Deep Learning

Neural Networks : Activation Functions

Activation functions in Neural Networks

- The reason for using activation functions in Neural Networks are as follows:
- **1.** The idea behind the activation function is to introduce nonlinearity into the neural network so that it can learn more complex functions.
- **2.** Without the Activation function, the neural network behaves as a linear classifier, learning the function which is a linear combination of its input data.
- **3.** The activation function converts the inputs into outputs.
- **4.** The activation function is responsible for deciding whether a neuron should be activated i.e, fired or not.
- **5.** To make the decision, firstly it calculates the weighted sum and further adds bias with it.
- **6.** So, the basic purpose of the activation function is to introduce non-linearity into the output of a neuron.

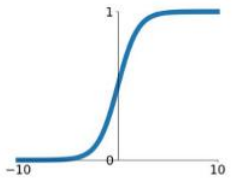
Activation Function

- Activation functions live inside neural network layers and modify the data they receive before passing it to the next layer.
- Activation functions give neural networks their power — allowing them to model complex non-linear relationships.
- By modifying inputs with non-linear functions neural networks can model highly complex relationships between features.

Popular activation functions

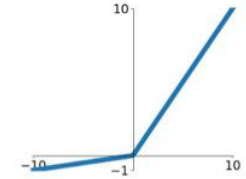
Sigmoid

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



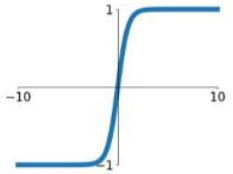
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

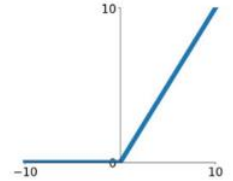


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

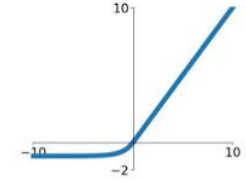
ReLU

$\max(0, x)$

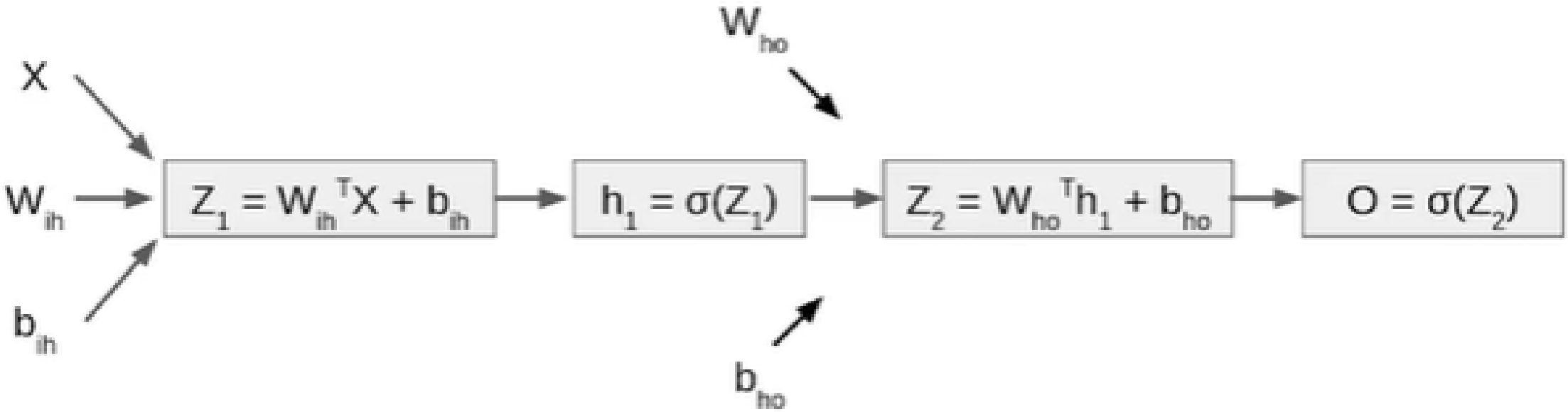


ELU

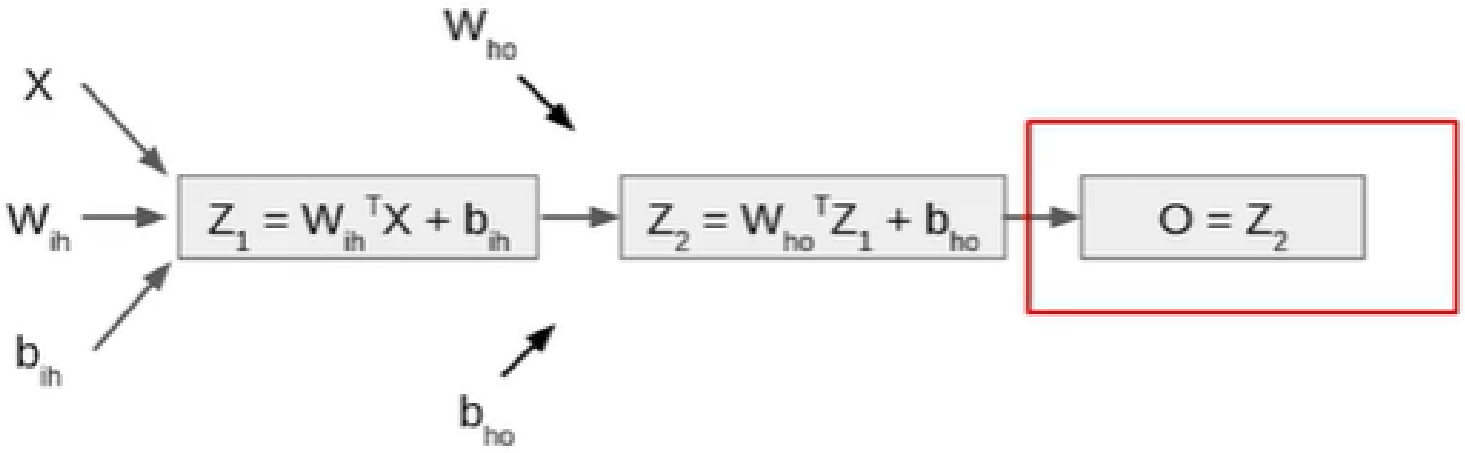
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Working of Neural Network



Removing Activation functions



$$Z_1 = W_{ih}^T X + b_{ih}$$

$$Z_2 = W_{ho}^T Z_1 + b_{ho} \quad = W_{ho}^T (W_{ih}^T X + b_{ih}) + b_{ho}$$

This is linear combination of inputs and weights. This network cannot capture the comlex relationships in data. Check in Tensorflow playground.

