S21CSE471: Statistical Methods in AI **Spring 2021**

Introduction to Neural Networks

Prepared by: Somya Lalwani, Sridhar M, Suman Chowdhury

In this note, we discuss basics of Artificial Neural Network and it's application as linear regressor and classifier. Also, activation functions have been discussed in this document

1 Introduction

1.1 Artificial Neural Network

Artificial neural networks (ANNs), or simply called neural networks (NNs), are computing systems vaguely inspired by the biological neural networks that constitute animal brains.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it.

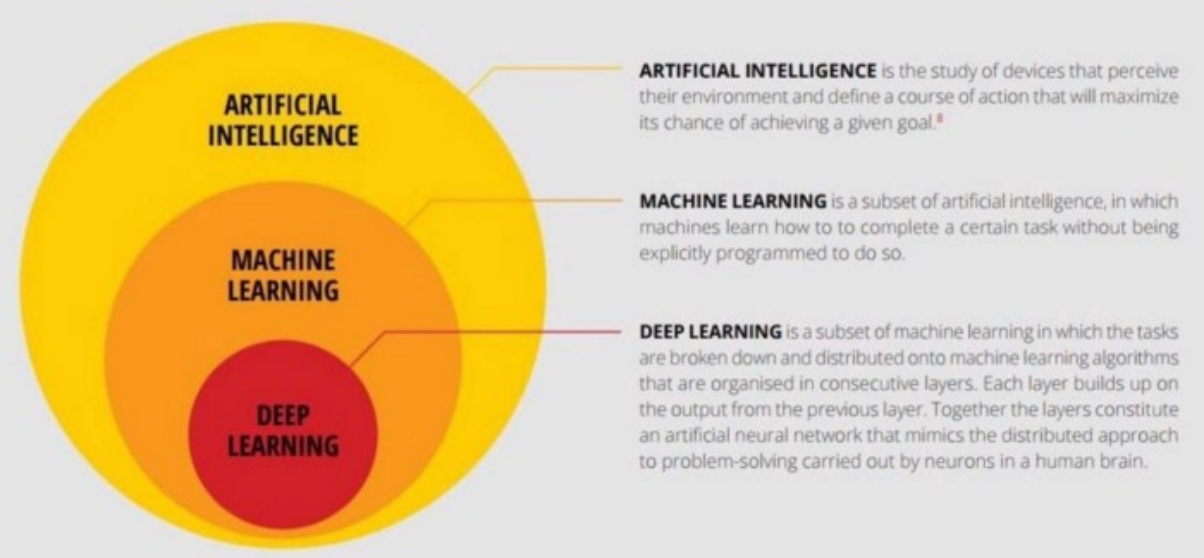


Figure 1: The neural network is this kind of technology that is not an algorithm, it is a network that has weights on it, and you can adjust the weights so that it learns. You teach it through trials." — Howard Rheingold

1.2 Definition

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input.

The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

1.3 Why neural network?

Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of the data you store and manage.

They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled data-set to train on. Neural networks can also extract features that are fed to other algorithms for clustering and classification.

