Module 1:

Supervised Learning Networks Feedforward DNN

1.1 Perceptron: Representational power of Perceptron

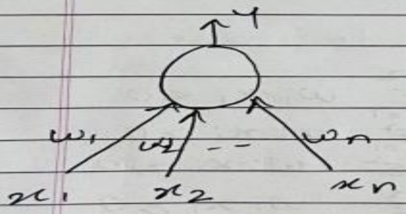
③ Perceptron -

Frank Rosenblatt proposed perception in 1958.

① It is more general computational model than MP neuron.

② Main difference is introduction of numerical weights and mechanism for learning those wts.

③ Inputs are no longer limited to boolean values



$$y = 1 \text{ if } \sum_{i=1}^{n} w_i x_i \geq \theta$$

$$= 0 \text{ if } \sum_{i=1}^{n} w_i x_i < \theta$$

Rewriting the eq^n.

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

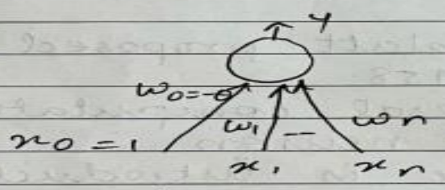
$$= 0 \text{ if } \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

Rewriting,

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

$$= 0 \text{ if } \sum_{i=0}^{n} w_i * x_i \leq 0$$

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



w_0 is call bias as it represents the prior (prejudice)

Example -

OR function -

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

① $w_0 + \sum_{i=1}^{2} w_i x_i < 0$

$= w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0$

$= w_0 < 0$

② $w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$= w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0$

$= w_0 + w_2 \geq 0$

$= w_2 > -w_0$

③ $w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$= w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0$

$= w_0 + w_1 \geq 0$

$= w_1 > -w_0$

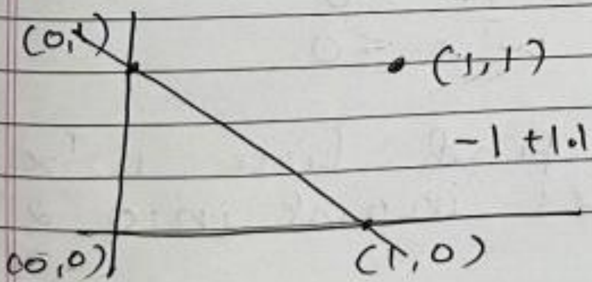
④ $w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$= w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0$

$= w_0 + w_1 + w_2 \geq 0$

$= w_1 + w_2 > -w_0$

⑤ One possible solution
$w_0 = -1, \quad w_1 = 1.1, \quad w_2 = 1.1$
(other solutions possible)



$-1 + 1.1x_1 + 1.1x_2 = 0$

$1.1x_1 + 1.1x_2 = 1$

($\because$ line will pass thr 1)

1.2 The Perceptron Training Rule

4. Perceptron Algorithm-

Perception learning Algorithm-

$P \leftarrow$ inputs with label 1
$N \leftarrow$ inputs with label 0
Initialize $w = [w_0, w_1 \,--\, w_n]$ randomly

while ! convergence do
    pick random $x \in P \cup N$
    if $x \in P$ and $\sum_{i=0}^{n} w_i x_i < 0$ then
        $w = w + x$

    end
    if $x \in N$ and $\sum_{i=0}^{n} w_i x_i \geq 0$ then
        $w = w - x$
    end
end

