Activation function:

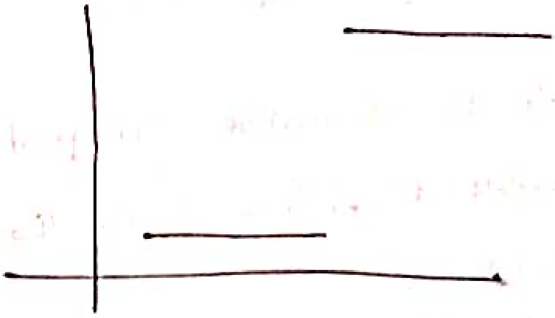
=> It defines the output of a neuron/node given an i/p or set of input (o/p of multiple neurons).

=> It's the mimic of the stimulation of a biological neuron.

=> The o/p of the activation function to the next layer (in shallow neural network: i/p layer and o/p layer, and in deep network to the next hidden layer) is called forward propagation (information propagation).

List :

- => binary
- => linear
- => sigmoid
- => Tanh
- => ReLU
- => Leakly ReLU (LReLU)
- => Parametric ReLU (PReLU)
- => Exponential Linear Unit (eLU)
- => ReLU-6
- => softplus
- => softsign
- => soft max
- => swish.



binary

Binary:

=> The binary activation function is the simplest.

=> It's based on binary classifier, the o/p is 0 if values are negatives else 1.

=> this activation function as a threshold in binary classification.

adv:

⇒ binary classification.

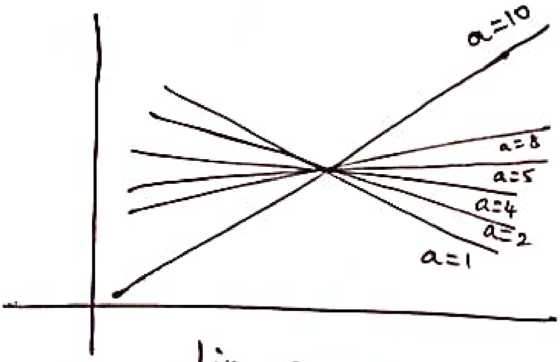
disadv :

⇒ doesn't work in multilabel classification.

⇒ The derivative for the gradient cla calculation is always 0 so it impossible to update weights.

Linear :.

In linear, the o/p is proportional to the i/p.



Linear.

adv :.

⇒ binary & multiclass classification

⇒ highly interpretable

dis :.

⇒ the derivative correspond to 'a' so the update of weights & biaises during the backpropagation will be constant.

⇒ not efficient if the gradient is always the same.

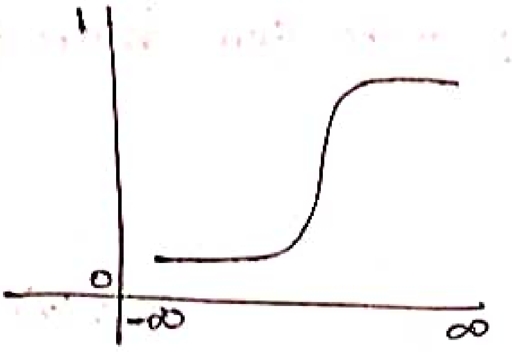
Sigmoid :.

⇒ It is the most used activation function with ReLU and tanh.

⇒ It is a non-linear activation function also called logistic function.

⇒ the o/p varies b/w 0 and 1.

⇒ the o/p of neurons are positive.



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

sigmoidal function.

adv:

⇒ the problem of this function is that the o/p of each neuron can saturate.

⇒ values greater than 1 are shape as 1 4 values smaller than 0 are shape as 0.

⇒ the best sensitivity for the sigmoid function is around the central point (0, 0.5).

⇒ the algorithm can not learn during this position (i.e the source of the vanishing gradient problem, corresponding to the absence of direction in the gradient) during saturation.

