CSE/ECE-471: Statistical Methods in AI **Spring 2020**

Multi Layer Perceptron

Upinder Singh(2019201083) *Diksha Singh(2019201066)* *Shanu Shrivastava(2019202005)*

1 Perceptron

The perceptron is the building block of neural network. It computes dot product of Weight vector with input and applies a step function to output. It is similiar to Logistic regression except it uses step function instead of sigmoid function at output. Constant 1 is introduced in the input layer to introduce a bias.

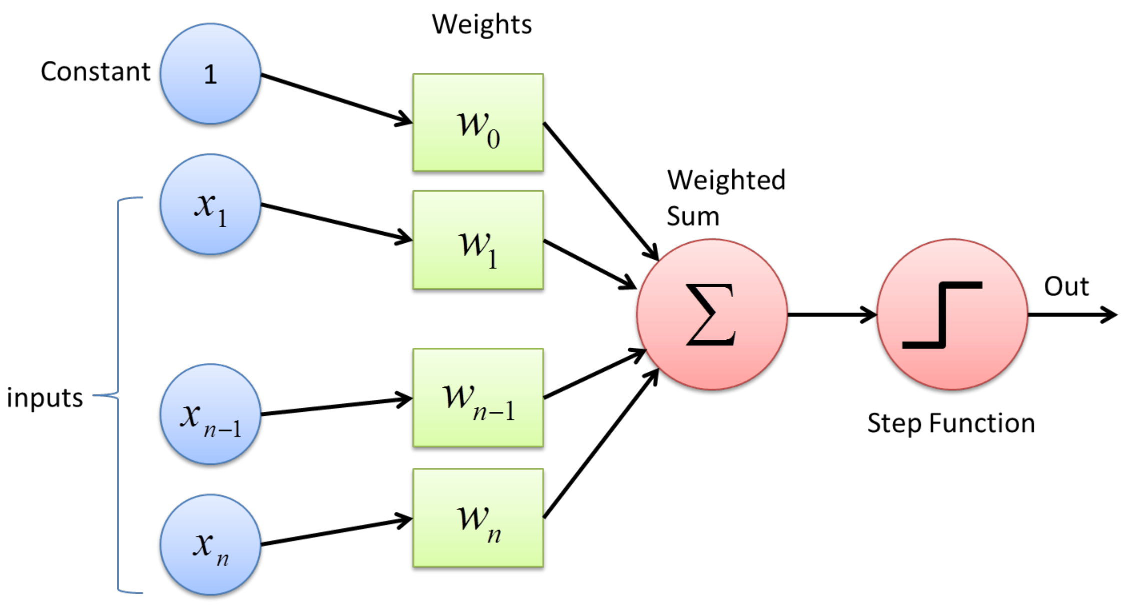


Fig-1: Basic Model

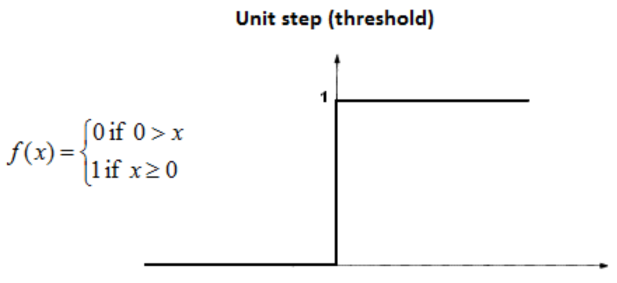


Fig: Unit Step Activation Function

2 Perceptron Learning

Once the weight are computed using $(W^T X + b)$, the weights are updated using below rule:

1. For correctly labelled inputs, don't update weight.
2. If a label is incorrectly classifed as negative label, add positive X, otherwise add -X.

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

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

$$\Delta W(k) = 0 \begin{Bmatrix} W(k)^T X(k) > 0\ \&\ \ y(k) = 1 \\ or\ \ W(k)^T X(k) < 0\ \&\ \ y(k) = 0 \end{Bmatrix}$$

Repeat untill convergence.

3 Multi Layer Perceptron

Problem with perceptron was that it wasn't sufficient to learn model for complex distributions.

Perceptron being too simplistic for real word scenarios, Multi layer perceptron was proposed to overcome the limitations of a single perceptron model. The idea was to stack multiple perceptron layers together to form a network of perceptrons, where each input was feed from output of previous layer perceptrons(except Input layer). Stacking layers could only be termed usefull if we introduce some non-linearity(otherwise, the output would be just some linear combination of input). Therfore, we added **activation functions** at the output of each perceptron instead of step function(we now call this a neuron).

