* **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.

1. **Gradient Descent**

Gradient Descent is the most basic but most used optimization algorithm. It’s used heavily in linear regression and classification algorithms. Back propagation 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.

algorithm: θ=θ−α⋅∇J(θ)

Advantages:

1. Easy computation and implement.
2. Easy to understand.

Disadvantages:

1.May trap at local minima.

2.Weights are changed after calculating gradient on the whole dataset. So, if the dataset is too large than this may take years to converge to the minima.

3.Requires large memory to calculate gradient on the whole dataset.

**2.Stochastic Gradient Descent:**

It’s a variant of Gradient Descent. It tries to update the model’s parameters more frequently. In this, the model parameters are altered after computation of loss on each training example. So, if the dataset contains 1000 rows SGD will update the model parameters 1000 times in one cycle of dataset instead of one time as in Gradient Descent.

θ=θ−α⋅∇J(θ;x(i);y(i)) , where {x(i) ,y(i)} are the training examples.

As the model parameters are frequently updated parameters have high variance and fluctuations in loss functions at different intensities.

Advantages:

1. Frequent updates of model parameters hence, converges in less time.
2. Requires less memory as no need to store values of loss functions.
3. May get new minima’s.

Disadvantages:

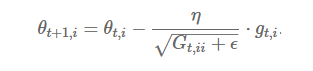
1. High variance in model parameters.
2. May shoot even after achieving global minima.
3. To get the same convergence as gradient descent needs to slowly reduce the value of learning rate.

**3.Adagrad**

One of the disadvantages of all the optimizers explained is that the learning rate is constant for all parameters and for each cycle. This optimizer changes the learning rate. It changes the learning rate ‘η’ for each parameter and at every time step ‘t’. It’s a type second order optimization algorithm. It works on the derivative of an error function.

Image for post

A derivative of loss function for given parameters at a given time t.



Update parameters for given input i and at time/iteration t

η is a learning rate which is modified for given parameter θ(i) at a given time based on previous gradients calculated for given parameter θ(i).

We store the sum of the squares of the gradients w.r.t. θ(i) up to time step t, while ϵ is a smoothing term that avoids division by zero (usually on the order of 1e−8). Interestingly, without the square root operation, the algorithm performs much worse.

It makes big updates for less frequent parameters and a small step for frequent parameters.

Advantages:

1. Learning rate changes for each training parameter.
2. Don’t need to manually tune the learning rate.
3. Able to train on sparse data.

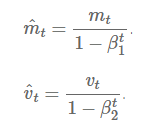
Disadvantages:

1. Computationally expensive as a need to calculate the second order derivative.
2. The learning rate is always decreasing results in slow training.

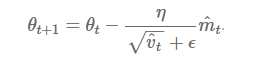
# 4.Adam

[Adam](https://arxiv.org/pdf/1412.6980.pdf) (Adaptive Moment Estimation) works with momentums of first and second order. The intuition behind the Adam is that we don’t want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search. In addition to storing an exponentially decaying average of past squared gradients like AdaDelta, *Adam*also keeps an exponentially decaying average of past gradients M(t).

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 gradientsrespectively.



First and second order of momentum

Here, we are taking mean of **M(t)** and **V(t)** so that **E[m(t)]** can be equal to **E[g(t)]** where **E[f(x)]** is an expected value of **f(x)**.Update the parameter

Advantages:

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

Disadvantages:

1.Computationally costly.

**5.Mini Batch Gradient Descent**

We have seen the Batch Gradient Descent. We have also seen the Stochastic Gradient Descent. Batch Gradient Descent can be used for smoother curves. SGD can be used when the dataset is large. Batch Gradient Descent converges directly to minima. SGD converges faster for larger datasets. But, since in SGD we use only one example at a time, we cannot implement the vectorized implementation on it. This can slow down the computations. To tackle this problem, a mixture of Batch Gradient Descent and SGD is used.

Neither we use all the dataset all at once nor we use the single example at a time. We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch. Doing this helps us achieve the advantages of both the former variants we saw. So, after creating the mini-batches of fixed size, we do the following steps in one epoch:

1.Pick a mini-batch

2.Feed it to Neural Network

1. Calculate the mean gradient of the mini-batch
2. Use the mean gradient we calculated in step 3 to update the weights
3. Repeat steps 1–4 for the mini-batches we created
4. Just like SGD, the average cost over the epochs in mini-batch gradient descent fluctuates because we are averaging a small number of examples at a time.