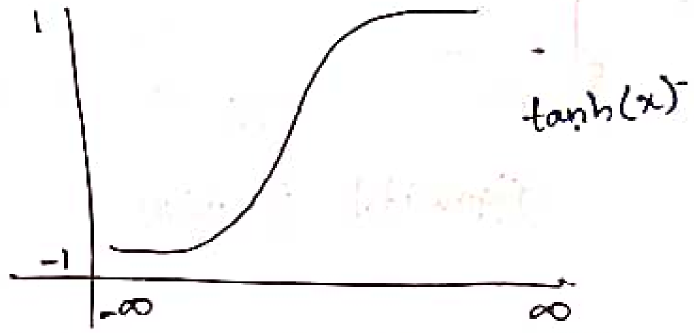
Tanh:

⇒ the tangent hyperbolic function (tanh) is similar to the sigmoid function in the way that their form are similar.

⇒ Tanh is symmetric in 0 4 the values are in the range of -1 & 1.

⇒ As the sigmoid they are very sensitive in the central point (0,0) but they saturate for very large number (+ve & -ve).

⇒ this symmetry make them better than the sigmoid function.



[illegible]:

dis :

~~⇒ the o/p of each neuron can be saturate.~~

~~⇒ Values greater than 1 are shape as 1 & values smaller than 0 are shape as 0.~~

~~⇒ the best sensitivity for the sigmoid functio is around the central point (0,0.5)~~

~~⇒ during saturation, the algorithm cannot learn during this po~~

Pros :

⇒ range b/w -1 & 1

⇒ the gradient is stronger than sigmoid (derivatives are steeper).

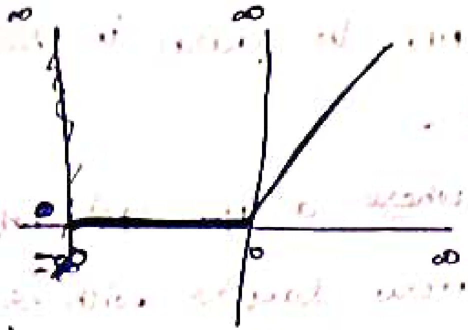
Cons :-

→ Like sigmoid, tanh also has a vanishing gradient problem.

→ Saturation.

ReLU :

Rectified linear activation function or ReLU for short is a piecewise linear function that will output the linear i/p directly if it is positive. otherwise, it will output zero.



ReLU function in its derivative both are monotonic.

Adv :

→ The ReLU function is another non-linear activation function that has gained populaority in the deep learning domain. ReLU stands for rectified linear unit.

→ the main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.

dis :

→ non-differentiable at zero and ReLU is at unbounded. The gradients for negative input is zero, which means for activations in that region, the weights are not updated during propagation. This can create dead neurons that never get activated.

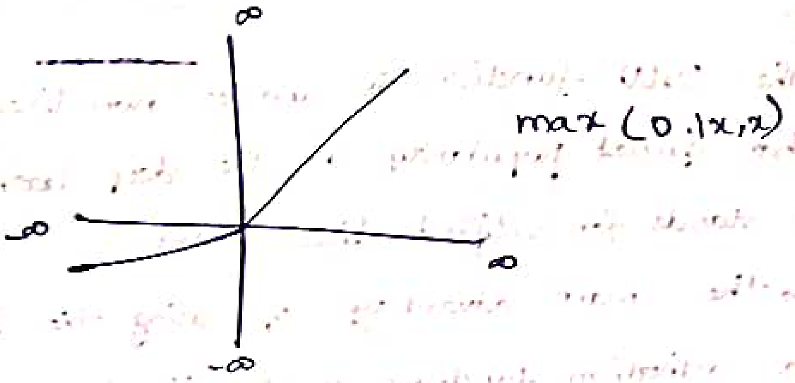
LReLU :-

⇒ The leakly ReLU modifies the function to allow small negative values when the i/p is zero.

⇒ the leakly rectifier allows for a small, non-zero gradient when the unit is saturated and non-active.

⇒ The leakly ReLU shows at least the same or better results in most comparisons with ReLU, but moreover, it allows NN to learn in setups (architecture where the ReLU fails.

Eg :- Its the case where a NN architecture contains "bottlenecks" - very narrow layers with ~~fair~~ small neurons count.