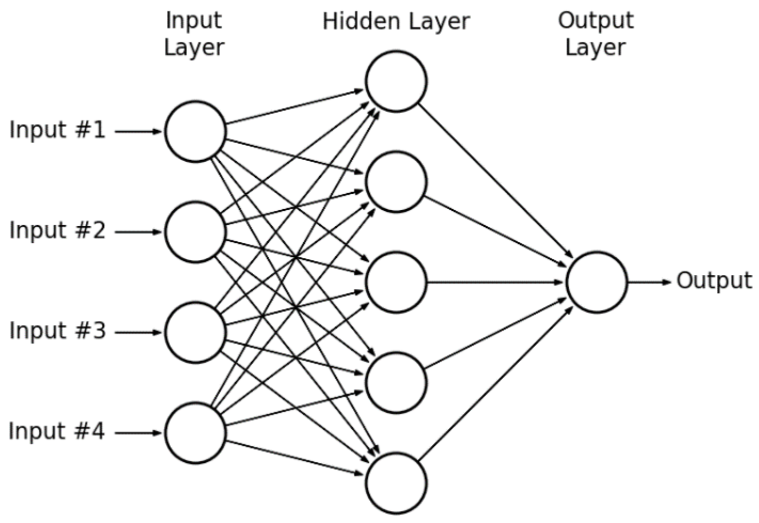


Fig-3: Multi-Layer Perceptron

Basic features of a multi layer perceptron:-

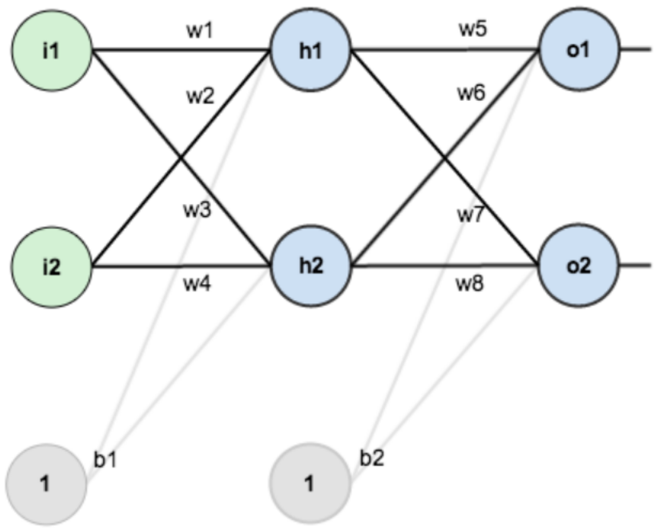
1. We require activation function to be differentiable with respect to weight.
2. The network contains one or more layers that are hidden from both input and output nodes.

Training of a Multi layer perceptron proceeds in two phases:

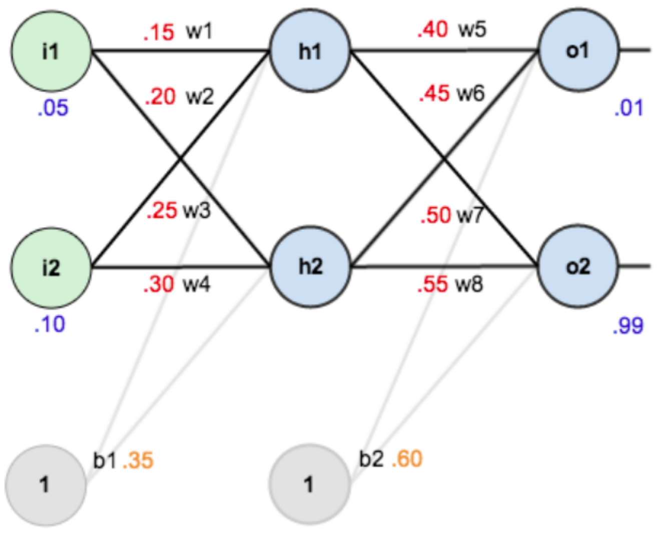
1. **Forward phase :** Weights of the network are fixed and the input signal is propagated through the network layer by layer until it reaches the output.
2. **Backward phase :** Error signal produced by comparing the output of the network and the desired response is propagated through the network layer by layer in backward direction.

3.1 Forward Pass

Consider the example below, with two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias.



In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



The goal is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs. given inputs $X = [0.05, 0.10]$, we want the neural network to output 0.01 and 0.99.