1.4 High level overview

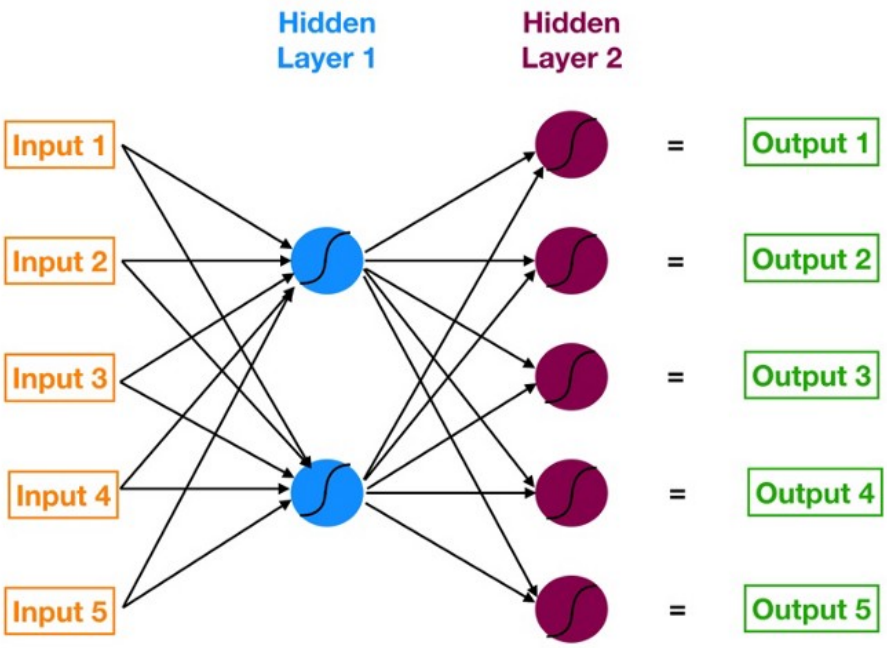


Figure 2: A Simple Neural Network

Starting from the left, we have:

1. The input layer of our model in orange.
2. Our first hidden layer of neurons in blue.
3. Our second hidden layer of neurons in magenta.
4. The output layer (a.k.a. the prediction) of our model in green.

The arrows that connect the dots show how all the neurons are interconnected and how data travels from the input layer all the way through to the output layer.

Also, the neural network learns from its mistake using a process known as back-propagation.

2 Architecture of Neural Network

2.1 Unit/Neuron

These are functions which contain weights and biases in them and wait for the data to come them. After the data arrives, they, perform some computations and then use an activation function to restrict the data to a range(mostly).

Think of these units as a box containing the weights and the biases. The box is open from 2 ends. One end receives data, the other end outputs the modified data. The data then starts to come into the box, the box then multiplies the weights with the data and then adds a bias to the multiplied data. This is a single unit which can also be thought of as a function.

2.2 Weights / Parameters / Connections

Weights play an important role in Neural Network, every node/neuron has some weights. Neural Networks learn through the weights, by adjusting weights the neural networks decide whether certain features are important or not.

2.3 Biases

Bias allows you to shift the activation function by adding a constant (i.e. the given bias) to the input. Bias in Neural Networks can be thought of as analogous to the role of a constant in a linear function, whereby the line is effectively transposed by the constant value.

2.4 Hyperparameter

Hyperparameters are the variables which determines the network structure(Eg: Number of Hidden Units) and the variables which determine how the network is trained(Eg: Learning Rate). Hyperparameters are set before training(before optimizing the weights and bias).

2.5 Activation Function

These are also known as mapping functions. They take some input on the x-axis and output a value in a restricted range(mostly). They are used to convert large outputs from the units into a smaller value — most of the times — and promote non-linearity in your NN. Your choice of an activation function can drastically improve or hinder the performance of your NN. Some of the common activation functions are:

1. Sigmoid
2. Tanh
3. Rectified Linear Unit (ReLU)
4. Leaky ReLU

2.6 Layers

These are what help a Neural Network gain complexity in any problem. Increasing layers(with units) can increase the non-linearity of the output of an NN. Each layer contains some amount of Units. There are 2 layers which every NN has. Those are the input and output layers. Any layer in between those is called a hidden layer.

1. Input Layer: The input layer consists of inputs that are independent variables. These inputs can be loaded from an external source such as a web service or a CSV file. In simple terms, these variables are known as features
2. Output Layer: This is the last layer in the neural networks and receives input from the last node in the hidden layer. This layer can be
   - Continuous (stock price)
   - Binary (0 or 1)
   - Categorical (Cat or Dog or Duck)
3. Hidden Layer: These lie between the input layer and the output layer. In this layer, the neurons take in a set of weighted inputs and produce an output with the help of the activation function.

2.6.1 Working with layers

Step 1: In this step, the input values and weights are multiplied and biased is added and are summed up together.

Step 2: In this step, we apply activation function, Activation functions are used to introduce non-linearity to neural networks. these neurons apply different transformations to the input data.

Step 3: In this step, it is passed through all the hidden layers and then passed to the output layer.

