TYPES OF OPTIMIZERS :

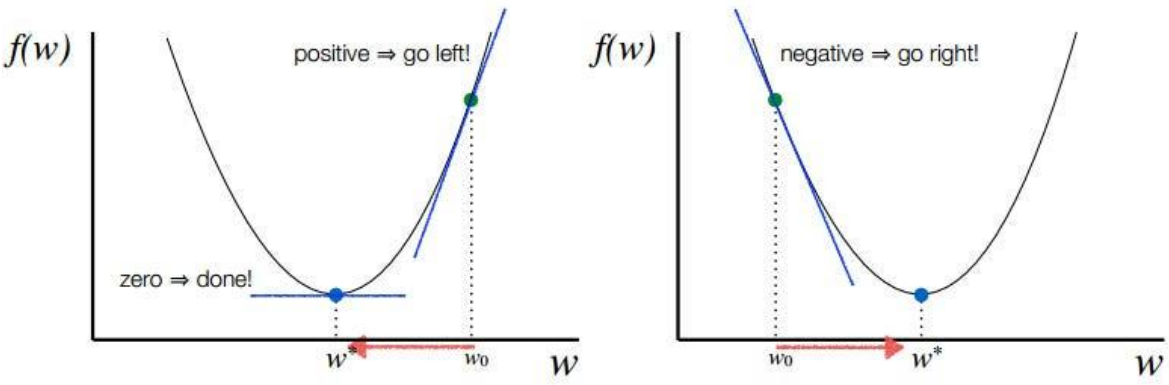
1. Gradient Descent
2. Stochastic Gradient Descent
3. Adagrad
4. Adadelta
5. RMSprop
6. Adam

G***radient Descent :***

This is one of the oldest and the most common optimizer used in neural networks, best for the cases where the data is arranged in a way that it possesses a convex optimization problem. It will try to find the least cost function value by updating the weights of your learning algorithm and will come up with the best-suited parameter values corresponding to the Global Minima.

This is done by moving down the hill with a negative slope, increasing the older weight, and positive slope reducing the older weight.

Press enter or click to view image in full size



Deciding the direction of descent

Although there are challenges while using this optimizer, suppose the data is arranged in a way that it possesses a non-convex optimization problem then it can possibly land on the Local Minima instead of the Global Minima thereby providing the parameter values with a higher cost function.

There is also a saddle point problem. This is a point where the gradient is zero but is not an optimal point. This is still an active area of research.

For certain cases, problems like Vanishing Gradient or Exploding Gradient may also occur due to incorrect parameter initialization. These problems occur due to a very small or very large gradient, which makes it difficult for the algorithm to converge.

S***tochastic Gradient Descent :***

This is another variant of the Gradient Descent optimizer with an additional capability of working with the data with a non-convex optimization problem. The problem with such data is that the cost function results to rest at the local minima which are not suitable for your learning algorithm.

Rather than going for batch processing, this optimizer focuses on performing one update at a time. It is therefore usually much faster, also the cost function minimizes after each iteration (EPOCH). It performs frequent updates with a high variance that causes the objective function(cost function) to fluctuate heavily. Due to which it makes the gradient to jump to a potential Global Minima.