Explanation

consider 2 vectors w and x

$$w = [w_0, w_1, w_2 \,---\, w_n]$$

$$x = [1, x_1, x_2 \,---\, x_n]$$

$$w \cdot x = w^T x = \sum_{i=0}^{n} w_i * x_i$$

Thus we can write perception Rule as,

$$y = 1 \text{ if } w^T x \geq 0$$
$$= 0 \text{ if } w^T x < 0$$

We need to find line $w^T x = 0$ which divides input into 2 halves

② Every point x on line satisfies equation $w^T x = 0$

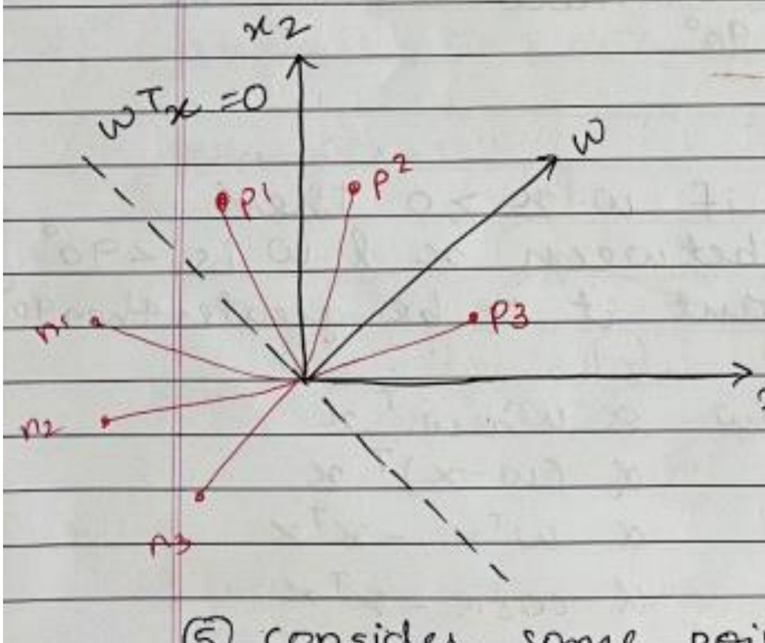
ex- $x_1 + x_2 = 0$
lets consider $x_1 = 1, x_2 = -1$
$1 - 1 = 0$, but 2, 2 cannot be pts on line

③ Angle α between w and any pt x lies on line is 90°

$$\because \cos\alpha = \frac{w^T x}{\|w\|\|x\|} = 0$$

$$\therefore \alpha = 90^\circ$$

④ Since vector w is perpendicular to every pt ~~or~~ on line it is perpendicular to itself



⑤ consider some points which lie on positive half space of line ($P_1 P_2 P_3$)
⑥ Angle between points and w is less than 90°
⑦ consider some points which lie on negative half space of line ($n_1 n_2 n_3$)

⑧ angle between n_1, n_2, n_3 and w is greater than 90°

⑨ Now from the perception Alg for $x \in P$ if $w \cdot x < 0$ then it means that α between x & w is greater than 90° (but we want it to be $< 90^\circ$)

solution-

α_{new} is new angle

$$\cos \alpha_{new} \propto w_{new}^T x$$
$$\propto (w + x)^T x$$
$$\propto w^T x + x^T x$$
$$\propto \cos\alpha + x^T x$$
$$\cos\alpha_{new} > \cos\alpha$$

that is $\cos\alpha_{new} > \cos\alpha$ so

$\alpha_{new} < 90^\circ$

⑩ For $x \in N$ if $w \cdot x \geq 0$ then angle α between x & w is $< 90^\circ$ but we want it to be greater than 90°

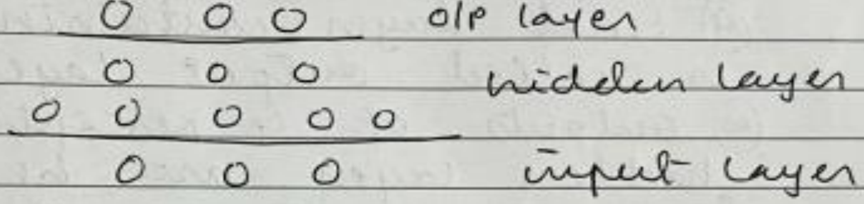
solution -

$$\cos \alpha_{new} \propto w_{new}^T x$$
$$\propto (w - x)^T x$$
$$\propto w^T x - x^T x$$
$$\propto \cos\alpha - x^T x$$
$$\cos\alpha_{new} < \cos\alpha$$
$$\alpha_{new} > 90^\circ$$