Why Activation function is needed

- The complex configuration of weights possessed by neural networks makes it difficult to solve the problem.
- The main reason lies in the concept of Non-Linearity

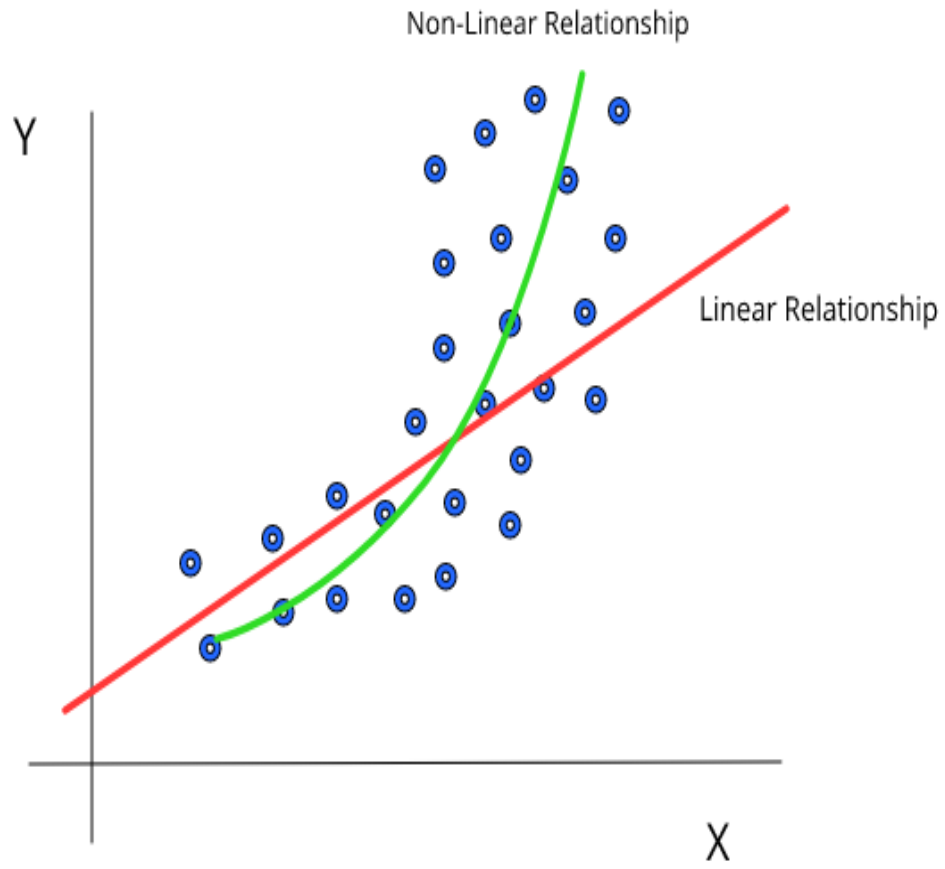
Linearity vs non-linearity

$$Y = W_1 * X_1 + W_2 * X_2$$ is a Linear function

- The above equation represents a **linear relationship** between Y and X1,X2. Regardless of what values W1 and W2 have, at the end of the day the change of value of X1 and X2 will result in a **linear** change in Y.
- **Activation functions are always differentiable.**

Why Activation function is needed

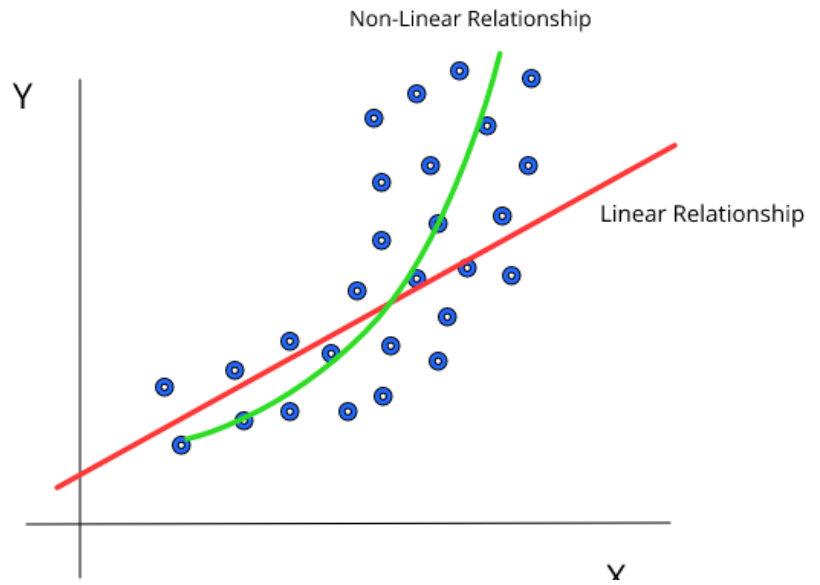
- If we do not apply the Activation function, then the output signal will only be a **Linear function.** Although a linear function is easy to solve, it will possess less functional complexity and mapping.
- Also without activation function, our Neural network would not be able to learn and model other complicated data such as images, videos, audios, speech etc.
- That is why we use Artificial Neural network techniques (**Deep learning) to make sense of something complicated, high dimensional, non-linear-big datasets, where the model has lots and lots of hidden layers.**



Why Activation function is needed

- If the data scientist tries to fit in the linear relationship, red is generated as an output, which is not accurate.
- But if **non linear** relationships is tried then green line is generated which is much better and near to the desired results.

✓*So keep some non linear function as the activation function for each neuron and our neural network is now **capable** of fitting on non linear data.*



Without Activation Function

$$y = \sum_{i=0}^{n}(W_i * X_i) + B$$

With Activation Function

$$y = f(\sum_{i=0}^{n}(W_i * X_i) + B)$$

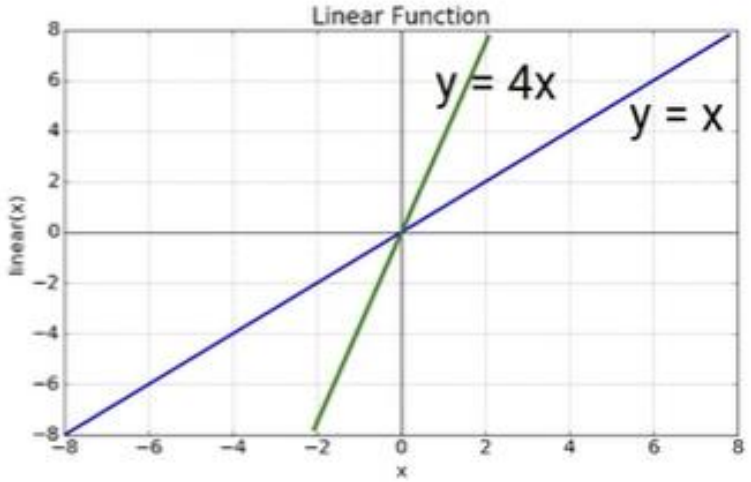
Activation function - Properties

- Non-linear - In linear regression we're limited to a prediction equation that looks like a straight line
- But what if the patterns in our dataset were non-linear? (e.g. x2, sin, log). To model these relationships we need a non-linear prediction equation.[1]
- Activation functions provide this non-linearity.
- Continuously differentiable — To improve our model with gradient descent, we need our output to have a nice slope so we can compute error derivatives with respect to weights.
- If our neuron instead outputted 0 or 1 (perceptron), we wouldn't know in which direction to update our weights to reduce our error.

