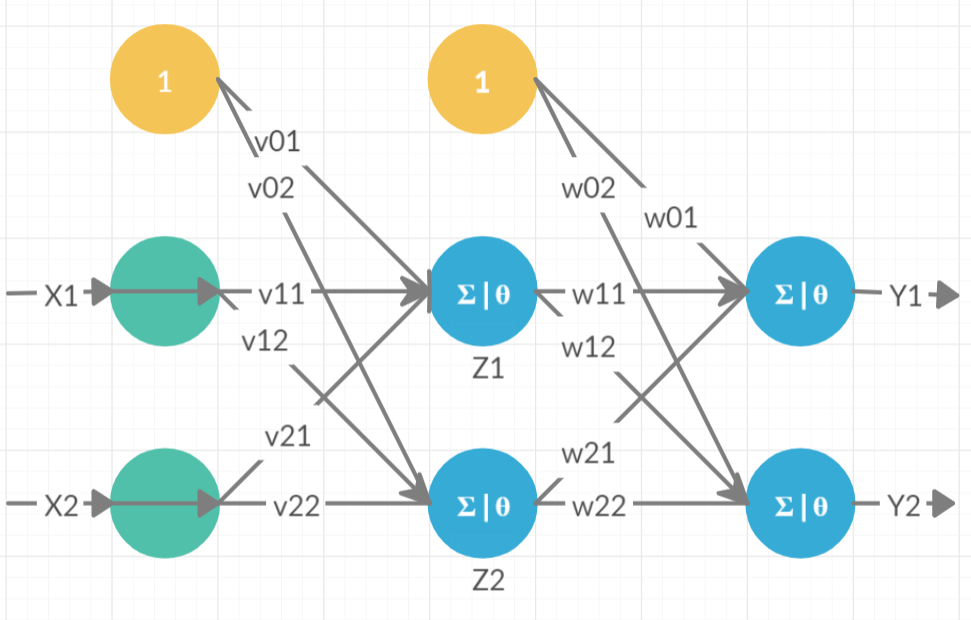
In the above perceptron, the inputs are x1 and x2. The outputs are y1, y2 and y3. Θ (or f) is the activation function. In the last blog, the step function is taken as the activation function. There are other activation functions such as:

**Sigmoid function**: It is S-shaped, continuous and differentiable where the output ranges from 0 to 1.

**Hyperbolic Tangent function**: It is S-shaped, continuous and differentiable where the output ranges from -1 to 1.

### Training

Training a network in ANN has three stages, feedforward of input training pattern, backpropagation of the error and adjustment of weights. Let us understand it using a simple example.  
Consider a simple two-layered perceptron as shown:



Nomenclature:

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Meaning** | **Symbol** | **Meaning** |
|  | Input neuron |  | Output neuron |
|  | Input value |  | Output value |
|  | Hidden neuron |  | The output of a hidden neuron |
|  | The portion of error correction for weight |  | The portion of error correction for weight |
|  | Weight of j to k |  | weight of i to j |
|  | Learning rate |  | Actual output |
| *f* | Activation function |  |  |

During feedforward, each input unit  receives input and broadcasts the signal to each of the hidden units . Each hidden unit then computes its activation and sends its signal () to each output unit. Each output unit computes its activation () to form the response to the input pattern.

During training, each output unit compares its predicted output with its actual output to determine the error associated with that unit. Based on this error, is computed. This is used to distribute the error at the output unit back to all input units in the previous layers. Similarly, is computed for all hidden layers which is propagated to the input layer.

The and are used to update the weights and respectively. The weight adjustment is based on gradient descent and is dependent on error gradient (), learning rate () and input to the neuron.

Mathematically this means the following:

##### Feedforward loop:

1. Each input unit () receives the input and broadcasts this signal to all units to the hidden layers .

Hidden layer:

1. Each hidden unit () sums its weighted input signals
2. The activation function is applied to this weighted sum to get the output. (where f is the activation function).
3. Each hidden layer send this signal () to the output layers.

Output layer

1. Each output unit () sums its weighted input signals y
2. The activation function is applied to this weighted sum to get the output. (where f is the activation function).

##### Backpropagation of error

Output layer

1. The error information term () is computed at every output unit ().
2. This error is propagated back to the hidden layer. (later weights will be updated using this )

Hidden layer

1. Each hidden unit () sums its weighted error from the output layer
2. The derivative of the activation function is multiplied to this weighted sum to get the weighted error information term at the hidden layer. (where f is the activation function).
3. This error is propagated back to the initial layer.

##### Update weights and biases

1. The weights are updated based on the error information terms
   1. , where
   2. , where

Steps 1 to 12 are done for each training epoch until a stopping criterion is met.

### Derivation of learning rules

In every loop while training, we are changing the weights ( and ) to find the optimal solution. What we want to do is to find the effect of changing the weights on the error, and minimise the error using gradient descent.

The error gradient that has to be minimised is given by:

The effect of changing an outer layer weight () on the error is given by:

Therefore

The effect of changing the weight of a hidden layer weight () on the error is given by:

Therefore

This way, for any number of layers, we can find the error information terms and using gradient descent, we can minimise the error and find optimal weights for the ANN. In the next blog, we will implement ANN on the Titanic problem and compare it with logistic regression.

### Reference

Fausett, L., 1994. *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, Inc..