**Optimizers in Deep Learning:**

The process of minimizing (or maximizing) any mathematical expression is called **optimization** .An optimizer is a method or algorithm to update the various parameters that can reduce the loss in much less effort. In a neural network, we have many **weights**in between each layer. We have to understand that **each and every** weight in the network will affect the output of the network in some way, because they are all directly or indirectly connected to the output. Optimizers are used to change the attributes of your Neural Network such as weights and biases (w,b) in order to reduce the losses. So in every Neural Network, you perform 2 types of computation -

1. Forward Propagation
2. Backward Propagation

We call 1 Epoch -> Forward Propagation + Backward Propagation for the entire training dataset.

An epoch refers to one cycle through the full training dataset. We repeat this cycle in the multiple **Epochs** until the loss/cost converges to the global minimum.

**Optimization Algorithms**

## Gradient Descent (GD)

This is the most basic optimizer that directly uses the derivative of the loss function and learning rate to reduce the loss and achieve the minimal. This approach is also adopted in back propagation in neural networks where the updated parameters are shared between different layers depending upon when the minimum loss is achieved.

## Batch Gradient Descent

In the Batch Gradient Descent, we pass the entire training set to the Neural Network. But there can be disadvantages with this approach as when we have millions of examples then, it will take a huge amount of RAM to load all the examples, and also weights updation will take a longer time, and eventually the convergence will happen very slowly.

## Stochastic Gradient Descent

In the Stochastic Gradient Descent, 1 record is passed at a time to the Neural Network. Suppose you have 10K examples, so in SGD 1 Epoch will have 10K iterations. Now if you are performing 20 epochs then there will be**10K \* 20 iterations**. This approach solves the Huge RAM problem but the convergence will be very-very slow in this as well.

## Mini-Batch Gradient Descent

Another variant of this GD approach is mini-batch, where the model parameters are updated in small batch sizes. It means that after every n batches, the model parameters will be updated and this ensures that the model is proceeding towards minimal in fewer steps without getting derailed often. This results in less memory usage and low variance in the model.

## Momentum Based Gradient Descent

An Adaptive Optimization Algorithm which uses exponentially weighted averages of gradients over previous iterations to stabilize the convergence, resulting in quicker optimization. For example, in most real-world applications of Deep Neural Networks, the training is carried out on noisy data. It is, therefore, necessary to reduce the effect of noise when the data are fed in batches during Optimization. This problem can be tackled using **Exponentially Weighted Averages** (or Exponentially Weighted Moving Averages).

## Adam (Adaptive Moment Estimation)

Fully known as the Adaptive Moment Estimation Algorithm, but abbreviated Adam, this optimization algorithm was introduced in 2015 by two researchers – Diederik P. Kingma and Jimmy Lei Ba. This algorithm simply estimates moments and uses them to optimize a function. It is essentially a combination of the gradient descent with momentum algorithm and the RMS (Root Mean Square) Prop algorithm. The Adam algorithm calculates an exponential weighted moving average of the gradient and then squares the calculated gradient. This algorithm has two decay parameters that control the decay rates of these calculated moving averages.

There are several advantages of the Adam Algorithm and some of them are listed below:

* Easy to implement
* Quite computationally efficient
* Requires little memory space
* Good for non-stationary objectives
* Works well on problems with noisy or sparse gradients
* Works well with large data sets and large parameters

**LOSS FUNCTION:**

Loss function is a method of evaluating “how well your algorithm models your dataset”. If your predictions are totally off, your loss function will output a higher number. If they’re pretty good, it’ll output a lower number. As you tune your algorithm to try and improve your model, your loss function will tell you if you’re improving or not. ‘Loss’ helps us to understand how much the predicted value differs from actual value.

**Types of Loss function:**

1. **Regression Loss Function:** Regression models deals with predicting a continuous value for example given floor area, number of rooms, size of rooms, predict the price of the room. The loss function used in the regression problem is called “Regression Loss Function”.

2. **Binary Classification Loss Functions:** Binary classification is a prediction algorithm where the output can be either one of two items, indicated by 0 or 1. The output of binary classification algorithms is a prediction score (mostly). So the classification happens based on the threshold value (default value is 0.5). If the prediction score > threshold then 1 else 0.

3. **Multi-class Classification Loss Functions:** Multi-Class classification are those predictive modeling problems where there are more target variables/class. It is just the extension of binary classification problem.