Conditions for Activation Functions

- Activation Function Must be Continuous (Infinity values)

  - Linear Activation Function: $y = ax$
  - Graph of Linear Function:
  - Input Range: (- ∞ to ∞)
  - Output Range: (- ∞ to ∞)



Conditions for Activation Functions

2. Activation Function must be Differentiable at all points. In bak propagation we update the gradients in order to update the parameters. We also calculate the partial derivatives for the activation functions used in the network.

- Linear Activation Function:

$$y = ax$$

- Linear Activation Function derivative:

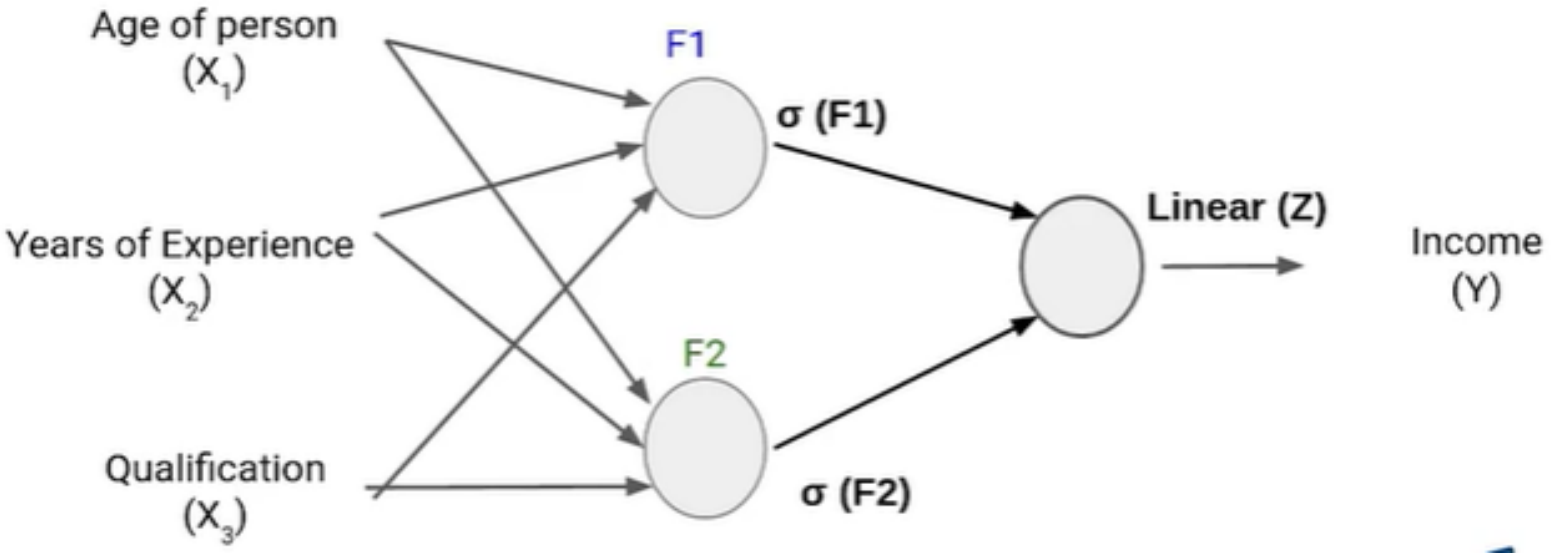
$$\frac{dy}{dx} = a$$

Eg y=ax 2…….. Dy/dx = 2ax

Linear Function : Output Layer

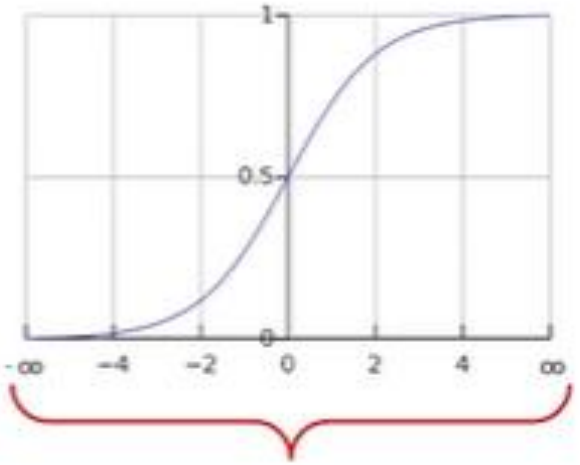
- As Linear Activation Function Does not capture the non-linear relationships in the data. So it is used at output layer for regression problem.

- Linear Activation Function: $y = ax$



Sigmoid Activation Function

- Sigmoid Function Formula: $$\sigma(x) = \frac{1}{1+e^{-(x)}}$$
- Sigmoid Function Graph:

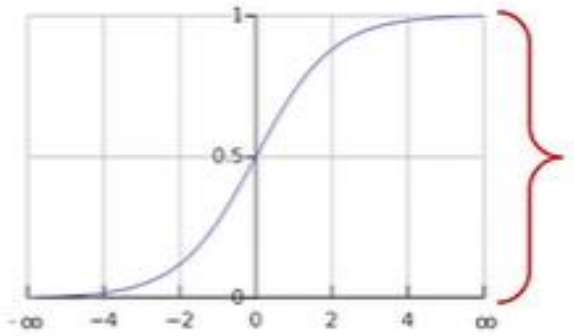


- Input Range: (- ∞ to ∞)

Can take any real values

Sigmoid Activation Function

- Sigmoid Function Formula: $$\sigma(x) = \frac{1}{1+e^{-(x)}}$$
- Sigmoid Function Graph:



- Input Range: (- ∞ to ∞)
- Output Range: (0 to 1)

Restricts the output between 0 to 1. These values are probabilities and this function is continuous and differentiable at all times.