To begin, lets see what the neural network currently predicts given the weights and biases above and inputs of 0.05 and 0.10. To do this we'll feed those inputs forward though the network.

We figure out the total net input to each hidden layer neuron, squash the total net input using an activation function (here we use the logistic function), then repeat the process with the output layer neurons. Here's how we calculate the total net input for h_1:

$$net_{h1} = w_1 * i_1 + w_2 * i2 + b1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

We then squash it using the logistic function to get the output of (h_1):

$$out_{h1} = \frac{1}{1 + e^{-net_h 1}} = \frac{1}{1 + e^{-0.3775}} = 0.593269992$$

Carrying out the same process for h_2 we get:

$$out_{h2} = 0.596884378$$

We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs. Here's the output for o_1 : $net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-1.105905967}} = 0.75136507$$

And carrying out the same process for o_2 we get:

$$out_{o2} = 0.772928465$$

Calculating the Total Error

We can now calculate the error for each output neuron using the squared error function and sum them to get the total error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

For example, the target output for o_1 is 0.01 but the neural network output 0.75136507, therefore its error is:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Repeating this process for o_2 (remembering that the target is 0.99) we get:

$$E_{o2} = 0.023560026$$

The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

3.2 The Backwards Pass

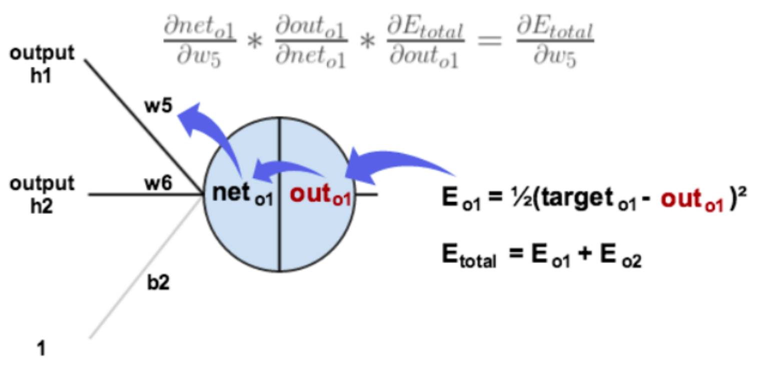
Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

Output Layer

Consider w_5. We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$. By applying the chain rule we know that:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Visually, here's what we're doing:


$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$
output h1
w5
output h2
w6
b2
1
net o1
out o1
E o1 = ½(target o1 - out o1)²
E total = E o1 + E o2

We need to figure out each piece in this equation.
First, how much does the total error change with respect to the output?

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

Note: When we take the partial derivative of the total error with respect to out_{o1}, the quantity $\frac{1}{2}(target_{o2} - out_{o2})^2$ becomes zero because out_{o1} does not affect it which means we're taking the derivative *of a constant which is zero.*

Next, how much does the output of o_1 change with respect to its total net input?

The partial derivative of the logistic function is the output multiplied by 1 minus the output:

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1-out_{o1}) = 0.75136507(1-0.75136507) = 0.186815602$$

Finally, how much does the total net input of o1 change with respect to w_5?

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

You'll often see this calculation combined in the form of the delta rule:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1-out_{o1}) * out_{h1}$$

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, eta, which we'll set to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

We can repeat this process to get the new weights w$_6, w_7, and w_8 : w_6^+ = 0.408666186$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

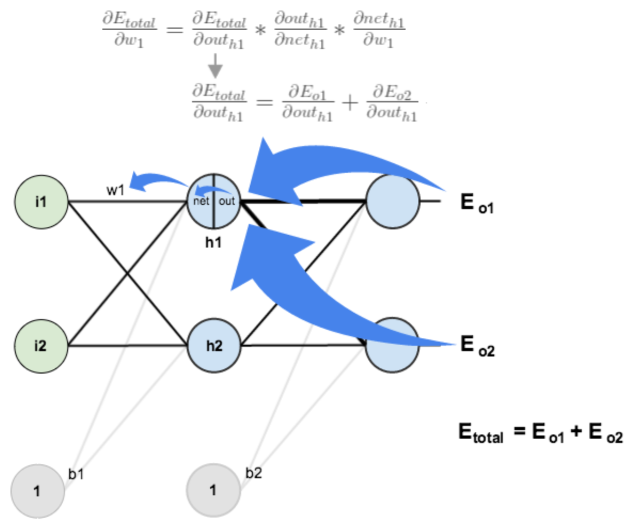
We perform the actual updates in the neural network after we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

Hidden Layer :

Next, we'll continue the backwards pass by calculating new values for w$_1, w_2, w_3, and w_4$.