PReLU:

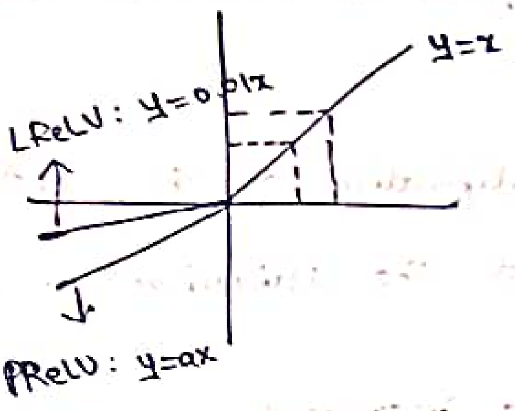
PReLU stands for Parametric ReLU is a type of leakly ReLU that, instead of having a predetermined slope like 0.01, makes it a parameter for the neural network to figure out itself:

$y = ax$ when $x<0$:

the leakly ReLU has 2 benefits :-

⇒ It fixes the "dying ReLU" problem. as it doesn't have zero slope parts.

⇒ It speeds up training.



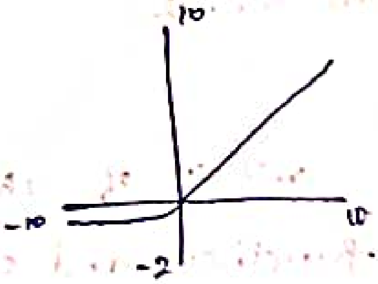
adv :-

It has the learning parameta function which fine-tunes the activation function based on its learning rate (unlike zero in the case of ReLU and 0.01 in the case of LReLU).

ELU :-

⇒ It stands for exponential linear unit.

⇒ In contrast to ReLU, ELUs have negative values which allows them to push mean unit activations closer to zero like batch normalization but with lower computational complexity.



$$\begin{cases} x & , x \geq 0 \\ \alpha(e^{x}-1) & , x < 0 \end{cases}$$

Adv :-

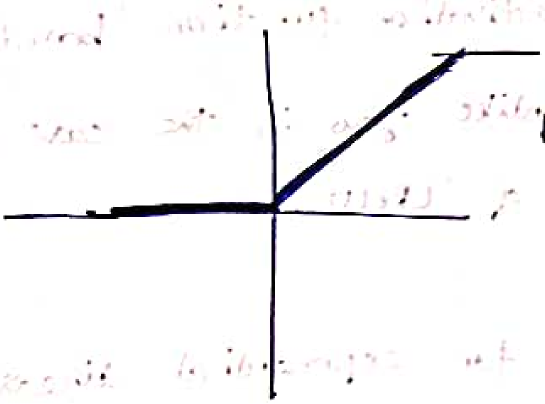
→ gives smoother convergence for any negative axis value.

→ for any positive output, it behaves like a step functive and gives a constant output.

ReLU-6 :-

→ It is a modification of the rectified linear unit where we limit the activation to a maximum size of 6.

→ this is due to increased robustness when used with low precision computation.



ReLU6 = min(max(features,0),6)

→ the range is from 0 to 6, and non-linearity only involves 8-bit ~~dim~~ integer arithmetic. 6 will take a max of 3 bits out of the 8 bits and leaving rest of 4/5 bits for the float values.

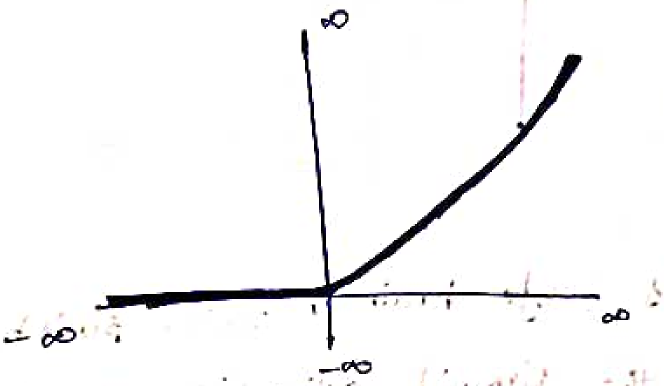
softplus :-

→ It is a smoother version of the rectifying non-linearity activation function and can be used to constrain a machine's output always to be positive.