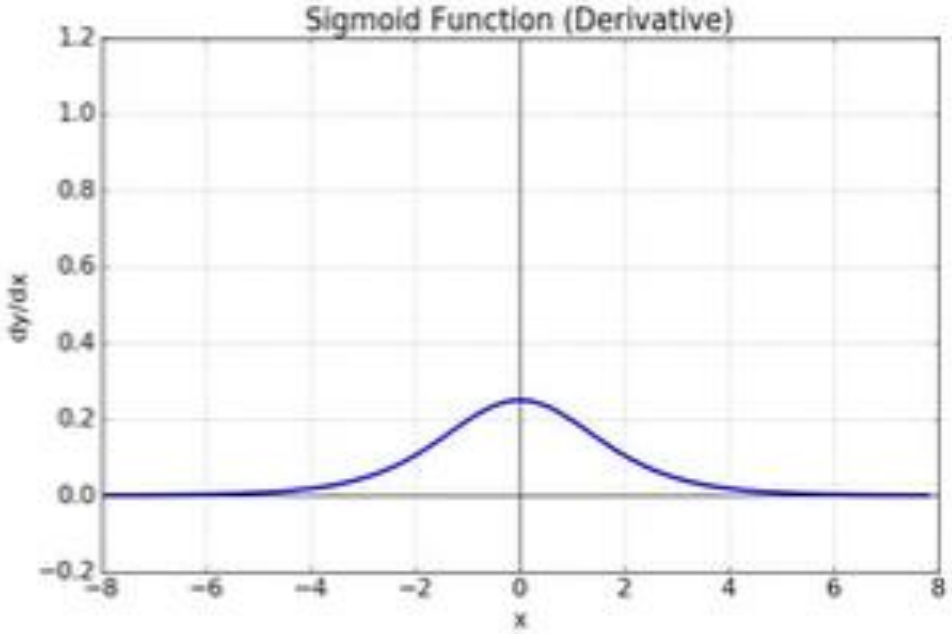
Sigmoid Activation Function

- Sigmoid Function:

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

- Sigmoid Function derivative:

$$\frac{d\sigma}{dx} = \sigma(x)\,(1-\sigma(x))$$



This curve is flat which means the derivatives or the gradient values for this activation function will be very small.

Tanh Activation Function (Scale Sigmoid Function)

Tanh Activation Function: $$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

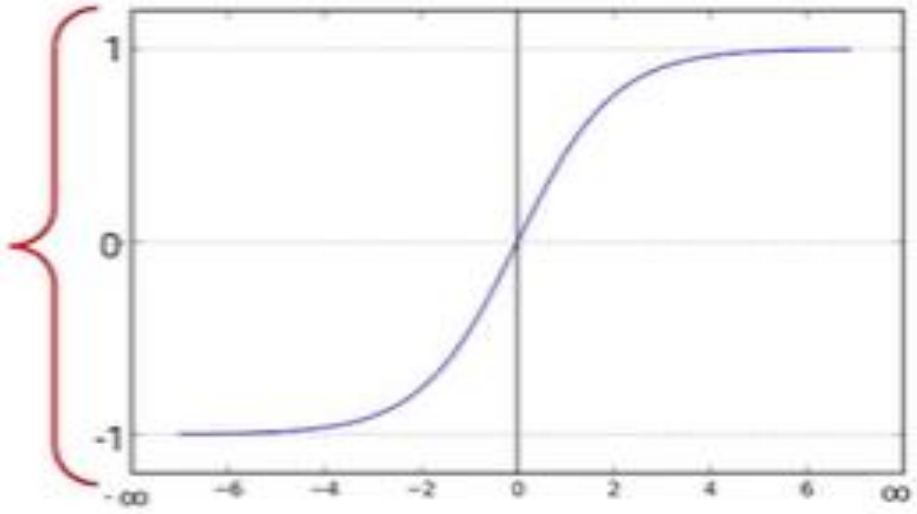
$$\sigma(x) = \frac{1}{1+e^{-(x)}}$$

$$tanh(x) = 2\,\sigma(2x) - 1$$

A factor of 2 and -1, that is why it is called scale activation function.

Graph of Tanh

- Tanh Activation Function: $$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$
- Tanh Activation Function Graph:
- Input Range: (- ∞ to ∞)
- Output Range: (-1 to 1)



When x=0, y is also 0, that is why the curve cuts the y axis at 0

Comparison of Sigmoid and Tanh

- Tanh function is steeper at the centre and it is simply a scaled version of sigmoid.
- It is continuous differentiable at all points.

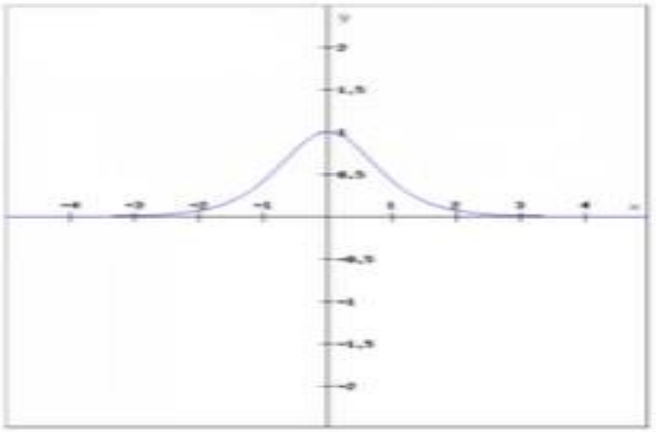
- Tanh Activation Function:

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

- Tanh Function derivative:

$$\frac{\mathrm{d}\, tanh}{\mathrm{d}\, x} = 1 - tanh^2(x)$$

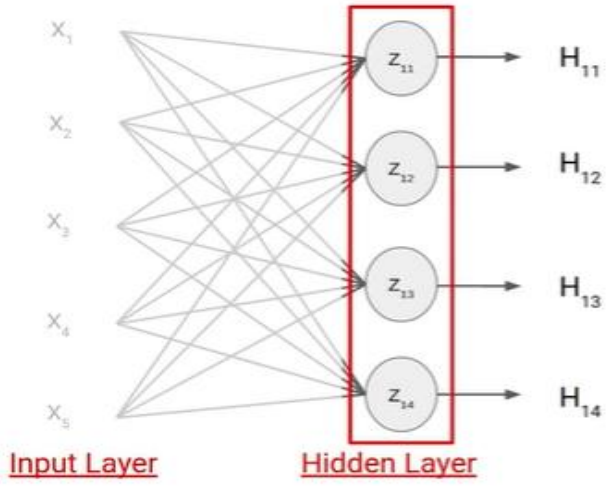
Graph of Derivative of tanh

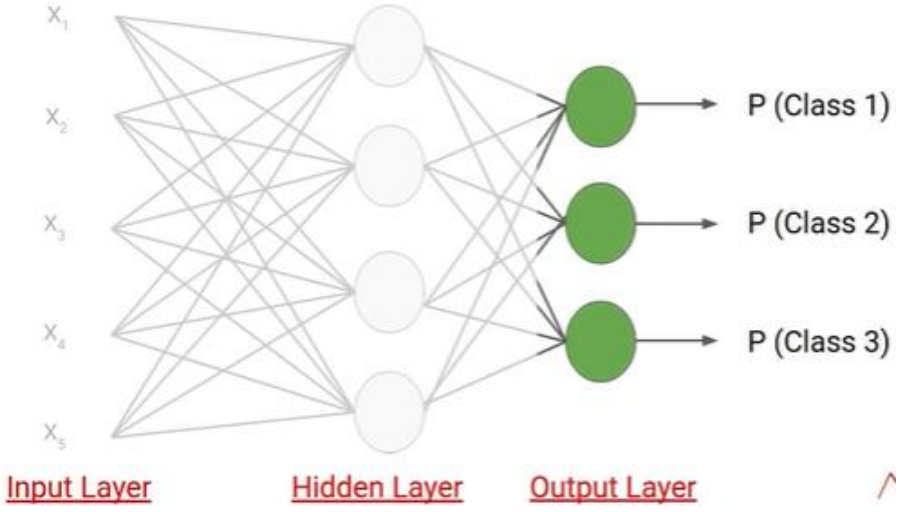


- In the graph, the values of tanh are comparatively larger. Hence the training in tanh is faster as compared to sigmoid since the gradient values would be larger and hence the update process would be faster. Check Tensorflow playground.

Softmax Activation Function

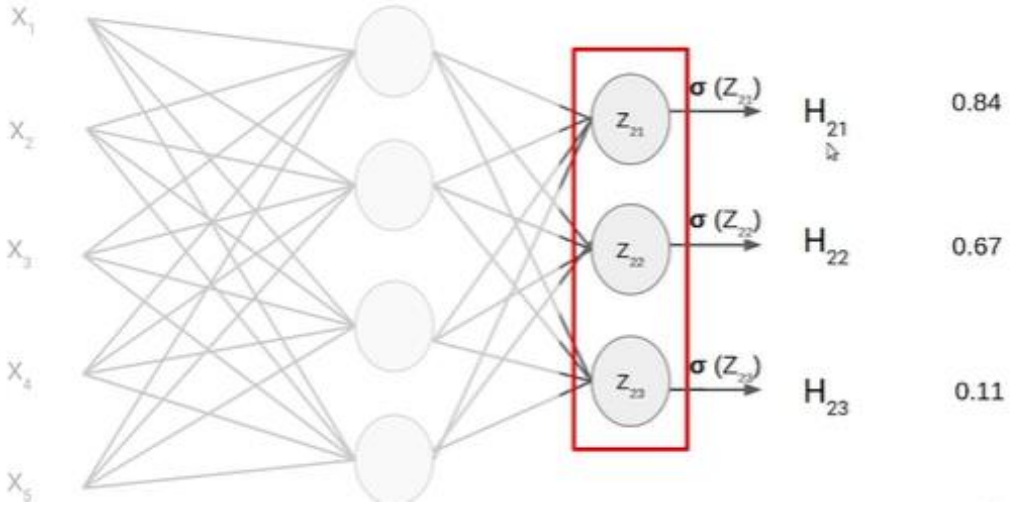
- Used for Multi-class Classification problems.
- Used in the output layer



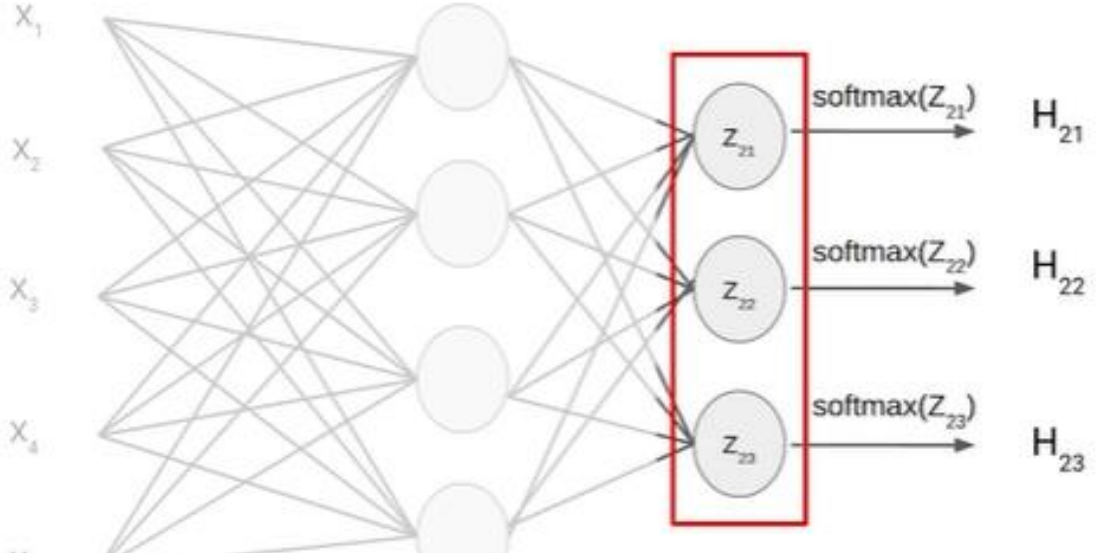


- The neurons in the output layer will be equal to the target classes. Each of individual neuron will give probability of the individual classes.

Sigmoid cannot be applied



Softmax is applied : calculates relative probabilities.

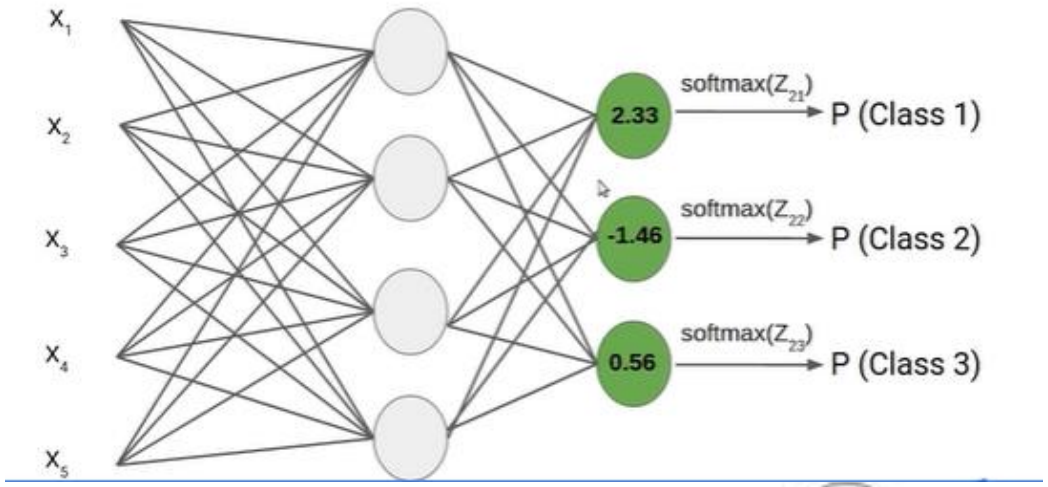


Softmax Activation Function

- Returns probability for each class
- Sigmoid Activation Function: $$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$
- Non-linear activation function
- Probability of all classes

Z values represent the values from the neurons in the output layer. If the number of neurons is 3, there will be 3 z values. Exp acts as the non linear functions. These values are divided by the sum of exp values in order to normalize the values and convert them into probabilities. If the no of classes is 2, it becomes same as sigmoid function. So sigmoid is just a variant of Softmax.