Here's what we need to figure out:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$


$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$
$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$
i1
i2
w1
net
out
h1
h2
E o1
E o2
b1
b2
1
1
E total = E o1 + E o2

We're going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that out_{h1} affects both out_{o1} and out_{o2} therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to w$_5$:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

Plugging them in:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$, we get:

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

Therefore:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$, we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

We calculate the partial derivative of the total net input to h_1 with respect to w_1 the same as we did for the output neuron:

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

We can now update w_1 :

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Repeating this for w_2, w_3 and w_4

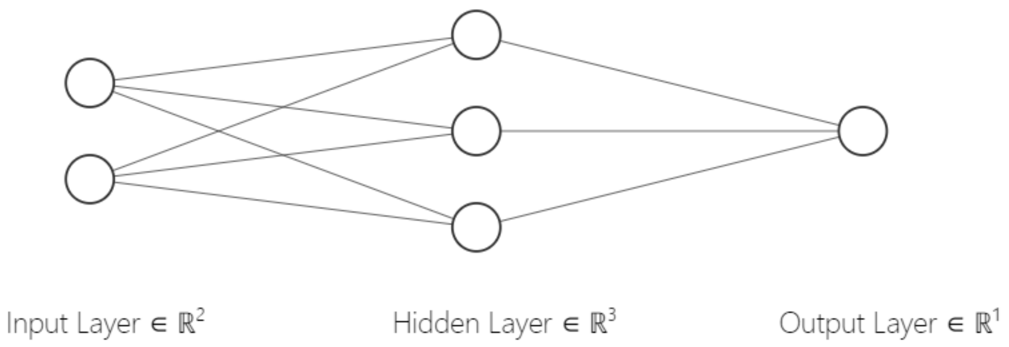
$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

Finally, we've updated all of our weights! When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109. After this first round of backpropagation, the total error is now down to 0.291027924. It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.0000351085. At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

4 Back Propagation


Input Layer ∈ $\mathbb{R}^2$
Hidden Layer ∈ $\mathbb{R}^3$
Output Layer ∈ $\mathbb{R}^1$

Let W_1, W_2 denote the weight matrix at Layer 1, 2 respectively.

Let $Z = W_1^T X$

$\hat{Y} = W_2^T Z$

Cost function:

$$\mathcal{L}(\hat{Y}, Y_i) = \frac{1}{2}\sum_{i=1}^{n}(\hat{y} - y_i)^2$$

Let $P = \phi(W_1^T X)$

Updation rule

$$W_1 = W_1 - \alpha\frac{\partial \mathcal{L}}{\partial W_1}$$
$$W_2 = W_2 - \alpha\frac{\partial \mathcal{L}}{\partial W_2}$$

Computing Grad of Cost function wrt W_2

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{1}{2}\frac{\partial\left(W_2^T P - Y\right)^2}{\partial W_2}$$
$$\frac{\partial \mathcal{L}}{\partial W_2} = (W_2^T P - Y)P$$

Computing Grad of Cost function wrt W_2

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial \hat{Y}}\frac{\partial \hat{Y}}{\partial P}\frac{\partial P}{\partial Z}\frac{\partial Z}{\partial W_1}$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = (\hat{Y} - Y)W_2^T \phi(W_1^T X)X$$

5 Activation Functions

The main function of activation function is to convert a input signal to output signal.If we do not apply a Activation function then the output signal would simply be a simple linear function.

Some popular activation functions:-

1. Sigmoid function:-

$$f(x) = \frac{1}{1+e^{-x}} \tag{1}$$

Diffrential of sigmoid function is given by f'(x)=f(x)(1-f(x))