3 Applications

3.1 Regression

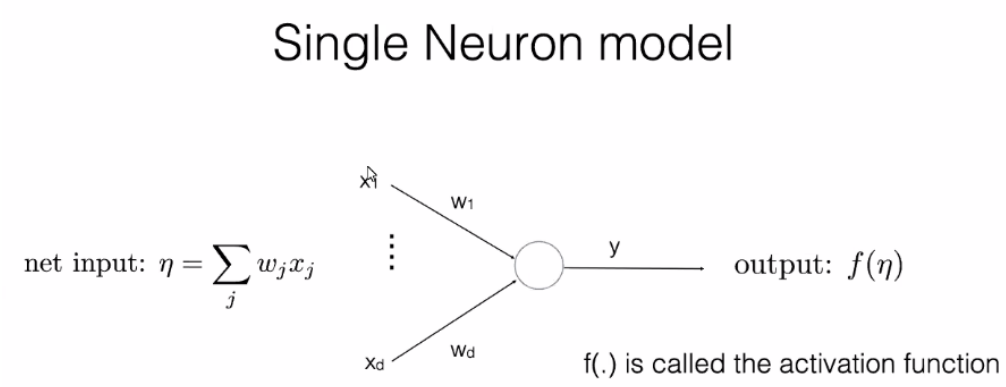


Figure 3: Single neuron

Single neuron model :

1. Using the output y, back propagate and update the value of weights. Similar to gradient descent where we want to minimize the loss function.
2. Input to neuron is a single vector X of d dimensions with weight associated with each x_i. Let y be summation of Wx. We want to minimize summation of square of difference between y and actual y.

3.2 Classification

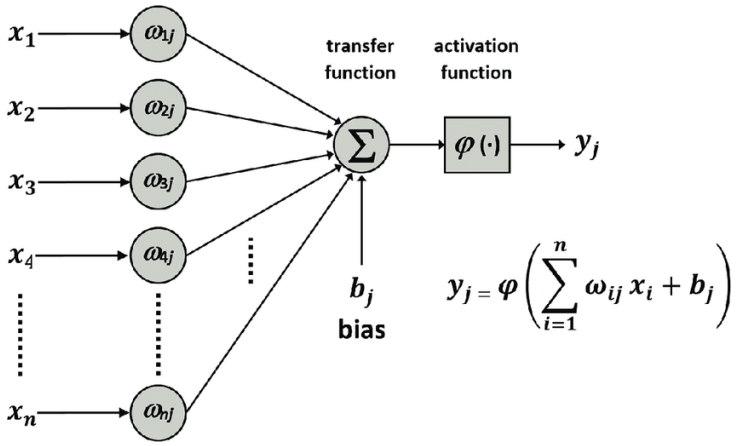
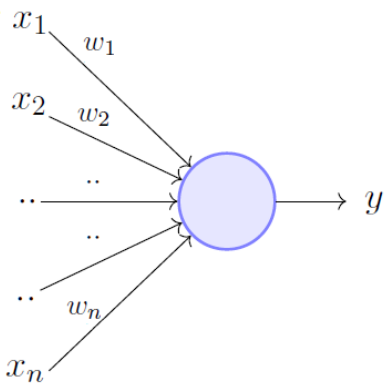


Figure 4: Classification methodology

The idea is to define the class by putting constraints on y and the using bias as threshold where sign of y will give classification.



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

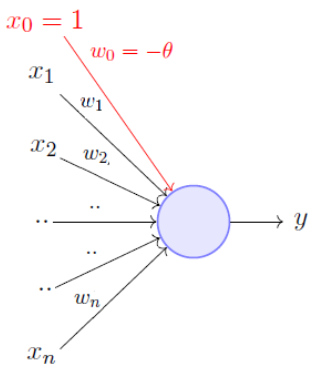
$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

The Perceptron Model is a more general computational model than McCulloch-Pitts neuron. It takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0. Rewriting the threshold as shown above and making it a constant input with a variable weight, we would end up with something like the following:

A single perceptron can only be used to implement linearly separable functions. It takes both real and boolean inputs and associates a set of weights to them, along with a bias.

3.2.1 Perceptron learning algorithm

Our goal is to find the w vector that can perfectly classify positive inputs and negative inputs in our data



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

Perceptron Learning algorithm

$$\Delta W(k) = W(k+1) - W(k)$$

$$\Delta W(k) = 0 \begin{cases} \text{if } W(k)^T X(k) > 0 \ \& \ y(k) = 1, \\ \text{or } W(k)^T X(k) < 0 \ \& \ y(k) = 0. \end{cases}$$

$$\Delta W(k) = \begin{cases} X(k) & \text{if } W(k)^T X(k) \leq 0 \ \& \ y(k) = 1, \\ -X(k) & \text{if } W(k)^T X(k) \geq 0 \ \& \ y(k) = 0. \end{cases}$$

- Error correcting algorithm: at every incorrect classification we perform 'local' correction

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize **w** randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

Figure 5: Perceptron learning algorithm

We initialize w with some random vector and then iterate over all the examples in the data, (P U N) both positive and negative examples. Now if an input x belongs to P, ideally what should the dot product

w.x be? I'd say greater than or equal to 0 because that's the only thing what our perceptron wants at the end of the day so let's give it that. And if x belongs to N, the dot product MUST be less than 0.