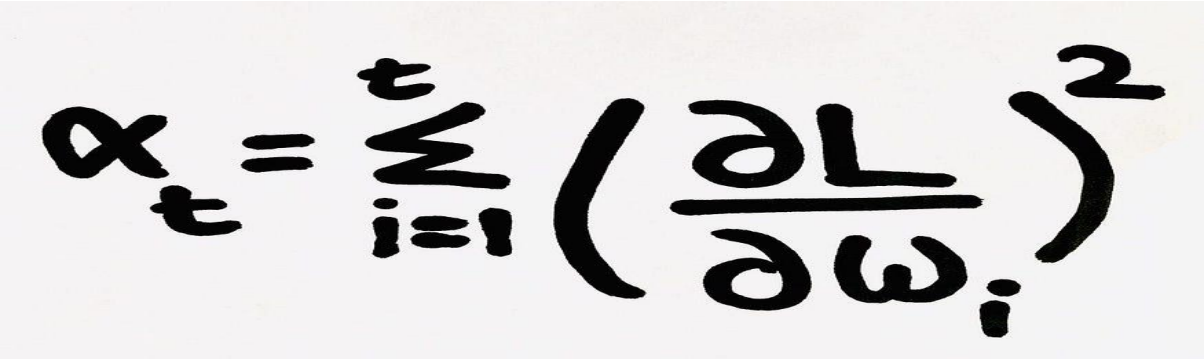
However, if we choose a learning rate that is too small, it may lead to very slow convergence, while a larger learning rate can make it difficult to converge and cause the cost function to fluctuate around the minimum or even to diverge away from the global minima.

A***dagrad :***

This is the Adaptive Gradient optimization algorithm, where the learning rate plays an important role in determining the updated parameter values. Unlike Stochastic Gradient descent, this optimizer uses a different learning rate for each iteration(EPOCH) rather than using the same learning rate for determining all the parameters.

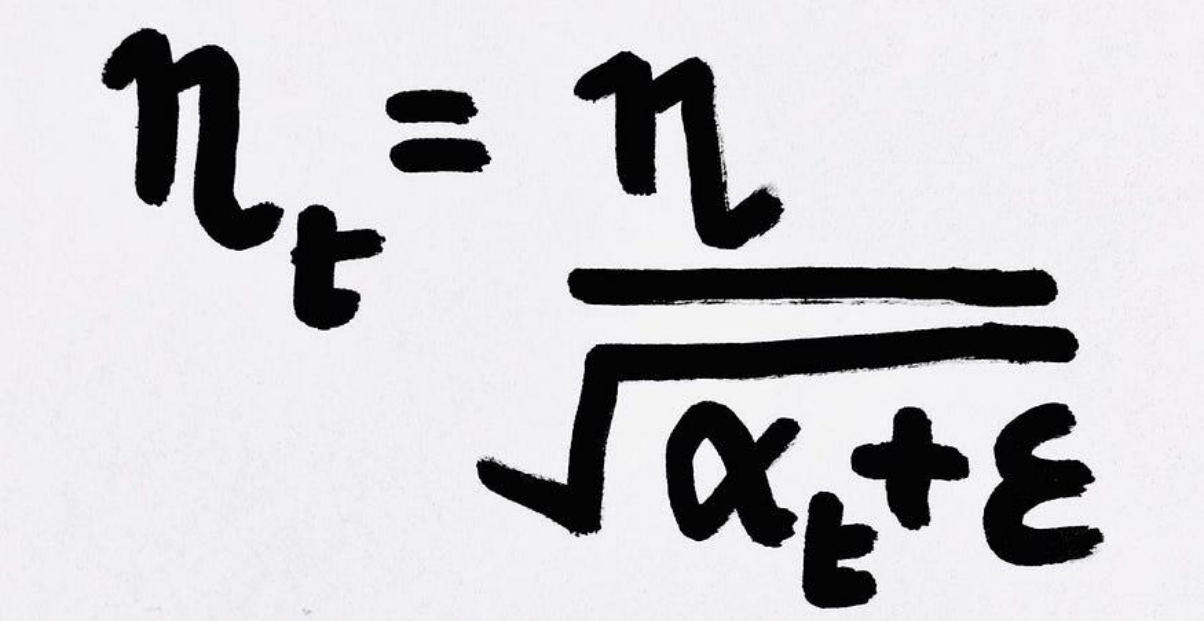
Thus it performs smaller updates(lower learning rates) for the weights corresponding to the high-frequency features and bigger updates(higher learning rates) for the weights corresponding to the low-frequency features, which in turn helps in better performance with higher accuracy. Adagrad is well-suited for dealing with sparse data.

