Optimizers in Deep Learning

What is Optimizer?

- Optimizer is a crucial element that fine-tunes a neural network's parameters during training.
- Its primary role is to minimize the model's error or loss function, enhancing performance.
- Various optimization algorithms, known as optimizers, employ distinct strategies to converge towards optimal parameter values for improved predictions efficiently.
- Well-known optimizers in deep learning encompass Stochastic Gradient Descent (SGD), Adam, and RMSprop.
- Optimizer algorithms are optimization method that helps improve a deep learning model's performance.
- These optimization algorithms or optimizers widely affect the accuracy and speed training of the deep learning model.
- While training the deep learning optimizers model, modify each epoch's weights and minimize the loss function.
- An optimizer is a function or an algorithm that adjusts the attributes of the neural network, such as weights and learning rates. Thus, it helps in reducing the overall loss and improving accuracy.
- The problem of choosing the right weights for the model is a daunting task, as a deep learning model generally consists of millions of parameters.
- It raises the need to choose a suitable optimization algorithm for your application.
- Hence understanding these machine learning algorithms is necessary for data scientists before having a deep dive into the field.
- You can use different optimizers in the machine learning model to change your weights and learning rate.

Important Deep Learning Terms

Before proceeding, there are a few terms that you should be familiar with.

- **Epoch** – The number of times the algorithm runs on the whole training dataset.
- **Sample** – A single row of a dataset.
- **Batch** – It denotes the number of samples to be taken to for updating the model parameters.
- **Learning rate** – It is a parameter that provides the model a scale of how much model weights should be updated.
- **Cost Function/Loss Function** – A cost function is used to calculate the cost, which is the difference between the predicted value and the actual value.
- **Weights/ Bias** – The learnable parameters in a model that controls the signal between two neurons.

Optimization Techniques popularly used in Deep Learning

> · Gradient Descent
>
> · Stochastic Gradient Descent (SGD)
>
> · Mini-Batch Stochastic Gradient Descent (MB — SGD)
>
> · SGD with Momentum
>
> · Nesterov Accelerated Gradient (NAG)
>
> · Adaptive Gradient (AdaGrad)
>
> · AdaDelta
>
> · RMSProp
>
> · Adam
>
> · Nadam

Gradient Descent

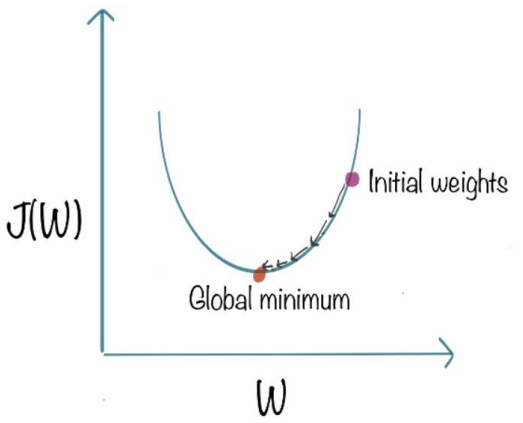
- Gradient Descent is a widely used optimization algorithm in deep learning.
- It utilizes calculus to iteratively adjust the model parameters towards the direction of the steepest decrease in the loss function, aiming to reach a local minimum.
- Imagine holding a ball at the top of a bowl. The gradient represents the direction of the steepest slope, akin to the direction the ball would roll down the bowl.
- In mathematical terms, the gradient indicates the rate of change of the loss function with respect to the model parameters.

Formula:

- The basic formula for Gradient Descent can be represented as:

  $\theta = \theta - \alpha \cdot \nabla J(\theta)$

  Where:
  - θ represents the model parameters.
  - α (alpha) is the learning rate, determining the size of the steps taken in each iteration.
  - $\nabla J(\theta)$ denotes the gradient of the loss function J with respect to the parameters θ.