Softmax Example



$$2.33 \rightarrow P(\text{Class 1}) = \frac{\exp(2.33)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.83827314$$

$$-1.46 \rightarrow P(\text{Class 2}) = \frac{\exp(-1.46)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.01894129$$

$$0.56 \rightarrow P(\text{Class 3}) = \frac{\exp(0.56)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.14278557$$

Why is ReLU the most commonly used Activation Function?

- **1. No vanishing gradient:** The derivative of the RELU activation function is either 0 or 1, so it could be not in the range of [0,1]. As a result, the product of several derivatives would also be either 0 or 1, because of this property, the vanishing gradient problem doesn't occur during backpropagation.
- **2. Faster training:** Networks with RELU tend to show better convergence performance. Therefore, we have a much lower run time.
- **3. Sparsity:** For all negative inputs, a RELU generates an output of 0. This means that fewer neurons of the network are firing. So we have sparse and efficient activations in the neural network.

Optimizers

- Optimizers are algorithms or methods used **to change the attributes of the neural network such as weights and learning rate to reduce the losses**. Optimizers are used to solve optimization problems by minimizing the function.
- **Optimizers** are algorithms or methods used to minimize an error function(*loss function*)or to maximize the efficiency of production. Optimizers are mathematical functions which are dependent on model's learnable parameters i.e Weights & Biases. Optimizers help to know how to change weights and learning rate of neural network to reduce the losses.

Types of Optimizers

- **Gradient Descent**
- Gradient descent is an optimization algorithm based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum.
- Gradient Descent iteratively reduces a loss function by moving in the direction opposite to that of steepest ascent. It is dependent on the derivatives of the loss function for finding minima.
- uses the data of the entire training set to calculate the gradient of the cost function to the parameters which requires large amount of memory and slows down the process.

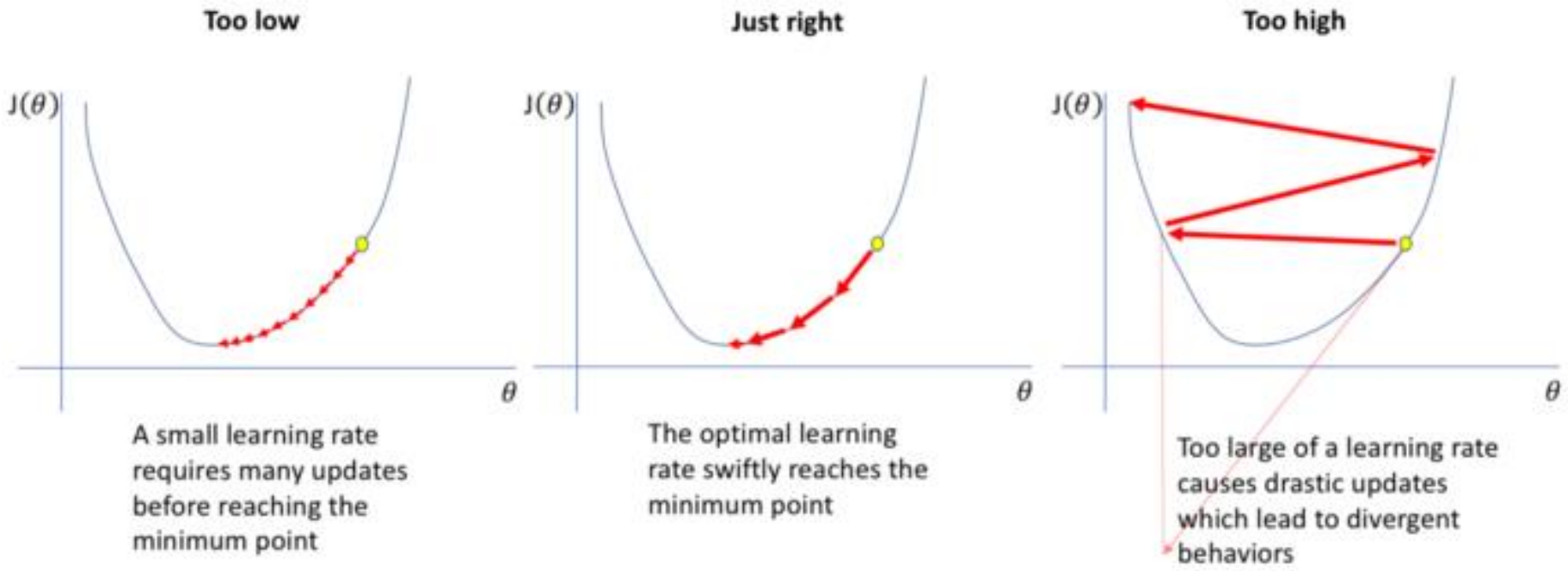
$$W_{new} = W_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$

Gradient Descent

- **Advantages of Gradient Descent**
- Easy to understand
- Easy to implement
- **Disadvantages of Gradient Descent**
- Because this method calculates the gradient for the entire data set in one update, the calculation is very slow.
- It requires large memory and it is computationally expensive.