Case 1: When x belongs to P and its dot product w.x ≤ 0

Case 2: When x belongs to N and its dot product w.x ≥ 0

Only for these cases, we are updating our randomly initialized w. Otherwise, we don't touch w at all because Case 1 and Case 2 are violating the very rule of a perceptron. So we are adding x to w (vector addition) in Case 1 and subtracting x from w in Case 2.

3.2.2 Why would the specified update rule work?

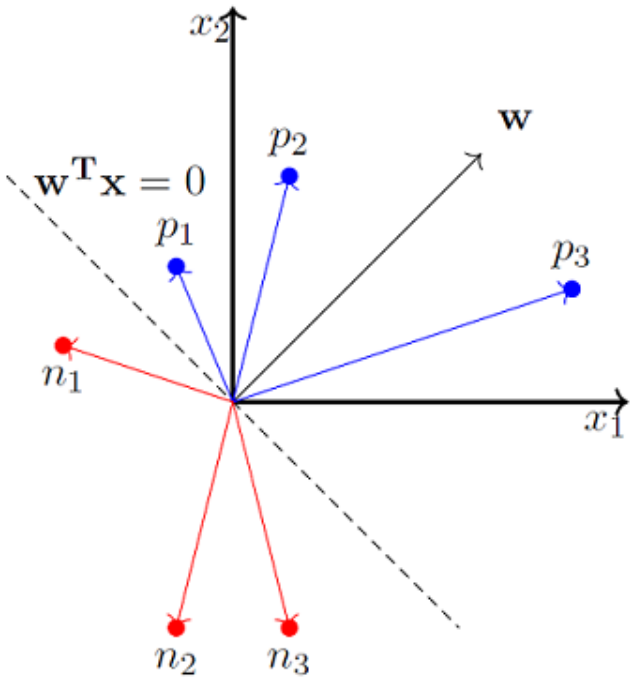
We have already established that when x belongs to P, we want w.x ¿ 0, basic perceptron rule. What we also mean by that is that when x belongs to P, the angle between w and x should be less than 90 degrees because the cosine of the angle is proportional to the dot product.

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||} \quad \Big| \quad cos\alpha \propto \mathbf{w}^T\mathbf{x}$$

$$\text{So if } \mathbf{w}^T\mathbf{x} > 0 \Rightarrow cos\alpha > 0 \Rightarrow \alpha < 90$$

$$\text{Similarly, if } \mathbf{w}^T\mathbf{x} < 0 \Rightarrow cos\alpha < 0 \Rightarrow \alpha > 90$$

So whatever the w vector may be, as long as it makes an angle less than 90 degrees with the positive example data vectors (x E P) and an angle more than 90 degrees with the negative example data vectors (x E N), we are cool. So ideally, it should look something like this:



So we now strongly believe that the angle between w and x should be less than 90 when x belongs to P class and the angle between them should be more than 90 when x belongs to N class.

Here's why the update works:
So when we are adding x to w, which we do when x belongs to P and w.x ¡ 0 (Case 1), we are essentially increasing the cos(alpha) value, which means, we are decreasing the alpha value, the angle between w and x, which is what we desire. And the similar intuition works for the case when x belongs to N and w.x $\geq$ 0 (Case 2).

3.2.3 Proof Of Convergence

Now, there is no reason for you to believe that this will definitely converge for all kinds of data-sets. It seems like there might be a case where the w keeps on moving around and never converges. But people have proved it that this algorithm converges. Attaching the proof, by Prof. Michael Collins of Columbia University — find the paper here.