-> produces output in the range of $(0, +\infty)$

-> difference between ReLU and softpuls is near 0, where the softplus is enticingly smooth and differentiable.

-> ReLU has efficient computation, but the computation for softpuls function is more expensive as it has log and exp in its formulatio



$\text{Softplus} = \ln(1+e^{x})$.

Softmax Activation function:

-> It also referred to as softargmax or normalized exponential function.

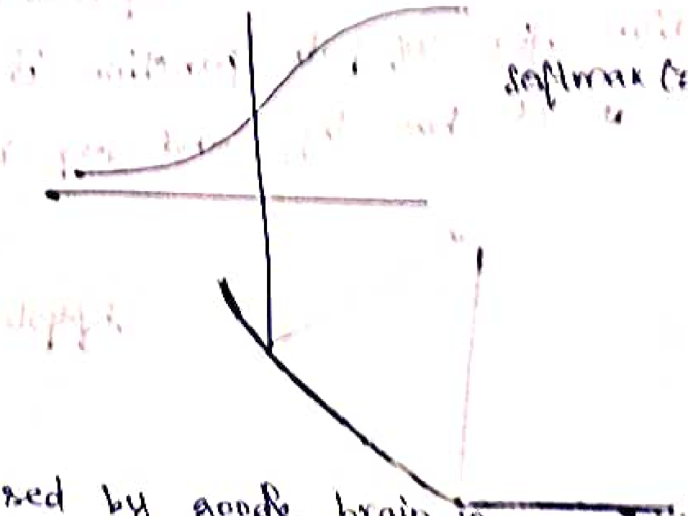
-> used for multinominal logistics regression, hence used in the output layer of a multi-class classification neural network.

-> softmax activation will output one value for each node in the output layer of the neural network. the target class will have the highest probability.

→ the output of softmax function is in the range between 0 to 1. the output of all nodes adds up to 1 and forms a probability of distribution.

→ It uses the cross entropy loss

→ It reduces the influence of outliers in the data without removing data points from the dataset.



$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Swish :-

→ proposed by google brain in 2017. swish is a gated version of the sigmoid activation function.

→ It is a smooth, non-monotonic function unlike ReLU.

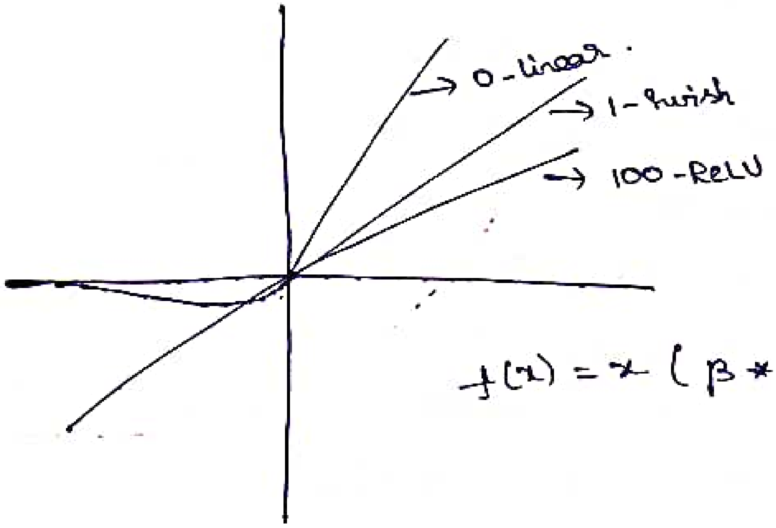
→ The non-monotonicity property of swish distinguish itself from most common activation functions. Non-monocity of swish increases expressivity and improves gradient flow.

→ It performs better than ReLU on deeper models across many challenging datasets.

→ like swish ReLU, swish is unbounded above and bounded below.

⇒ Unbounded above is derivable in the activation function to avoid saturation. and bounded below helps with a strong regularization effect.

⇒ Simplicity of swish and its similarity to ReLU gives better performance for deep neural networks by just replacing ReLU with swish, which is just a simple one-line code change.



