**Submitted by :**

**Chandrashekhar M**

**Optimizers :**

Optimizers are algorithms or methods used to change the attributes of neuralnetwork such as weights and learning rate in order to reduce the losses. Optimizers help to get results faster.

**Why do we need Optimizers :**

To train a neural network model, we must define a loss function in order to measure the difference between our model predictions and the label that we want to predict. What we are looking for is a certain set of weights, with which the neural network can make an accurate prediction, which automatically leads to a lower value of the loss function.

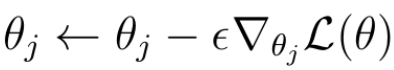
**Types of Optimizers :**

* Gradient Descent
* Stochastic Gradient Descent
* Adam
* AdaGrad
* RMSProp

**1.Gradient Descent :**

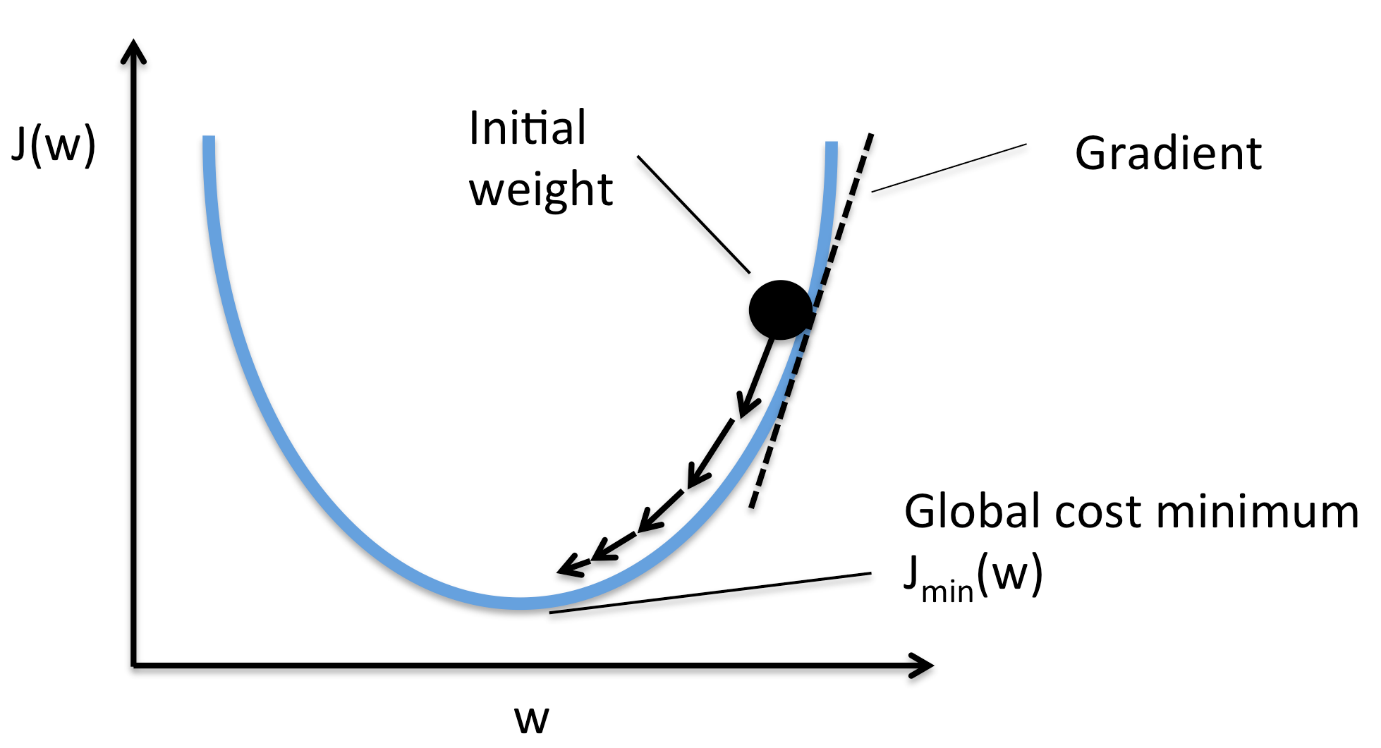
Gradient Descent is the most basic but most used optimization algorithm. It’s used heavily in linear regression and classification algorithms. Backpropagation in neural networks also uses a gradient descent algorithm.