3.2.4 Multiple neuron Multi-layer Perceptron

Multilayer feedforward neural networks are a special type of fully connected network with multiple single neurons. They are also called Multilayer Perceptrons (MLP). The following figure illustrates the concept of an MLP consisting of three layers:

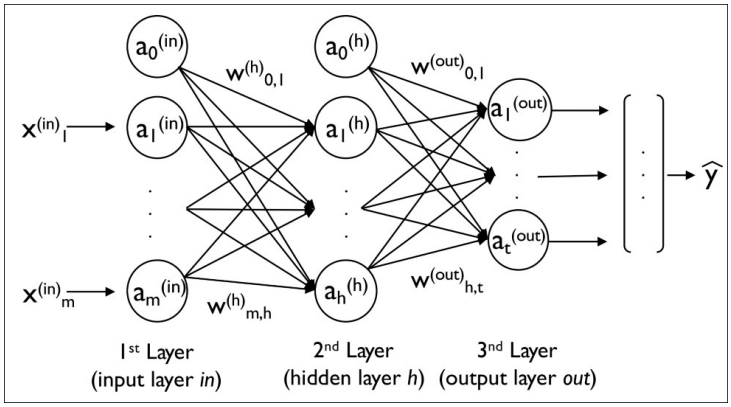


Figure 6: 3- layered MLP

The MLP depicted in the preceding figure has one input layer, one hidden layer, and one output layer. The units in the hidden layer are fully connected to the input layer, and the output layer is fully connected to the hidden layer. If such a network has more than one hidden layer, we also call it a deep artificial neural network.

We can add an arbitrary number of hidden layers to the MLP to create deeper network architectures. Practically, we can think of the number of layers and units in a neural network as additional hyper parameters that we want to optimize for a given problem task.

As shown in the preceding figure, we denote the i^{th} activation unit in the i^{th} layer as a_i^l.

To make the math and code implementations a bit more intuitive, we will use the in superscript for the input layer, the h superscript for the hidden layer, and the o superscript for the output layer.

For instance: a_i^{in} refers to the i^{th} value in the input layer, a_i^h refers to the i^{th} value in the hidden layer, and a_i^{out} refers to the i^{th} unit in the output layer. Here, the activation units a_0^{in} and a_0^{out} are the bias units, which we set equal to 1. The activation of the units in the input layer is just its input plus the bias unit

Each unit in layer l is connected to all units in layer l+1 via a weight coefficient. For example, the connection between the kth unit in layer l to the jth unit in layer l+1 will be written as $w_{k,j}^{(}l)$.

$$a^{(in)} = \begin{bmatrix} a_0^{(in)} \\ a_1^{(in)} \\ \vdots \\ a_m^{(in)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(in)} \\ \vdots \\ x_m^{(in)} \end{bmatrix}$$

Referring back to the previous figure, we denote the weight matrix that connects the input to the hidden layer as W^h , and we write the matrix that connects the hidden layer to the output layer as W^{out} .

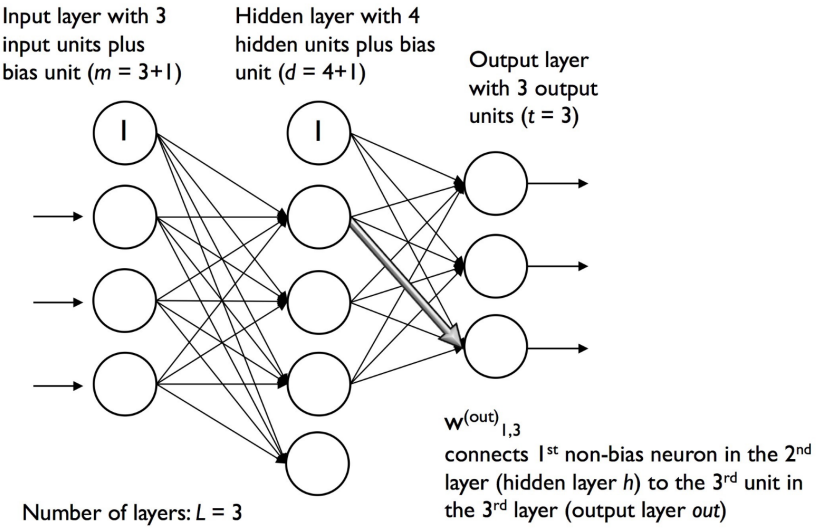


Figure 7: 3–4–3 Multilayer Perceptron

We summarize the weights that connect the input and hidden layers by a matrix: where d is the number of hidden units and m is the number of input units including the bias unit. Since it is important to internalize this notation to follow the concepts later in this tutorial, let's summarize what we have just learned in a descriptive illustration of a simplified 3–4–3 Multilayer Perceptron:

4 Activation function

Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

In coming section we will see variants of activation function.