So at each iteration, first the alpha at time t will be calculated and as the iterations increase the value of t increases, and thus alpha t will start increasing.



The alpha term increases because of squaring the derivative of loss with respect to the weight at time step t

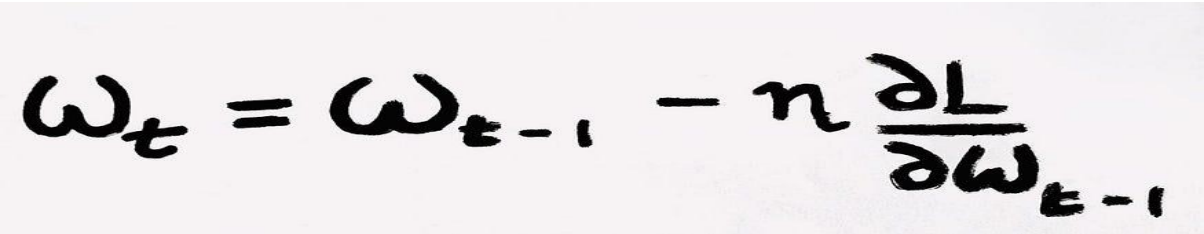
Now the learning rate is calculated at each time step. Given by,



Here epsilon is a small positive number added to alpha to avoid the error if at any instance alpha becomes zero

Therefore as the alpha at time step t increases, it makes the learning rate to decrease gradually.

As the learning rate changes for each iteration, the formula for updating the weight also changes. Given by,



This determines the updated weight for each iteration

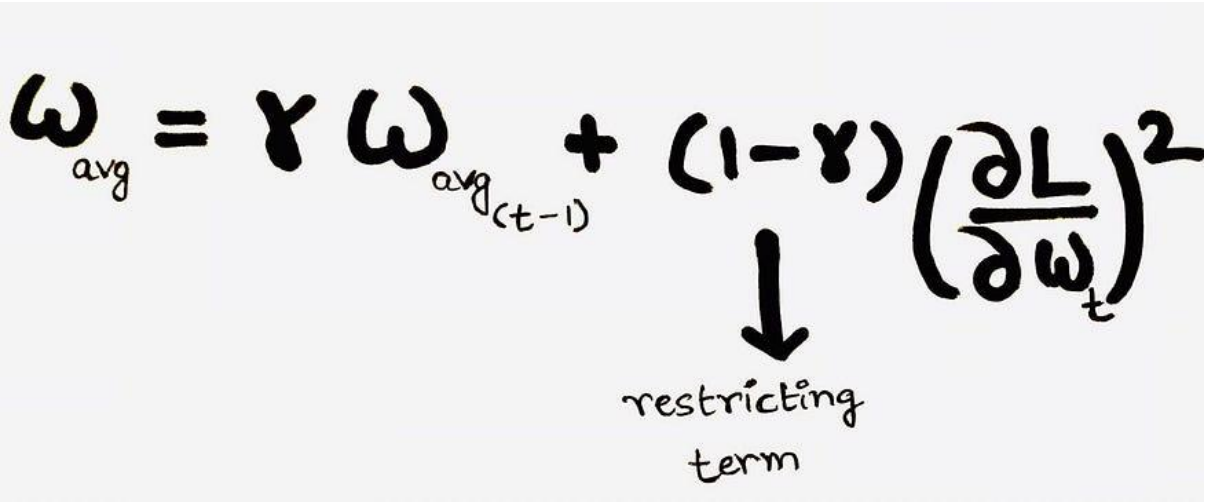
However, there is a disadvantage of getting into the problem of *Vanishing Gradient* because after a lot of iterations the alpha value becomes very large making the learning rate very small leading to no change between the new and the old weight. This in turn causes the learning rate to shrink and eventually become very small, where the algorithm is not able to acquire any further knowledge.