Gradient descent is a first-order optimization algorithm which is dependent on the first order derivative of a loss function. It calculates that which way the weights should be altered so that the function can reach a minima. Through backpropagation, the loss is transferred from one layer to another and the model’s parameters also known as weights are modified depending on the losses so that the loss can be minimized.

****

In this technique we must calculate the gradient of the loss function **L** with respect to the weights (or parameters **θ**) that we want to improve. Subsequently, the weights/parameters are updated in the direction of the negative direction of the gradient

**By periodically applying the gradient descent to the weights, we will eventually arrive at the optimal weights that minimize the loss function and allow the neural network to make better predictions.**

****

**Advantages and Disadvantages :**

**Advantages :**

* Less oscillations and noisy steps taken towards the global minima of the loss function due to updating the parameters by computing the average of all the training samples rather than the value of a single sample.
* It can benefit from the vectorization which increases the speed of processing all training samples together.
* It is computationally efficient as all computer resources are not being used to process a single sample rather are being used for all training samples.

**Disadvantages :**

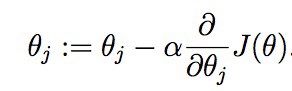
* Sometimes a stable error gradient can lead to a local minima and unlike stochastic gradient descent no noisy steps are there to help get out of the local minima.
* The entire training set can be too large to process in the memory due to which additional memory might be needed.
* Depending on computer resources it can take too long for processing all the training samples as a batch.

**2. Stochastic Gradient Descent :**

There are a few downsides of the gradient descent algorithm. We need to take a closer look at the amount of computation we make for each iteration of the algorithm.

Say we have 10,000 data points and 10 features. The sum of squared residuals consists of as many terms as there are data points, so 10000 terms in our case. We need to compute the derivative of this function with respect to each of the features, so in effect we will be doing 10000 \* 10 = 100,000 computations per iteration. It is common to take 1000 iterations, in effect we have 100,000 \* 1000 = 100000000 computations to complete the algorithm. That is pretty much an overhead and hence gradient descent is slow on huge data.

In this method one training sample (example) is passed through the neural network at a time and the parameters (weights) of each layer are updated with the computed gradient. So, at a time a single training sample is passed through the network and its corresponding loss is computed. The parameters of all the layers of the network are updated after every training sample.



Here, (Theta(j)) corresponds to the parameter, (alpha) is the learning rate that is the step size multiplied by the derivative of the function by which to move on the loss function curve toward the minima.

## **Advantages and Disadvantages :**

**Advantages :**

* It is easier to fit into memory due to a single training sample being processed by the network.
* It is computationally fast as only one sample is processed at a time.
* For larger datasets it can converge faster as it causes updates to the parameters more frequently.
* Due to frequent updates the steps taken towards the minima of the loss function have oscillations which can help getting out of local minimums of the loss function (in case the computed position turns out to be the local minimum).

## **Disadvantages :**

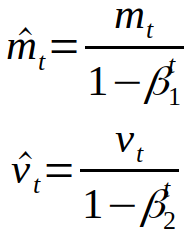
* Due to frequent updates the steps taken towards the minima are very noisy. This can often lead the gradient descent into other directions.
* Also, due to noisy steps it may take longer to achieve convergence to the minima of the loss function.
* Frequent updates are computationally expensive due to using all resources for processing one training sample at a time.
* It loses the advantage of vectorized operations as it deals with only a single example at a time.

**3. Adam Optimizer :**

Adam is an adaptive learning rate optimization algorithm that’s been designed specifically for training deep neural networks. Adam actually finds worse solution than [stochastic gradient descent](https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d). A lot of research has been done to address the problems of Adam.