4.1 Linear Function

1. Equation : Linear function has the equation similar to as of a straight line i.e. y = ax
2. No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
3. Range : -inf to +inf
4. Uses : Linear activation function is used at just one place i.e. output layer.
5. Issues : If we will differentiate linear function to bring non-linearity, result will no more depend on input "x" and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

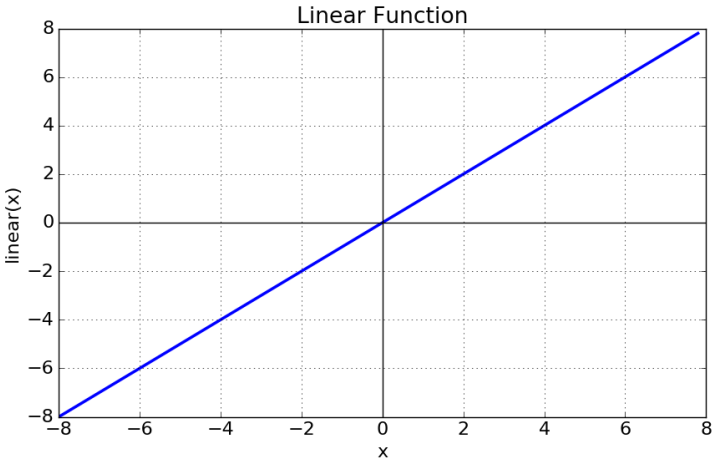


Figure 8: Linear Function

4.2 Sigmoid Function

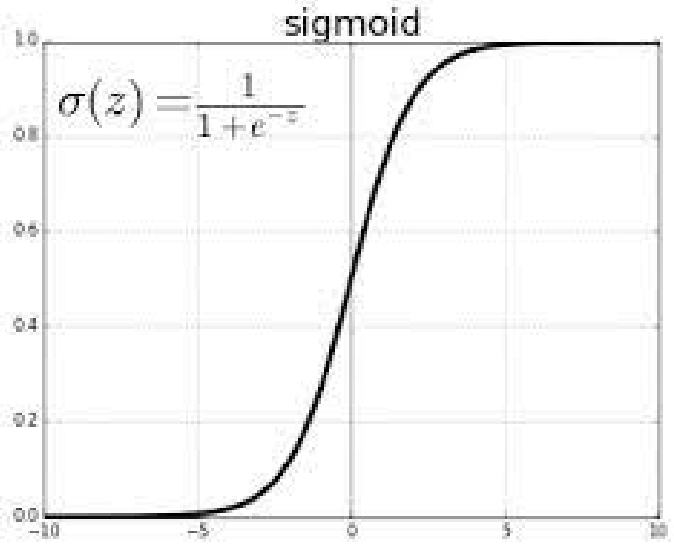


Figure 9: Sigmoid Function

1. It is a function which is plotted as 'S' shaped graph.

2. Equation : A = $1/(1 + e^{-x})$
3. Nature : Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
4. Value Range : 0 to 1
5. Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.

4.3 Tanh Function

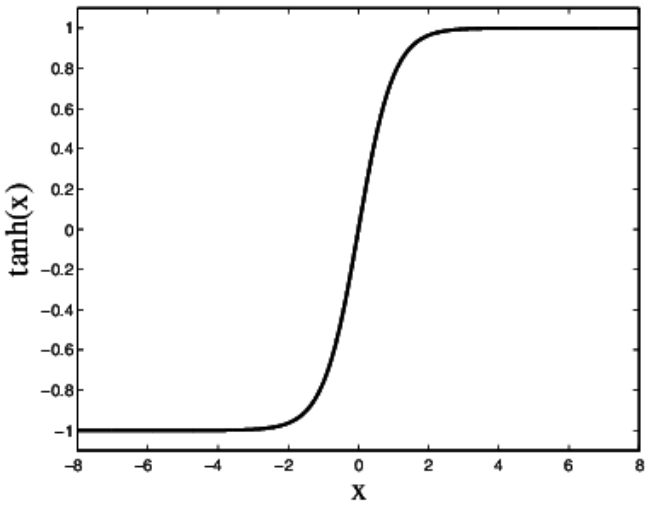


Figure 10: Tanh Function

The activation that works almost always better than sigmoid function is Tanh function also knows as Tangent Hyperbolic function. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.