A*dadelta :*

This is an extension of the Adaptive Gradient optimizer, taking care of its aggressive nature of reducing the learning rate infinitesimally. Here instead of using the previous squared gradients, the sum of gradients is defined as a reducing weighted average of all past squared gradients(weighted averages) this restricts the learning rate to reduce to a very small value.

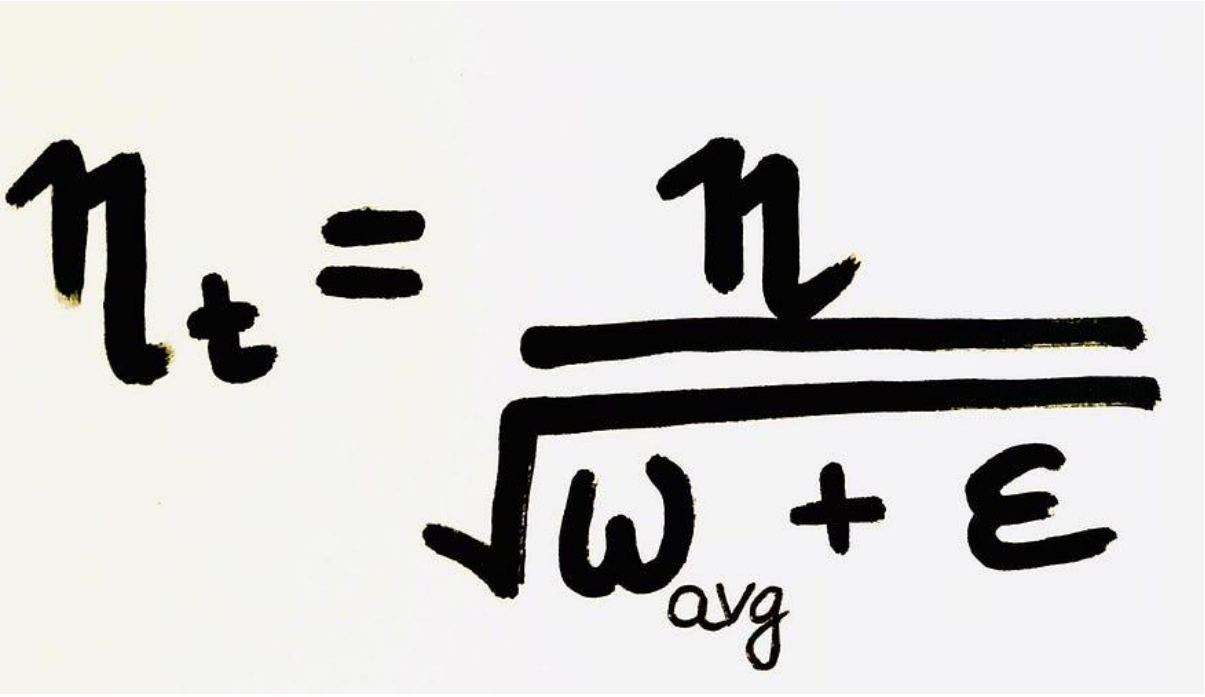
The formula for the new weight remains the same as in Adagrad. However, there are some changes in determining the learning rate at time step t for each iteration.

At each iteration, first the weighted average is calculated. Where we have the restricting term(gamma = 0.95) which helps in avoiding the problem of *Vanishing Gradient*.



Formula for calculating the Weighted average

After which the Learning rate is calculated using the formula,


Formula for calculating the learning rate at time step t

Thus because of the restricting term, the weighted average will increase at a slower rate, making the learning rate to reduce slowly to reach the global minima.

RMSprop :

Both the optimizing algorithms, RMSprop(Root Mean Square Propagation) and Adadelta were developed around the same time, for the same purpose to resolve Adagrad's problem of destructive learning rates. However, both use the same method which utilizes an Exponential Weighted Average to determine the learning rate at time t for each iteration.