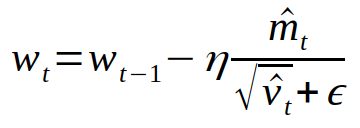
Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum. Let’s take a closer look at how it works.

Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters. Its name is derived from adaptive [moment](https://en.wikipedia.org/wiki/Moment_(mathematics)) estimation, and the reason it’s called that is because Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network.

****

**M(t) and V(t)** are values of the first moment which is the Mean and the second moment which is the uncentered variance of the gradients respectively.

The only thing left to do is to use those moving averages to scale learning rate individually for each parameter. The way it’s done in Adam is very simple, to perform weight update we do the following:

****

Where w is model weights, eta (look like the letter n) is the learning rate (it can depend on iteration). And that’s it, that’s the update rule for Adam.

**Advantages and Disadvantages :**

**Advantages :**

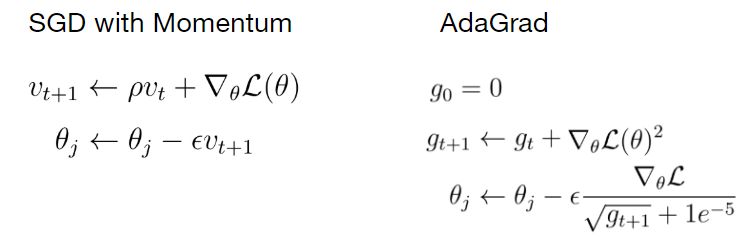
* The method is too fast and converges rapidly.
* Rectifies vanishing learning rate, high variance.

**Disadvantages:**

Computationally costly.

**4. AdaGrad :**

Another optimization strategy is called AdaGrad. Adagrad is an adaptive gradient algorithm. The idea is that you keep the running sum of squared gradients during optimization. In this case, we have no momentum term, but an expression g that is the sum of the squared gradients.

****

When we update a weight parameter, we divide the current gradient by the root of that term g. To explain the intuition behind AdaGrad, imagine a loss function in a two-dimensional space in which the gradient of the loss function in one direction is very small and very high in the other direction.

Summing up the gradients along the axis where the gradients are small causes the squared sum of these gradients to become even smaller. If during the update step, we divide the current gradient by a very small sum of squared gradients g, the result of that division becomes very high and vice versa for the other axis with high gradient values.

**Advantages and Disadvantages :**

**Advantages :**

* It eliminates the need to manually tune the learning rate.
* convergence is faster and more reliable – than simple SGD when the scaling of the weights is unequal.
* It is not very sensitive to the size of the master step.

**Disadvantages :**

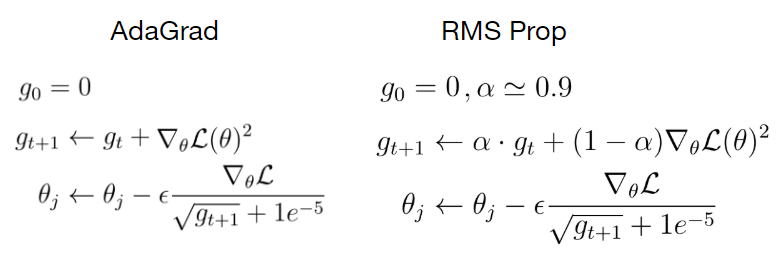
The disadvantage causes the learning rate to shrink and become infinitesimally small.The algorithm can no longer acquire additional knowledge.

The following algorithms aim to resolve this flaw :

* Adadelta
* RMSprop
* Adam

**5. RMSProp :**

There is a slight variation of AdaGrad called RMSProp that addresses the problem that AdaGrad has. With RMSProp we still keep the running sum of squared gradients but instead of letting that sum grow continuously over the period of training we let that sum actually decay.



In RMSProp we multiply the sum of squared gradients by a decay rate **α**and add the current gradient weighted by (1- **α)**. The update step in the case of RMSProp looks exactly the same as in AdaGrad where we divide the current gradient by the sum of squared gradients to have this nice property of accelerating the movement along the one dimension and slowing down the movement along the other dimension.