1.3 Multilayer Perceptron:

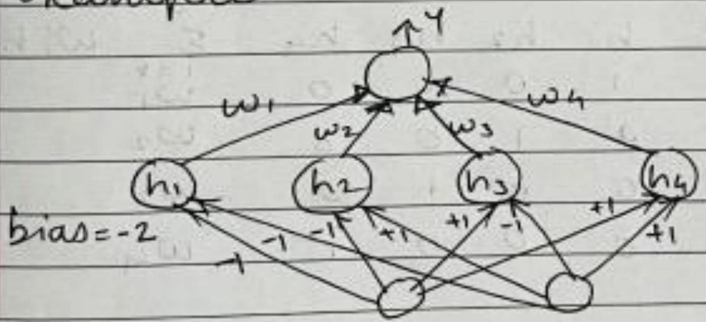
5. Multilayer Perception

① A multilayer perception is feed forward neural network that generates a set of outputs from set of inputs
Multilayer perception has 3 segments
① Input layer
② Hidden layer
③ output layer



② It is called FFNN because data moves in single direction from input to output layer.

Example-



① Lets consider True = +1 false = -1

② we consider 2 inputs & 4 perception
③ Each Input is connected to all the 4 perception with specific weights
④ the bias w_0 of each perception

is -2 (each perception will fire only if weighted sum of its input is ≥ 2)
⑤ Each of these perception is connected to an output perception by weights
⑥ The output of this perception (4) is output of this network.
⑦ $x_1\ x_2$ is input layer
⑧ Middle layer containing 4 perception is called hidden layer
⑨ Final layer containing 1 neuron is called output layer.
⑩ outputs of 4 perception in hidden layer are h1 h2 h3 h4
⑪ $w_1\ w_2\ w_3\ w_4$ are layer 2 wts.
⑫ Each perception in hidden layer fires only for specific input (and no 2 perception fire for same input)

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

~~This~~ let w_0 bias output of neuron i.e it will fire only if $\sum_{i=1}^{4} w_i h_i \geq w_0$

This results in 4 conditions

$w_1 < w_0$ (since XOR o/p is 0)
$w_2 \geq w_0$ " " 1
$w_3 \geq w_0$ " " "
$w_4 < w_0$ " " 0

⑬ so each w_i is now responsible for one of the 4 possible inputs and can be adjusted to get the desired output for that input.

Delta Training Rule

The Generalized Delta Rule is an algorithm used to update the weights in an artificial neural network during training. The rule is based on the concept of calculating the gradient of the loss function with respect to each weight in the network and adjusting the weights in the opposite direction of the gradient, hence "backpropagating" the error from the output layer back to the input layer. The **Delta Rule** uses the difference between *target activation* (i.e., target output values) and *obtained activation* to drive learning.

Let $o=w_0+w_1x_1+w_2x_2+\text{----}+w_nx_n$

Hence o(x)=w.x

o is output

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

E is training error

Derivative of error is

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

Training rule for gradient descent is,

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

Where

$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$

Where eta is the learning rate which determines the step size, negative sign indicates we are moving in direction that decreases E

$$w_i \leftarrow w_i + \Delta w_i$$

Where

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\
\frac{\partial E}{\partial w_i} &= \sum_{d \in D} (t_d - o_d)(-x_{id})
\end{aligned}$$

Where

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

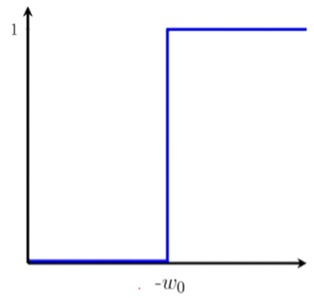
$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\ x_{id}$$

$$w_i \leftarrow w_i + \Delta w_i$$

1.4 Multilayer Networks:

Differentiable Threshold Unit (Sigmoid Neurons)

Perceptron:

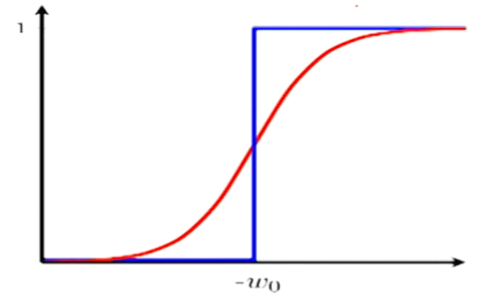


Thresholding logic used by a perceptron is very harsh, it is a characteristic of the perceptron function itself which behaves like a step function

there is always this sudden change in the decision from 0 to 1 when $z = \sum_{i=1}^{n} w_i x_i$ crosses the threshold

Perceptron is not smooth not continuous not differentiable.