RMSprop is an adaptive learning rate method proposed by Geoffrey Hinton, which appropriately divides the learning rate by an exponentially weighted average of squared gradients. It is suggested to set gamma at 0.95, as it has been showing good results for most of the cases.

Adam :

This is the Adaptive Moment Estimation algorithm which also works on the method of computing adaptive learning rates for each parameter at every iteration. It uses a combination of *Gradient Descent with Momentum* and RMSprop to determine the parameter values.

When introducing the algorithm, there was a list of attractive benefits of using Adam on non-convex optimization problems which made it the most commonly used optimizer.

It comes with several advantages combining the benefits of both Gradient with Momentum and RMSProp like low memory requirements, appropriate for non-stationary objectives, works best with large data and parameters with efficient computation. This works using the same methodology of adaptive learning rate in addition to storing an exponential weighted average of the past squared derivative of loss with respect to the weight at time t-1.

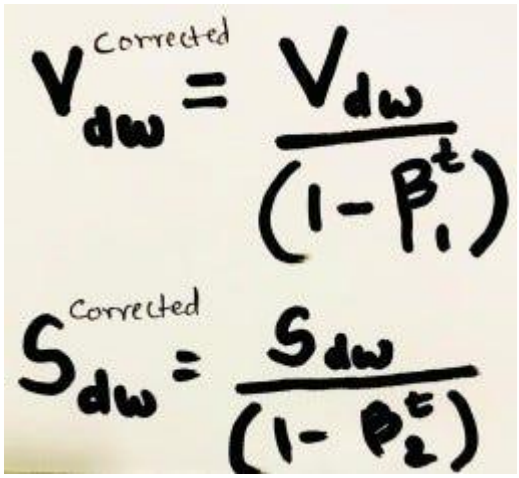
It comes with several parameters, which are β_1, β_2, and ε (epsilon). Where β_1 and β_2 are the initial restricting parameters for Momentum and RMSprop respectively. Here, β_1 corresponds to the first moment and β_2 corresponds to the second moment.

For updating the weights with an adaptive learning rate at iteration t, first, we need to calculate the first and second moment given by the following formulae,

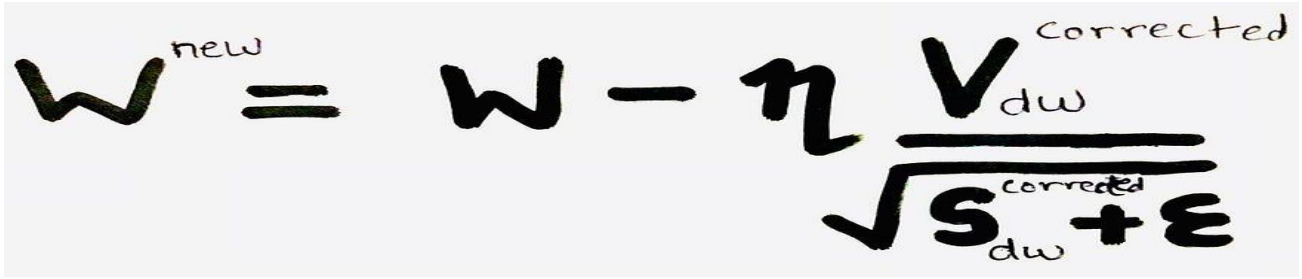
$VdW = \beta_1 \times VdW + (1-\beta_1) \times dW$ — — GD with Momentum (1st)

$SdW = \beta_2 \times SdW + (1-\beta_2) \times dW^2$ — — RMSprop (2nd)

The corrected VdW and SdW is given by,



Therefore the new weight will be updated using the formula,



The initial value of n is to be tuned for better results.

Adam is relatively easy to configure where the default configuration parameters do well on most problems. It is proposed to have default values of β1=0.9 ,β2 = 0.999 . Studies show that Adam works well in practice, in comparison to other adaptive learning algorithms.