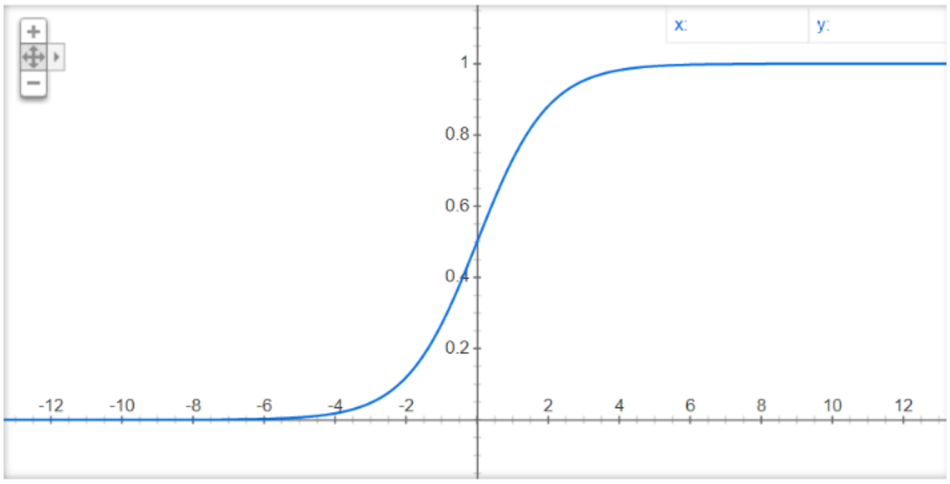
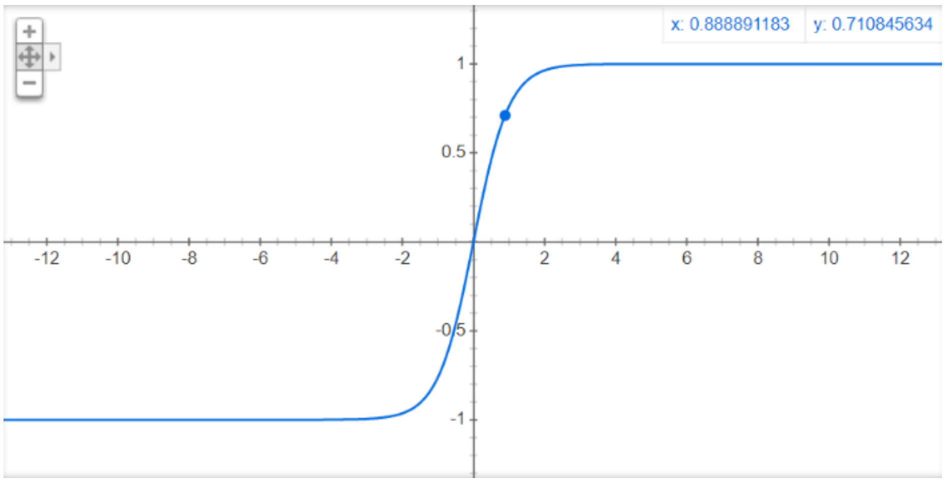


Figure 1: Sigmoid function

2. tanh

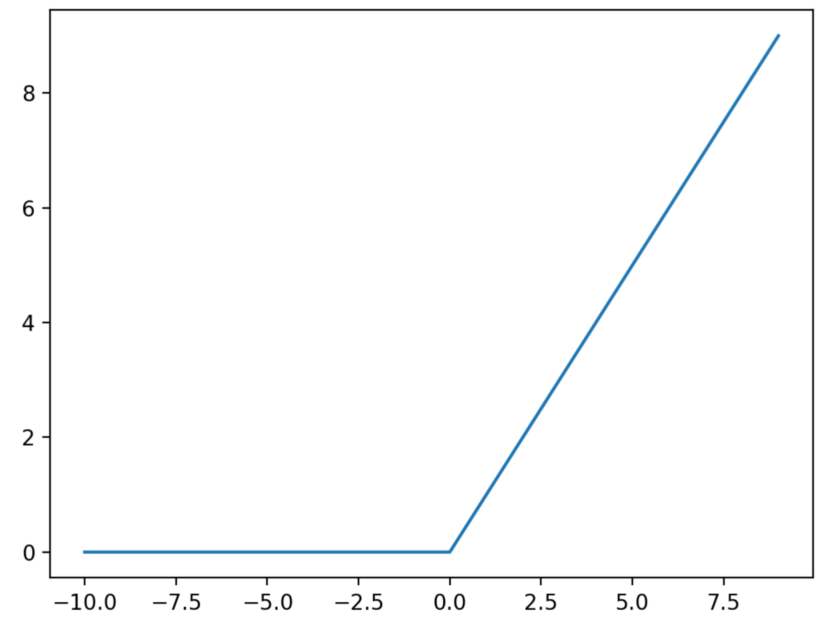
$$f(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2}$$

Diffrential of sigmoid function is given by f'(x)=1-f(x)^2



3. ReLU

$$f(x) = \max(0, x) \tag{3}$$


8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5