$f(x) = x\,(\beta * \text{sigmoid}(x))$

⇒ Activation functions introduce nonlinearity to the linear transformed input in a neural network layer.

⇒ They play a critical role in the performance and training time of neural networks, and thus the choice of activation function is imperative for a neural network with better performance, less training time and lower loss.

Loss functions:

→ It is used to measure how far an estimated value is from its true value. It maps decisions to their associated costs.

→ loss functions are not fixed, they change depending on the task in hand and the goal to met.

→ Loss is nothing but a prediction error of neural net. and the method to calculate the loss is called loss function. Loss is used to calculate the gradients. gradients are used to update the weights of the neural net.

types:

- → mean squared error (MSE)
- → binary crossentropy (BCE)
- → categorical crossentropy (CC)
- → sparse categorical crossentropy (SCC)

MSE:

It is used for regression tasks. this loss is calculated by taking the mean of squared difference between actual (target) and predicted values.

$$MSE = 1/n \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2.$$

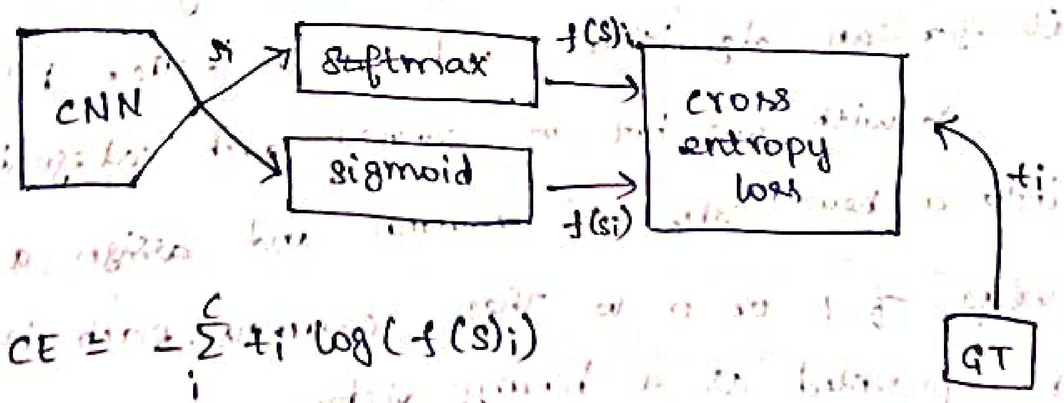
BCE :-

→ It is used for the binary classification. It is used when you need one output node to classify the data into two classes.

→ the output value should be passed through a sigmoid activation function and the range of output is (0,-1).

CC :-

→ It is used for multi class classification. It is used, w then the same no. of output nodes as the classes and the final layer output should be passed through a softmax activation so that each node output a probability value b/w (0-1).



$$CE = -\sum_{i}^{C} t_i \log(f(S)_i)$$

$$CE = -\sum_{i=1}^{C'=2} t_i \log(f(S_i)) = -t_1 \log(f(s_1)) - (1-t_1) \log(1-f(s_1)).$$

Sparse categorical crossentropy: (SCCE)

It is similar to categorical crossentropy ove, for one change.

=> When we use SCCE loss function, you do, need to one hot encode the target vector. If the target image is of a cat, you simply pass 0, othe Basically, whichever the class is you just pass the index of that class.

One hot Encoding:

=> It is the process in which categorical variable are converted into a form that could be provided to ML algorithm to do a better job in prediction.

=> It is one method of converting data to prepa it for an algorithm and get a better prediction.

=> With one-hot, we convert each categorical value into a new categorical column and assign a binary value of 1 or 0 to these columns. each integer val is represented as a binary vector.

=> Hot code are execution paths in your application / compiler in which most of the execut time is spent, and thus which are potentially executed very often.

problems in one hot encoding:

It doesn't scale well when the no. of o/p labels is large.

eg: in language modeling the no. of output labels = vocabulary size.

this means each input feature (word) will be represented as a huge vector. each vector is equidistant from every other vector in a one hot encoding.