Sigmoid Neuron



$$z = \sum_{i=1}^{n} w_i x_i$$

In Sigmoid neurons, the output function is much smoother than the step function

Output is no longer binary but a real value between 0 and 1 which can be interpreted as probability

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}$$

More precisely, the sigmoid unit computes its output o as

$$o = \sigma(\vec{w} \cdot \vec{x})$$

Where

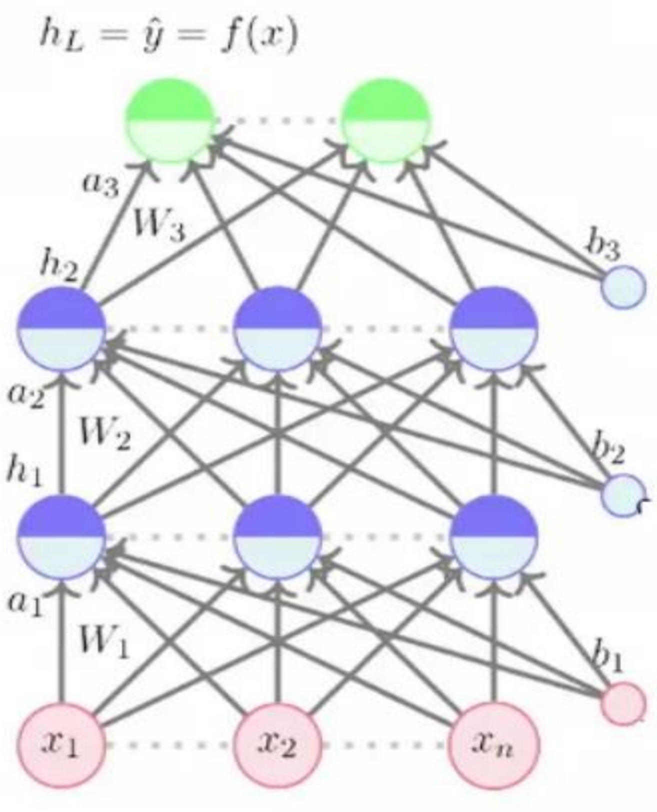
$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

σ is often called the ***sigmoid function*** or, alternatively, the ***logistic function***. Its output ranges between 0 and 1, increasing monotonically with its input.

It maps a very large input domain to a small range of outputs, it is often referred to as the s***quashing function of*** the unit. The sigmoid function has the useful property that its derivative is easily expressed in terms of its output.

The function ***tanh*** is also sometimes used in place of the *sigmoid function*

1.5 Representational Power of Feedforward Networks

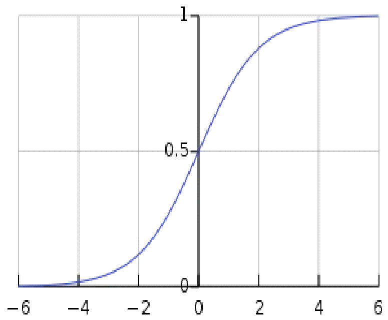


1. Input to the network is n-dimensional vector
2. Network contains L-1 hidden layer having n neurons each
3. One output layer containing k neurons
4. Each neuron in hidden and output layer is divided into two parts: pre activation and activation.
5. Input layer is 0^{th} layer output layer is Lth layer
    - Pre activation at layer i is,
      $a_i(x)=b_i+w_ih_{i-1}(x)$
    - activation at layer i is given by
      $h_i(x)=g(a_i(x))$
      g is activation function like logistic, sigmoid etc
    - activation at layer L is given by
      $f(x)=h_L=o(a_L)$
      where o is output activation function like softmax

1.6 Activation functions: Tanh, Logistic, Linear, Softmax, ReLU, Leaky ReLU,

1. Sigmoid / Logistic

Sigmoid function gives an ‘S’ shaped curve. In order to map predicted values to probabilities, we use the sigmoid function. The function maps any real value into another value between 0 and 1.



