TECHNEEDS

WEEK-5

NOTES

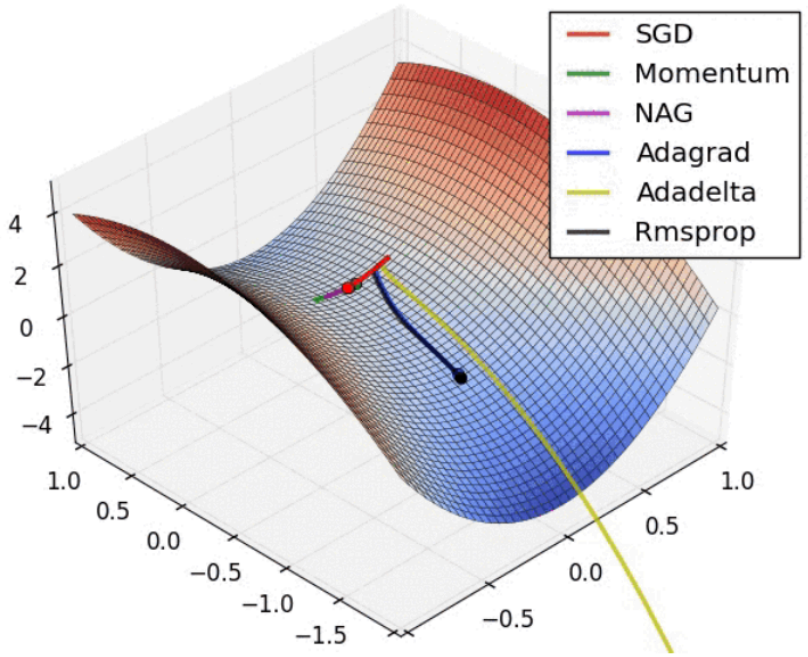
Topic Covered:

- **What is optimization?**
- **How does the optimizer works?**
- **Classification of Optimization Techniques**
- **Gradient Descent**
- **Batch Normalization**

Mind Map of Optimizer

What is optimization?

In machine learning, optimizers and loss functions are two components that help improve the performance of the model. By calculating the difference between the expected and actual outputs of a model, a loss function evaluates the effectiveness of a model. Among the loss functions are log loss, hinge loss, and mean square loss. By modifying the model's parameters to reduce the loss function value, the optimizer contributes to its improvement. RMSProp, ADAM, and SGD are a few examples of optimizers. The optimizer's job is to determine which combination of the neural network's weights and biases will give it the best chance to generate accurate predictions.



How does the optimizer work?

To understand working of optimizers just think of a situation...
You are somewhere on top of a mountain with a blindfold on. Now you have to get down from the mountain to reach your home safely.

What will be your approach...???
You have to take one step at a time and if you are going down then it makes you make some progress but if you are going up then you are going in the wrong direction.
So you will start doing hit and trial and take steps that will lead you downwards.

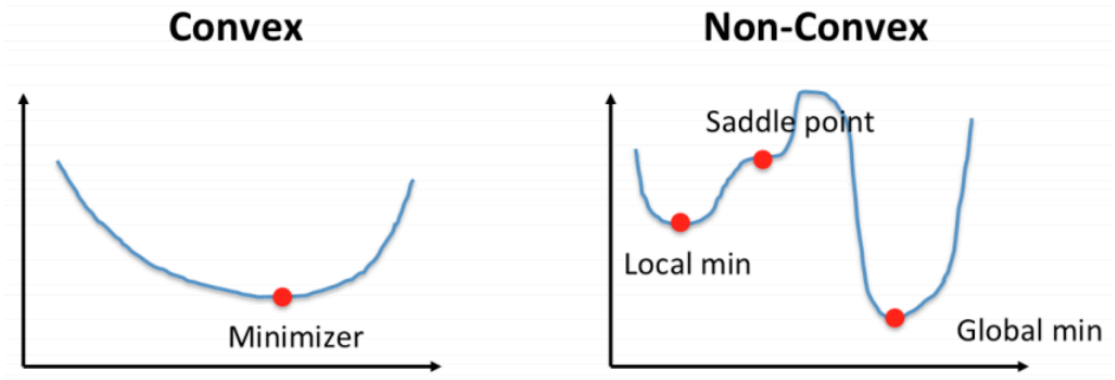
Similarly our optimization techniques works. Initially we don't know the weights so we start randomly but with some trial and error based on loss function we can end up getting our loss downwards.

Optimization techniques are responsible for reducing the loss and provide the most accurate results possible.

