# Background:

Deep Learning Neural networks

The idea of neural network was first stirked in the year 1957 in the form of percetron(single neuron). In 1986 Hinton gave the concept of back propogation but at that time due to lack of resources the training of deep neural networks was not feasible then with the advancement of resources(memory and computation power) the neural networks got powerful and deeper.

# Inspiration:

Artificial neural networks are inspired by biological neural networks. Biological neuron has nucleus which acts as activation function and it has dendrites which acts as input vectors and the output is axon. The thicker the dendrite is the more important the feature is. And in artificial neural network the thickness of a dendrite is determined by the weight vector. Biological neural networks is a group of interconnected neurons.

# Logistic Regression and Preceptron:

Perceptron is a single artificial neural with threshold activation function on the other hand if we make this threshold activation into sigmoid then it will act as logistic regression.

# Why Multilayer Perceptron?

1. Biological Inspiration.
2. Mathematical inspiration: More the depth , more complex functions can be fitted into it.

**Training** a neural network means finding the best and optiomal weight vector.

# Back propogation:

1. **Initialize weights:** This can be done randomly or any method can be used.
2. 2.1 Forward propogation: Sending data forward through the network.
   1. At output compute loss.
   2. Compute all the required derivatives.
   3. Update weights starting from the end of the network.
3. Repeat step 2 till convergence.

Convergence is when the new weight vector is similar to the old one.

When all the training points traverse through the network once is known as one epoch.

Back propogation only works when the activation function is differentiable.

# Memoization:

If there is any operation that is used multiple times repeatedly then compute it once, store it and reuse it. In case of Back propogation some derivatives will be required again repeatedly so there memoization can be used.

# Vanishing Gradients:

Multiplication of small derivatives leads to very smaller gradients and that means the change in new and old weight vector won't be there. This is vanishing gradient problem.

# Exploding gradient:

When larger dervatives multiplies then the result is larges which do not let the weights to converge.

# Activation Functions:

Provide non linearity to the network. Sigmoid and Tanh functions are differentiable and their derivative can be represented in their form itself.

**RELU (Rectified Linear Units)**: To avoid vanishing and exploding gradient problem this is the most famous activation functions these days. f(z)=maz(0,z) Here also we have a dying relu problem so to overcome these there are other versions of relu such as leaky relu etc.

**DroupOut**: Droupout technique is used as a regularization in deep neural networks so as to reduce overfitting. In this technique some of the neurons get shut down and they are not a part of any forward or backward pass in a particular iteration. But while testing all the neurons are present and their weights are mulltiplied with the frequency of activation while training.

**Batch Normalization**: It refers to implementation of normalisation in deep layers of neural networks. As the networks gets complex the inputs of subsequent layers starts to get shift from the original scale so by normalising the inputs of some layers this can be avoided. This shifting is known as internal covariate shift. Batch Normalization helps in training the model faster as larger learning rate can be used and it also acts as weak regularizer. It performs shifting and scaling using alpha and beta parameters while are to be learned while training. BN : alpha\*xi' + beta. Here alpha and beta are parameters which are learned while training.

**Softmax Activation:** This is used in the final output layer of the network in case of multiple class classification. It gives the probality of output's belongingness to different classes.

**Epoch**: When whole of the input data itrerate once through the network is known as one epoch.

**Weight Initialization:** This is very important as this can decide that which way our model should work. initial weights should not be biased. There are different techniques for weight initialization such as Uniform Initilization, Xavier/Glorot Initialization , He Iniatialization etc.

These uses two terms fan-in and fan-out:

**Fan-in**: Number of inputs to a neuron.

**Fan-out**: Number of outputs from a neuron.

**Optimizers**: The way to reach convergence i.e minima or maxima.

**Convex function:** These functions have only one minima or one maxima.

**Non-convex function:** These function have multiple minima and maxima known as local minima and maxima.

At minima and maxima the slope is 0. But at saddle point also the slope is 0. So weight initialization play a important role here..

**SGD: Stochastic Gradient Descent:** This is a technique to update weights using gradients(slope) ti reach the convergence point.

**Momentum**: This is a technique used to faster the convergence speed and uses the concept of weighted sum. It has a parameter alpha that can be between 0 to 1.

# Vt = Alpha Vt-1 + learning rate. gradient Wt = Wt-1 - Vt

**Nesterov Accelerated Gradient (NAG)**: This is just a modification of above . It first calculates momentum and then gradient over that. ALmost both the approaches' results are similar.

**AdaGrad (Adaptive Gradients):** It tweaks with the learning i.e it makes the learning rate adaptive as per the gradients. So no need to manually tune the learning rate. But sometime it can lead to very small learning rate resulting in slower convergence.

**Adadelta:** This optimizer overcomes the problems of adagrad i.e very small learning rates. It uses Exponentially Decaying Averages of gradients to calculate learning rates.

**Adam**: It uses Exponiantially decaying Averages of gradients and gradient squares. It is the most famous and fast optimizer of today.

# Bias variance tradeoff Neural Networks:

If number of layers increases there will be more weight and that leads to overfitting and higher variance.

If the number of layers is less then less weights will be there and that leads to underfitting and higher bias.

Thanks