Sigmoid function

- **Equation:** $f(x) = s = 1/(1+e^{-x})$
- **Derivative:** $f'(x) = s*(1-s)$
- **Range:** (0,1)

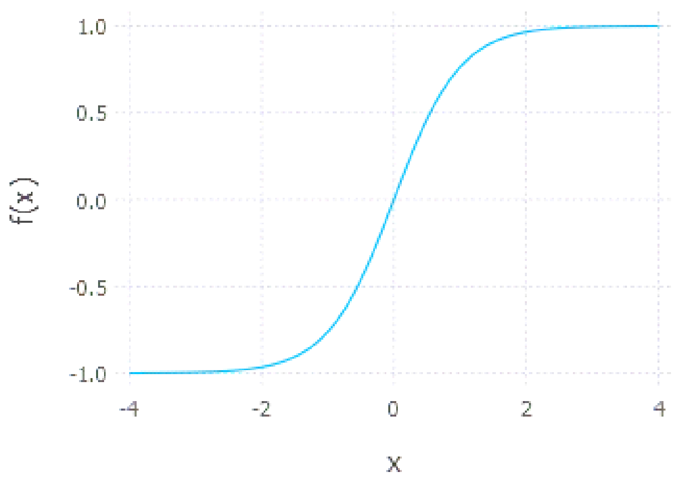
Advantages:

- The function is differentiable.That means, we can find the slope of the sigmoid curve at any two points.
- Output values bound between 0 and 1, normalizing the output of each neuron.

Disadvantages:

- Vanishing gradient — for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem.
- Due to vanishing gradient problem, sigmoids have slow convergence.
- Outputs not zero centered.
- Computationally expensive.

2.Tanh



Tan-h function

- **Equation :** $f(x) = a = \tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})$

- **Derivative:** $(1 - a^2)$
- **Range:** (-1, 1)

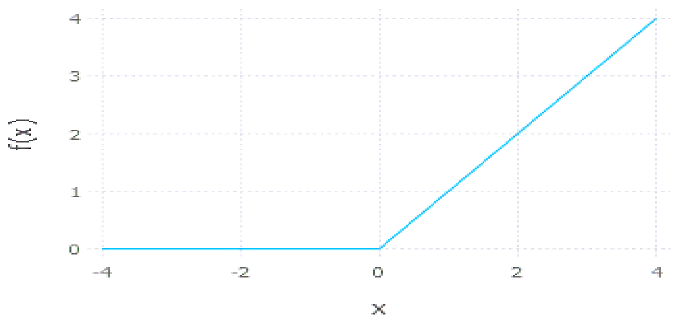
Advantages:

- Zero centered — making it easier to model inputs that have strongly negative, neutral, and strongly positive values.
- The function and its derivative both are monotonic.
- Works better than sigmoid function

Disadvantage:

- It also suffers vanishing gradient problem and hence slow convergence.

3. **ReLU (Rectified Linear Unit)**



ReLU function

- **Equation:** f(x) = a =max(0,x)
- **Derivative:** f'(x) = { 1 ; if z>0, 0; if z<0 and undefined if z=0 }
- **Range:** (0, +∞)

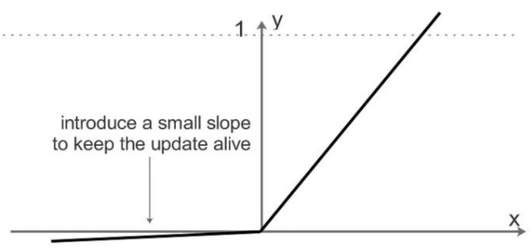
Advantages:

- Computationally efficient — allows the network to converge very quickly
- Non-linear — although it looks like a linear function, ReLU has a derivative function and allows for back-propagation

Disadvantages:

- The Dying ReLU problem — when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform back-propagation and cannot learn.