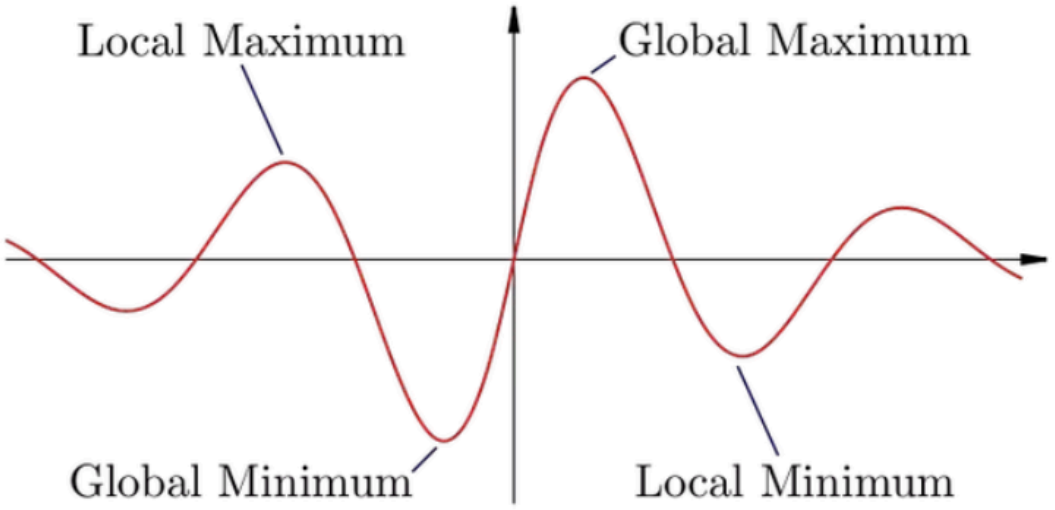
There are various optimization techniques, we'll learn about different types of optimizers and how do they work to minimize loss.

Gradient Descent

Gradient Descent is one of the popular techniques to perform optimization. It's based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum.



Convex vs Non-Convex



Local and Global Minimum & Maximum

Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function.
We start by defining initial parameter's values and from there gradient descent uses calculus to iteratively adjust the values so they minimize the given cost-function.

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Repeat Until Convergence

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is an extension of Gradient Descent, where it overcomes some of the disadvantages of Gradient Descent algorithm. SGD tries to overcome the disadvantage of computationally intensive by computing the derivative of one point at a time. Due to this fact, SGD takes more number of iterations compared to GD to reach minimum and also contains some noise when compared to Gradient Descent. As SGD computes derivatives of only 1 point at a time, the time taken to complete one epoch is large compared to Gradient Descent algorithm.

Mini Batch — Stochastic Gradient Descent

MB-SGD is an extension of SGD algorithm. It overcomes the time-consuming complexity of SGD by taking a batch of points / subset of points from dataset to compute derivative.

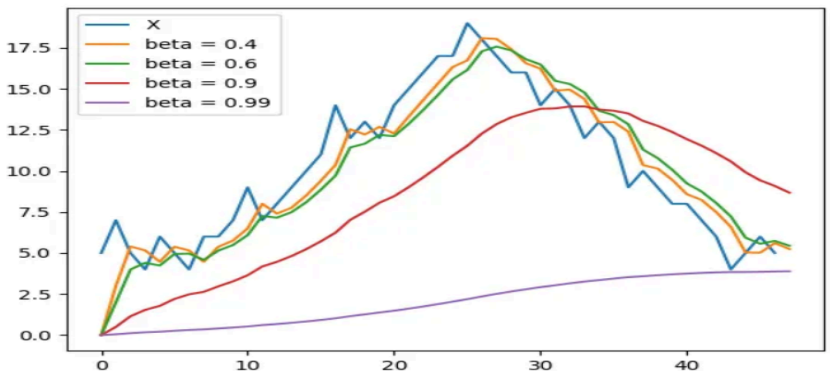
Note- *It is observed that the derivative of loss function of MB-SGD is similar to the loss function of GD after some iterations. But the number iterations to achieve minima in MB-SGD is large compared to GD and is computationally expensive. The update of weights in much noisier because the derivative is not always towards minima.*

In recent times Adaptive Optimization Algorithms are gaining popularity due to their ability to converge swiftly. These algorithms use statistics from previous iterations to speed up the process of convergence.

Momentum based Optimizer

It is an adaptive optimization algorithm which exponentially uses weighted average gradients over previous iterations to stabilize the convergence, resulting in quicker optimization. This is done by adding a fraction (gamma) to the previous iteration values. Essentially the momentum term increase when the gradient points are in the same directions and reduce when gradients fluctuate. As a result, the value of loss function converges faster than expected.

