All Types of Optimizers

What is Optimizer?

In deep learning, an 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.

Gradient Descent Deep Learning Optimizer

Gradient Descent can be considered the popular kid among the class of optimizers in deep learning. This optimization algorithm uses calculus to consistently modify the values and achieve the local minimum. Before moving ahead, you might question what a gradient is.

In simple terms, consider you are holding a ball resting at the top of a bowl. When you lose the ball, it goes along the steepest direction and eventually settles at the bottom of the bowl. A Gradient provides the ball in the steepest direction to reach the local minimum which is the bottom of the bowl.

Gradient descent works as follows:

1. It starts with some coefficients, sees their cost, and searches for cost value lesser than what it is now.
2. It moves towards the lower weight and updates the value of the coefficients.
3. The process repeats until the local minimum is reached. A local minimum is a point beyond which it cannot proceed.

Stochastic Gradient Descent Deep Learning Optimizer

At the end of the previous section, you learned why there might be better options than using gradient descent on massive data. To tackle the challenges large datasets pose, we have stochastic gradient descent, a popular approach among optimizers in deep learning. The term stochastic denotes the element of randomness upon which the algorithm relies. In stochastic gradient descent, instead of processing the entire dataset during each iteration, we randomly select batches of data. This implies that only a few samples from the dataset are considered at a time, allowing for more efficient and computationally feasible optimization in deep learning models.

The procedure is first to select the initial parameters w and learning rate n. Then randomly shuffle the data at each iteration to reach an approximate minimum.

Since we are not using the whole dataset but the batches of it for each iteration, the path taken by the algorithm is full of noise as compared to the gradient descent algorithm. Thus, SGD uses a higher number of iterations to reach the local minima. Due to an increase in the number of iterations, the overall computation time increases. But even after increasing the number of iterations, the computation cost is still less than that of the gradient descent optimizer. So the conclusion is if the data is enormous and computational time is an essential factor, stochastic gradient descent should be preferred over batch gradient descent algorithm.

Mini Batch Gradient Descent Deep Learning Optimizer

In this variant of gradient descent, instead of taking all the training data, only a subset of the dataset is used for calculating the loss function. Since we are using a batch of data instead of taking the whole dataset, fewer iterations are needed. That is why the mini-batch gradient descent algorithm is faster than both stochastic gradient descent and batch gradient descent algorithms. This algorithm is more efficient and robust than the earlier variants of gradient descent. As the algorithm uses batching, all the training data need not be loaded in the memory, thus making the process more efficient to implement. Moreover, the cost function in mini-batch gradient descent is noisier than the batch gradient descent algorithm but smoother than that of the stochastic gradient descent algorithm. Because of this, mini-batch gradient descent is ideal and provides a good balance between speed and accuracy.

Despite all that, the mini-batch gradient descent algorithm has some downsides too. It needs a hyperparameter that is “mini-batch-size”, which needs to be tuned to achieve the required accuracy. Although, the batch size of 32 is considered to be appropriate for almost every case. Also, in some cases, it results in poor final accuracy. Due to this, there needs a rise to look for other alternatives too.

Adagrad (Adaptive Gradient Descent) Deep Learning Optimizer

The adaptive gradient descent algorithm is slightly different from other gradient descent algorithms. This is because it uses different learning rates for each iteration. The change in learning rate depends upon the difference in the parameters during training. The more the parameters get changed, the more minor the learning rate changes. This modification is highly beneficial because real-world datasets contain sparse as well as dense features. So it is unfair to have the same value of learning rate for all the features.

The benefit of using Adagrad is that it abolishes the need to modify the learning rate manually. It is more reliable than gradient descent algorithms and their variants, and it reaches convergence at a higher speed.

One downside of the AdaGrad optimizer is that it decreases the learning rate aggressively and monotonically. There might be a point when the learning rate becomes extremely small. This is because the squared gradients in the denominator keep accumulating, and thus the denominator part keeps on increasing. Due to small learning rates, the model eventually becomes unable to acquire more knowledge, and hence the accuracy of the model is compromised.

RMS Prop (Root Mean Square) Deep Learning Optimizer

RMS prop is one of the popular optimizers among deep learning enthusiasts. This is maybe because it hasn't been published but is still very well-known in the community. RMS prop is ideally an extension of the work RPPROP. It resolves the problem of varying gradients. The problem with the gradients is that some of them were small while others may be huge. So, defining a single learning rate might not be the best idea. RPPROP uses the gradient sign, adapting the step size individually for each weight. In this algorithm, the two gradients are first compared for signs. If they have the same sign, we're going in the right direction, increasing the step size by a small fraction. If they have opposite signs, we must decrease the step size. Then we limit the step size and can now go for the weight update.

The problem with RPPROP is that it doesn't work well with large datasets and when we want to perform mini-batch updates. So, achieving the robustness of RPPROP and the efficiency of mini-batches simultaneously was the main motivation behind the rise of RMS prop. RMS prop is an advancement in AdaGrad optimizer as it reduces the monotonically decreasing learning rate.

AdaDelta Deep Learning Optimizer

AdaDelta can be seen as a more robust version of the AdaGrad optimizer. It is based upon adaptive learning and is designed to deal with significant drawbacks of AdaGrad and RMS prop optimizer. The main problem with the above two optimizers is that the initial learning rate must be defined manually. One other problem is the decaying learning rate which becomes infinitesimally small at some point. Due to this, a certain number of iterations later, the model can no longer learn new knowledge.

To deal with these problems, AdaDelta uses two state variables to store the leaky average of the second moment gradient and a leaky average of the second moment of change of parameters in the model.

Adam Optimizer in Deep Learning

Adam optimizer, short for Adaptive Moment Estimation optimizer, is an optimization algorithm commonly used in deep learning. It is an extension of the stochastic gradient descent (SGD) algorithm and is designed to update the weights of a neural network during training.

The name “Adam” is derived from “adaptive moment estimation,” highlighting its ability to adaptively adjust the learning rate for each network weight individually. Unlike SGD, which maintains a single learning rate throughout training, Adam optimizer dynamically computes individual learning rates based on the past gradients and their second moments.

The creators of Adam optimizer incorporated the beneficial features of other optimization algorithms such as AdaGrad and RMSProp. Similar to RMSProp, Adam optimizer considers the second moment of the gradients, but unlike RMSProp, it calculates the uncentered variance of the gradients (without subtracting the mean).

By incorporating both the first moment (mean) and second moment (uncentered variance) of the gradients, Adam optimizer achieves an adaptive learning rate that can efficiently navigate the optimization landscape during training. This adaptivity helps in faster convergence and improved performance of the neural network.

In summary, Adam optimizer is an optimization algorithm that extends SGD by dynamically adjusting learning rates based on individual weights. It combines the features of AdaGrad and RMSProp to provide efficient and adaptive updates to the network weights during deep learning training.