Working Process:

- It initiates with initial coefficients and computes the cost.
- It then adjusts the coefficients towards lower costs by updating them based on the calculated gradients.
- This process repeats iteratively until a local minimum of the loss function is reached, indicating convergence.

Pros and Cons:

- *Advantages*:
  - Simple and intuitive conceptually.
  - Effective for convex functions and many practical deep learning problems.
- *Challenges*:
  - Computationally expensive for large datasets due to frequent gradient computations.
  - May struggle with non-convex functions as it can get stuck in local minima or plateaus.

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

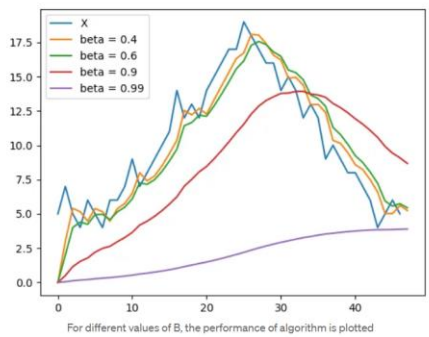
- MB-SGD is an extension of SGD algorithm.
- It overcomes the time-consuming complexity of SGD by taking a batch of points / subset of points from dataset to compute derivative.
- Note- It is observed that the derivative of loss function of MB-SGD is similar to the loss function of GD after some iterations. 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.

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

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.
- 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.

$$\mathrm{w_t} = \mathrm{w_{t-1}} - \eta'_{\mathrm{t}} \frac{\partial \mathrm{L}}{\partial \mathrm{w(t-1)}}$$

$$\eta'_{\mathrm{t}} = \frac{\eta}{\mathrm{sqrt}(\alpha_{\mathrm{t}} + \epsilon)}$$

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

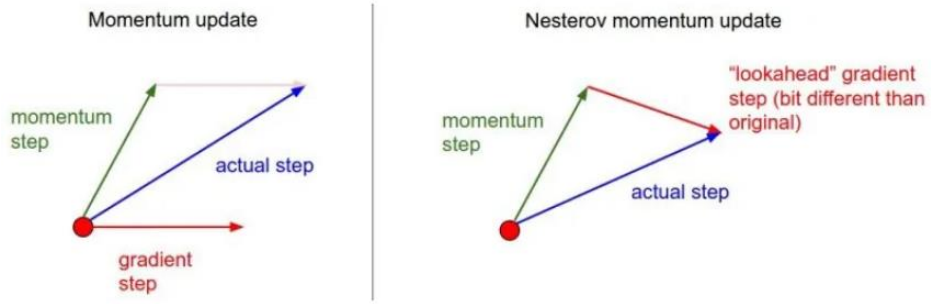
- Root Mean Squared Prop is another adaptive learning rate method that tries to improve AdaGrad. Instead of taking cumulative sum of squared gradients like in AdaGrad, we take the exponential moving average.
- The first step in both AdaGrad and RMSProp is identical.
- RMSProp simply divides learning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



Optimizer	When to Use
Gradient Descent	Small datasets, simple models, convex optimization problems
Stochastic Gradient Descent (SGD)	Large datasets, faster convergence, when computational resources are limited
Mini-Batch SGD	Balanced approach between GD and SGD, suitable for medium-sized datasets
SGD with Momentum	Faster convergence, smoother optimization, deals with noisy or sparse gradients
Nesterov Accelerated Gradient	Accelerates SGD in relevant directions, useful for minimizing oscillations and reaching optima
Adaptive Gradient (AdaGrad)	Sparse data, learning rate adaptation for individual parameters, stable convergence
AdaDelta	Adaptive learning rate, no manual tuning of learning rate hyperparameter
RMSProp	Dealing with non-stationary or noisy problems, suitable for RNNs
Adam	Wide range of applications, efficient for large-scale datasets, handles sparse gradients effectively
Nadam	Combination of Adam and Nesterov momentum, effective in practice for various deep learning scenarios