> **Cost function:** $V(t)=\gamma V(t-1)+\alpha.\nabla J(\theta)$
>
> $\theta=\theta - V(t)$



For different values of B, the performance of algorithm is plotted

Nesterov Accelerated Gradient (NAG)

In momentum-based optimization, the current gradient takes the next step based on previous iteration values. But we need a much smarter algorithm that knows when to intuitively stop so that the gradient doesn't further increase. To do this the algorithm should have an approximate idea of the parameter values in its next iteration. In doing so we can efficiently look ahead by calculating the gradient values wrt to future position of parameters.

From the previous equation, we know that momentum includes the term **[γV(t−1)]** to calculate the value of previous iterations. Computing (**θ-γV(t−1)**) gives us an approximation of next position of parameters[**θ**]. We can now conclusively look ahead of current parameters by approximating future position with the help of below equation.

> **Cost function:** $V(t)=\gamma V(t-1)+\alpha . \nabla J[\theta-\gamma V(t-1)]$
>
> $\theta=\theta - V(t)$

AdaGrad

Adaptive Gradient as the name suggests adopts the learning rate of parameters by updating it at each iteration depending on the position it is present, i.e- by adapting slower learning rates when features are occurring frequently and adapting higher learning rate when features are infrequent.

Technically it acts on learning rate parameter by dividing the learning rate by the square root of gamma, which is the summation of all gradients squared.

In the update rule, AdaGrad modifies the general learning rate N at each step for all the parameters based on past computations. One of

the biggest disadvantages is the accumulation of squared gradients in the denominator. Since every added term is positive, the accumulated sum keeps growing during the training. This makes the learning rate to shrink and eventually become small. This method is not very sensitive to master step size and also converges faster.

AdaDelta

It is simply an extension of AdaGrad that seeks to reduce its monotonically decreasing learning rate. Instead of summing all the past gradients, AdaDelta restricts the no. of summation values to a limit (w). In AdaDelta, the sum of past gradients (w) is defined as “Decaying Average of all past squared gradients”. The current average at the iteration then depends only on the previous average and current gradient.

RMSProp

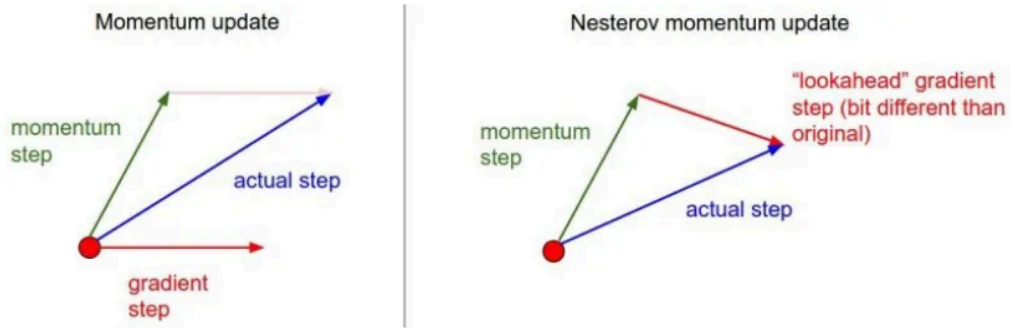
Root Mean Squared Prop is another adaptive learning rate method that tries to improve AdaGrad. Instead of taking cumulative sum of squared gradients like in AdaGrad, we take the exponential moving average. The first step in both AdaGrad and RMSProp is identical. RMSProp simply divides learning rate by an exponentially decaying average.

Adaptive Moment Estimation (Adam)

It is a combination of RMSProp and Momentum. This method computes adaptive learning rate for each parameter. In addition to storing the previous decaying average of squared gradients, it also holds the average of past gradient similar to Momentum. Thus, Adam behaves like a heavy ball with friction which prefers flat minima in error surface.

$$w_{t+1} = (1-\lambda) w_t - \eta \nabla f_t(w_t)$$

weight decay update



Batch Normalization

Batch normalization is a deep learning approach that has been shown to significantly improve the efficiency and reliability of neural network models. It is particularly useful for training very deep networks, as it can help to reduce the internal covariate shift that can occur during training.

Batch normalization is a technique used to improve the performance of a deep learning network by first removing the batch mean and then splitting it by the batch standard deviation.

Stochastic gradient descent is used to rectify this standardization if the loss function is too big, by shifting or scaling the outputs by a parameter, which in turn affects the accuracy of the weights in the following layer.