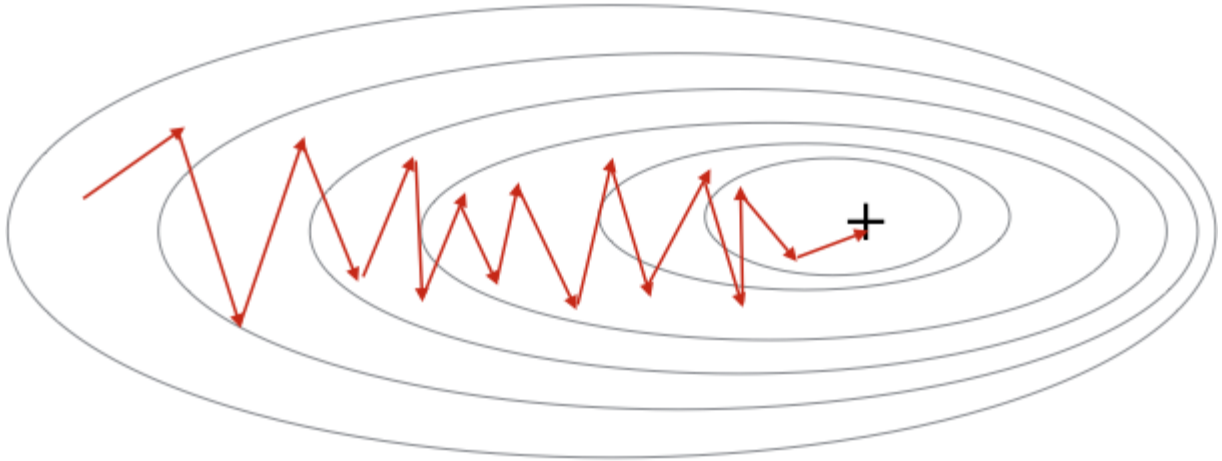
Learning Rate

- How big/small the steps are gradient descent takes into the direction of the local minimum are determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights.



Stochastic Gradient Descent

- It is a variant of Gradient Descent. It update the model parameters one by one. If the model has 10K dataset SGD will update the model parameters 10k times.

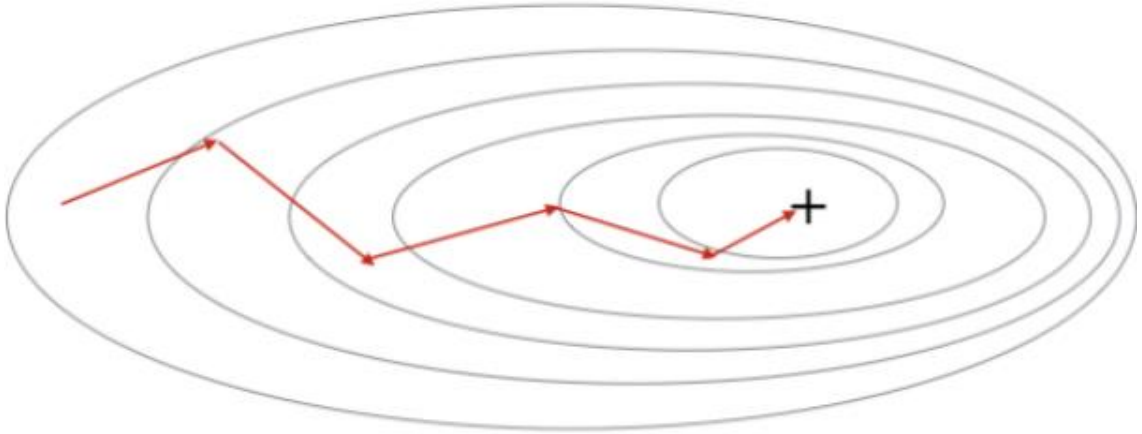


Stochastic Gradient Descent

- **Advantages of Stochastic Gradient Descent**
- Frequent updates of model parameter
- Requires less Memory.
- Allows the use of large data sets as it has to update only one example at a time.
- **Disadvantages of Stochastic Gradient Descent**
- The frequent can also result in noisy gradients which may cause the error to increase instead of decreasing it.
- High Variance.
- Frequent updates are computationally expensive.

Mini-Batch Gradient Descent

- It is a combination of the concepts of SGD and batch gradient descent. It simply splits the training dataset into small batches and performs an update for each of those batches.
- This creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent. it can reduce the variance when the parameters are updated, and the convergence is more stable. It splits the data set in batches in between 50 to 256 examples, chosen at random.



Mini Batch Gradient Descent

- **Advantages of Mini Batch Gradient Descent:**
- It leads to more stable convergence.
- more efficient gradient calculations.
- Requires less amount of memory.
- **Disadvantages of Mini Batch Gradient Descent**
- Mini-batch gradient descent does not guarantee good convergence,
- If the learning rate is too small, the convergence rate will be slow. If it is too large, the loss function will oscillate or even deviate at the minimum value.

Adagrad optimizer

- Keras Adagrad optimizer has **learning rates that use specific parameters**.
- Based on the frequency of updates received by a parameter, the working takes place.
- Even the learning rate is adjusted according to the individual features. This means there are different learning rates for some weights.
- The intuition behind AdaGrad is can we use different Learning Rates for each and every neuron for each and every hidden layer based on different iterations.

Types of optimizers

- **Advantages of AdaGrad**
- Learning Rate changes adaptively with iterations.
- It is able to train sparse data as well.
- **Disadvantage of AdaGrad**
- If the neural network is deep the learning rate becomes very small number which will cause dead neuron problem.

RMS-Prop (Root Mean Square Propagation)

- RMS-Prop is a special version of Adagrad in which the learning rate is an exponential average of the gradients instead of the cumulative sum of squared gradients. RMS-Prop basically combines momentum with AdaGrad.
- **Advantages of RMS-Prop**
- In RMS-Prop learning rate gets adjusted automatically and it chooses a different learning rate for each parameter.
- **Disadvantages of RMS-Prop**
- Slow Learning

AdaDelta

- Adadelta is an extension of Adagrad and it also tries to reduce Adagrad's aggressive, monotonically reducing the learning rate and remove decaying learning rate problem. In Adadelta we do not need to set the default learning rate as we take the ratio of the running average of the previous time steps to the current gradient.
- **Advantages of Adadelta**
- The main advantage of AdaDelta is that we do not need to set a default learning rate.
- **Disadvantages of Adadelta**
- Computationally expensive

Adam(Adaptive Moment Estimation)

- Adam optimizer is one of the most popular and famous gradient descent optimization algorithms. It is a method that computes adaptive learning rates for each parameter. It stores both the decaying average of the past gradients , similar to momentum and also the decaying average of the past squared gradients , similar to RMS-Prop and Adadelta. Thus, it combines the advantages of both the methods.
- **Advantages of Adam**
- Easy to implement
- Computationally efficient.
- Little memory requirements.

How to choose optimizers?