1. Equation :-f(x) = tanh(x) = $2/(1 + e^{-2x})$ - 1
   OR
   tanh(x) = 2 * sigmoid(2x) - 1
2. Value Range :- -1 to +1 Nature :- non-linear
3. Uses :- Usually used in hidden layers of a neural network as it's values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.

4.4 ReLU

ReLU stands for Rectified linear unit. It is the most widely used activation function. Chiefly implemented in hidden layers of Neural network.

1. Equation :- A(x) = max(0,x). It gives an output x if x is positive and 0 otherwise.
2. Value Range :- [0, inf)

3. Nature :- non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
4. Uses :- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.
5. In simple words, RELU learns much faster than sigmoid and Tanh function.

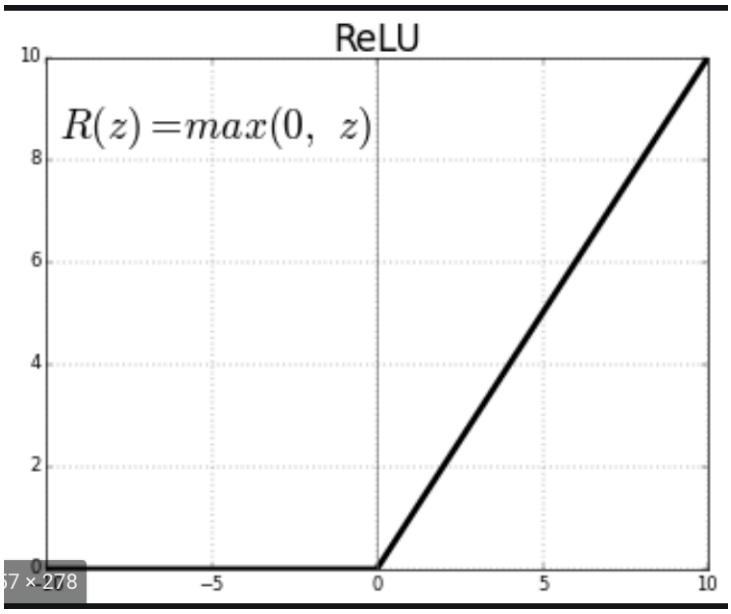


Figure 11: ReLU

4.5 Softmax Function

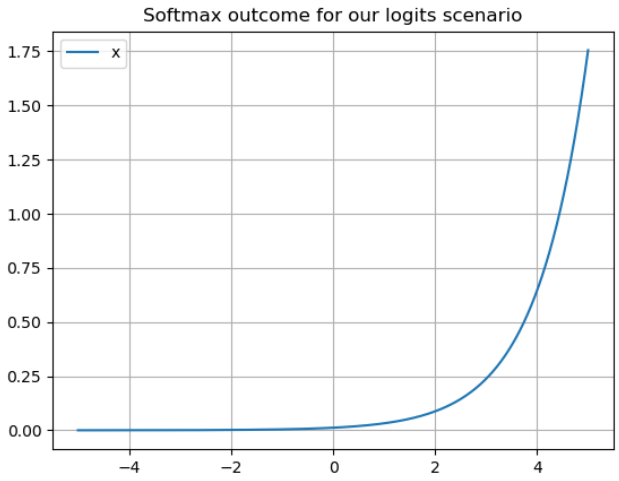


Figure 12: Softmax Function

The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems.

1. Nature :- non-linear

2. Uses :- Usually used when trying to handle multiple classes. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
3. Output:- The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

4.6 Choosing the right activation function

The basic rule of thumb is if you really don't know what activation function to use, then simply use RELU as it is a general activation function and is used in most cases these days.

If your output is for binary classification then, sigmoid function is very natural choice for output layer.

References

[1] Neural Networks: All You Need to Know

[2] Introduction to Artificial Neural Networks

[3] Understanding Neural Networks

[4] A Beginners Guide to Neural Networks

[5] Multilayer Neural Network

[6] Perceptron learning algorithm

[0]These notes are based on lectures taken by Dr. Vineet Gandhi.