4. **Leaky ReLU**



- **Equation:** f(x)= a = max(0.01x, x)

- **Derivative:** f'(x) = {0.01 ; if z<0, 1 ; otherwise}
- **Range:** (0.01, +∞)

Advantage:

- Prevents dying ReLU problem — this variation of ReLU has a small positive slope in the negative area, so it does enable back-propagation, even for negative input values

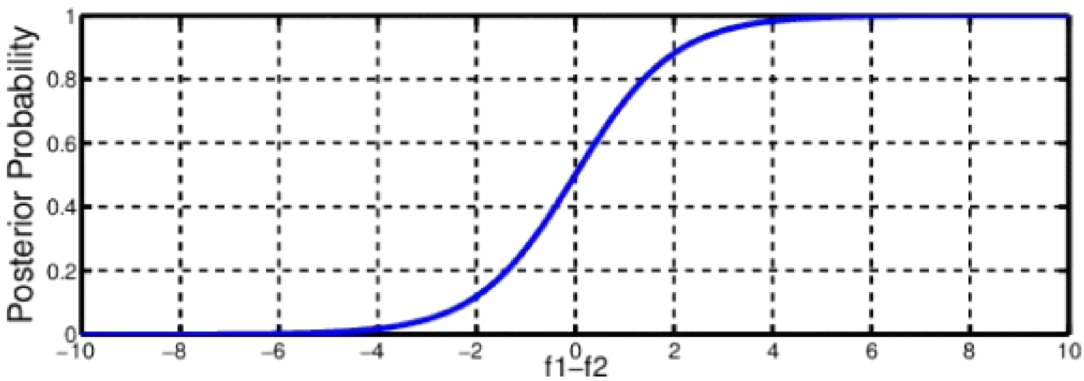
Disadvantage:

- **Results not consistent** — leaky ReLU does not provide consistent predictions for negative input values.
- During the front propagation if the learning rate is set very high it will overshoot killing the neuron.

The idea of leaky ReLU can be extended even further. Instead of multiplying x with a constant term we can multiply it with a hyper-parameter which seems to work better the leaky ReLU. This extension to leaky ReLU is known as **Parametric ReLU**.

5. Softmax

Softmax function calculates the probabilities distribution of the event over 'n' different events.



- **Equation:** f(x) = $e^{x}{}_{i}$ / ($\Sigma_{j=0}$ $e^{x}{}_{i}$)
- **Probabilistic interpretation:** S_j = P(y=j|x)
- **Range:** (0, 1)

Advantages:

- Able to handle multiple classes only one class in other activation functions — normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class.
- Useful for output neurons — typically Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple categories.

1.7 Loss functions: Squared Error loss, Cross Entropy

Loss function is also known as cost function, its a mathematical function which is used to evaluate how well algorithm is modelling the dataset

MSE is the most common loss function for regression problems. It calculates the average squared difference between predicted and actual values:

$$E = \frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K - t_j\right)^2$$

- Error Magnification: Squaring errors penalizes larger errors more heavily, making it sensitive to outliers.

$$E = \frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K - t_j\right)^2$$

$$W_{ij}^K \leftarrow W_{ij}^K - \eta\delta_j^K O_i^{K-1}$$

$$\delta_j^K = O_j^K(1 - O_j^K)(O_j^K - t_j)$$

O_j^K is the output of the jth node in the kth layer, delta $_j^K$ was derived from the derivative of the sif I consider I have a single neuron at the output say it is a two class problem

Let us consider a case that weight vector X, which actually belongs to class 1 So, when it is classified by this classifier the output of the neuron should be 1 or close to 1. If the output of the neuron is not 1 or say it is very close to 0, that means, it has an error. Consider y should be equal to 1 but value of y we are getting here is equal to 0 or near to 0 and that comes output product O_j^K *(1- O_j^K) becomes very very low. Similarly, in the other case, if a training vector is given as belonging to 0, but classified that to class1, that means, output of the neuron should actually be 0. But the classifier has given a very high output And here you find that O_j^K as decided by the classifier being very high, 1- O_j^K will be very low. And that is because when output is very low then sigmoidal function, derivative of the sigmoidal function that is very very low and in the extreme case, it may even vanish. So, the gradient vanishes. And if the gradient vanishes or the gradient is very very low, that this gradient is directly influencing the rate of training, because rate of training is controlled by not only the convergence rate eta, it is also controlled by delta j K. So, if O_j^K*(1-O_j^K) whether O_j^K is 0 or O_j^K should be 1, whatever the case may be, if any of the terms is very low, then your rate of learning becomes very very low. So, that is bad effect of this quadratic loss function that we have.