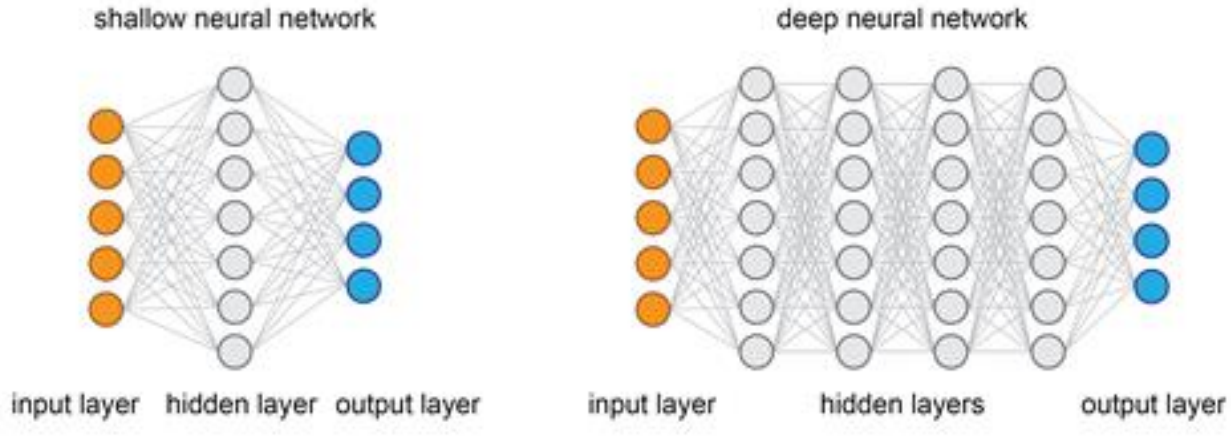
- If the data is sparse, use the self-applicable methods, namely Adagrad, Adadelta, RMSprop, Adam.
- RMSprop, Adadelta, Adam have similar effects in many cases.
- Adam just added bias-correction and momentum on the basis of RMSprop,
- As the gradient becomes sparse, Adam will perform better than RMSprop.

Deep Neural Networks vs Shallow Neural Networks

- Neural networks contain hidden layers apart from input and output layers. There is only a single hidden layer between the input and output layers for shallow neural networks whereas, for Deep neural networks, there ar
- To approximate any function, both shallow and deep networks are good enough and capable but when a shallow neural network fits into any function, it requires a lot of parameters to learn. On the contrary, deep networks can fit functions even better with a limited number of parameters since they contain several hidden layers.
- So, for the same level of accuracy, deeper networks can be much more powerful and efficient in terms of both computation and the number of parameters to learn.
- e multiple layers used.

Deep Neural Networks vs Shallow Neural Networks

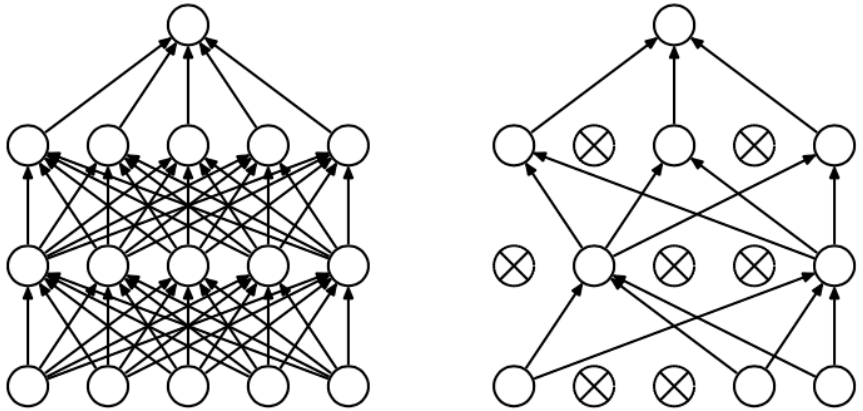
- One other important thing about deeper networks is that they can create deep representations and at every layer, the network learns a new, more abstract representation of the input.
- Therefore, in modern days deep neural networks have become preferable owing to their ability to work on any kind of data modeling.



Google Images

Overfitting in Neural Networks.

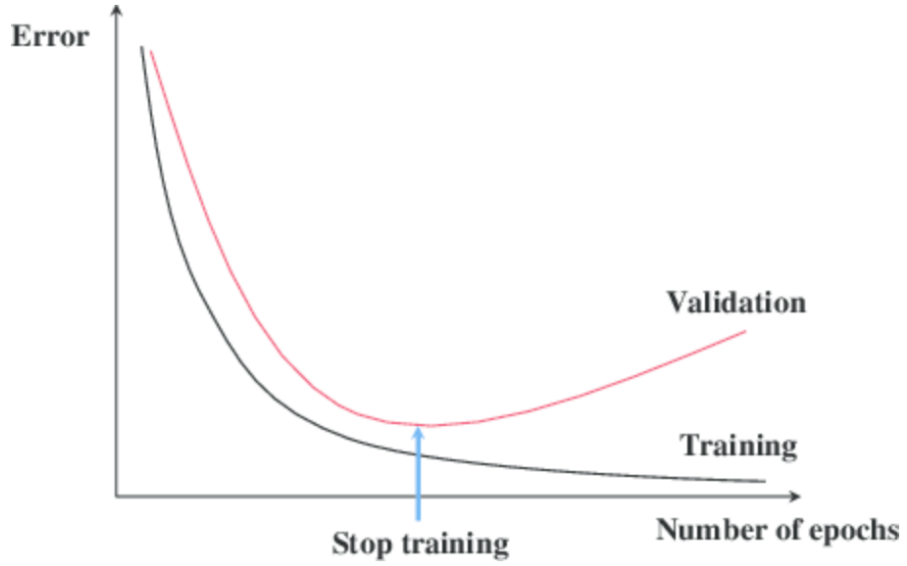
- **Dropout:** It is a regularization technique that prevents the neural network from overfitting. It randomly drops neurons from the neural network during training which is equivalent to training different neural networks. The different networks will overfit differently, so the net effect of the dropout regularization technique will be to reduce overfitting so that our model will be good for predictive analysis.



Google Images

Overfitting in Neural Networks

- **Early stopping:** This regularization technique updates the model to make it better fit the training data with each iteration. After a certain number of iterations, new iterations improve the model. After that point, however, the model begins to overfit the training data. Early stopping refers to stopping the training process before that point.



Google Images

Conclusion

- SGD is a very basic algorithm and is hardly used in applications now due to its slow computation speed.
- One more problem with that algorithm is the constant learning rate for every epoch.
- Moreover, it is not able to handle saddle points very well. Adagrad works better than stochastic gradient descent generally due to frequent updates in the learning rate.
- It is best when used for dealing with sparse data. RMSProp shows similar results to that of gradient descent algorithm with momentum, just differs the way by which the gradients are calculated.
- Lastly comes the Adam optimizer that inherits the good features of RMSProp and other algorithms.
- The results of the Adam optimizer are generally better than every other optimization algorithms, have faster computation time, and require fewer parameters for tuning. Because of all that, Adam is recommended as the default optimizer for most of the applications.
- Choosing the Adam optimizer for your application might give you the best probability of getting the best results.

References

- Medium.com
- https://www.analyticsvidhya.com/