When to Use MSE

- Regression problems where large errors are unacceptable.
- Scenarios where you want to emphasize minimizing large deviations.

Mean Absolute Error (MAE)

MAE computes the average of the absolute differences between predicted and actual values:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i|$$

- Equal Weightage: MAE treats all errors equally, making it robust to outliers compared to MSE.
- Non-Differentiable at Zero: Its absolute value introduces a point of non-differentiability at 0, which optimizers handle using sub-gradients.

When to Use MAE

- Regression problems with noisy data or outliers.
- Situations where equal error treatment is desired.

Cross-Entropy Loss

Cross-Entropy is the go-to loss function for classification tasks. It measures the dissimilarity between predicted probabilities and actual labels
Consider a two class problem, if a feature vector X, input training vectors are given as ordered pairs x, y where x is the input vector and y is the ground truth that is the actual class to which this vector X belongs

If y is actually equal to 1 that means, we get our training vector from class one for which output should be equal to 1. Whereas output of your neural network is o, so whatever is the output, this output actually gives you the likelihood that y is 1.

In the same way if y is equal to 0, that means, the training vector belongs to another class, then 1- o, where o is the output of the neuron that gives you the likelihood that y is 0.

o=likelihood that y is 1

(1-o)=likelihood that y is 0

Likelihood that is to be maximized=$o^y(1-o)^{(1-y)}$

Log likelihood=ylogo+(1-y)log(1-o)

$$\text{Minimize} \Rightarrow C = -\frac{1}{N}\sum_{\forall X}[y\log o + (1-y)\log(1-o)]$$

$$\frac{\partial C}{\partial W_i} = -\frac{1}{N}\sum_{\forall X}\left[\frac{y}{\sigma(\theta)} - \frac{(1-y)}{1-\sigma(\theta)}\right]\frac{\partial \sigma(\theta)}{\partial W_i}$$

$$= -\frac{1}{N}\sum_{\forall X}\left[\frac{y}{\sigma(\theta)} - \frac{(1-y)}{1-\sigma(\theta)}\right]\frac{\partial \sigma(\theta)}{\partial \theta}\cdot\frac{\partial \theta}{\partial W_i}$$

$$\frac{\partial C}{\partial W_i} = -\frac{1}{N}\sum_{\forall X}\left[\frac{y}{\sigma(\theta)} - \frac{(1-y)}{1-\sigma(\theta)}\right]\frac{\partial \sigma(\theta)}{\partial \theta}.\frac{\partial \theta}{\partial W_i}$$

$$= -\frac{1}{N}\sum_{\forall X}\left[\frac{y}{\sigma(\theta)} - \frac{(1-y)}{1-\sigma(\theta)}\right]\frac{\partial \sigma(\theta)}{\partial \theta}.\frac{\partial \theta}{\partial W_i}$$

$$= -\frac{1}{N}\sum_{\forall X}\left[\frac{y-\sigma(\theta)}{\sigma(\theta)(1-\sigma(\theta))}\right]\sigma(\theta)(1-\sigma(\theta).x_i$$

$$= \frac{1}{N}\sum_{\forall X} x_i(\sigma(\theta)-y) \qquad \boxed{= \frac{1}{N}\sum_{\forall X} x_i(o-y)}$$

$$C = -\frac{1}{N}\sum_{\forall X}\sum_{j}\left[y_j \log o_j^K + (1-y_i)\log(1-o_j^K)\right]$$

$$\frac{\partial C}{\partial W_{ij}^K} = \frac{1}{N}\sum_{\forall X} o_i^{K-1}(o_j^K - y_j)$$

$$W_{ij}^K \leftarrow W_{ij}^K - \eta\frac{1}{N}\sum_{\forall X} o_i^{K-1}(o_j^K - y_j)$$

When to Use Cross-Entropy

- Binary or multi-class classification problems.
- Tasks where the model outputs probabilities.

1.8 Choosing output function and loss function

Output Function:

1. If the binary output is expected from neural network, sigmoid is a default choice as activation function

2. If the network has multiclass output softmax is the default choice for output layer

3. For hidden layers, either tanh or ReLu can be used as activation function

Loss Function-

1. For classification problems cross entropy is used as loss function. In regression MSE, MAE is used as loss function