Exploring Optimizers in Machine Learning

A guide to the widely used optimizer functions and a breakdown of their benefits and limitations

Nikita Sharma Follow
Sep 29, 2020 · 8 min read

Photo by Myles Tan on Unsplash

In this post we're going to embark on a journey to explore and dive deep into the world of optimizers for machine learning models. We will also try and understand the foundational mathematics behind these functions and discuss their use cases, merits, and demerits.

So, what are we waiting for? Let's get started!

What is an Optimizer?

Don't we all love when neural networks work their magic and provide us with tremendous accuracy? Let's get to the core of this magic trick by understanding how our networks find the most optimal parameters for our model.

We know that loss functions are used to understand how good/bad our model performs on the data provided to it. Loss functions are essentially the summation of the difference between the predicted and calculated values for given training samples. For training a neural network to minimize its losses so as to perform better, we need to tweak the weights and parameters associated with the model and the loss function. This is where optimizers play a crucial role.

Optimizers associate loss function and model parameters together by updating the model, i.e. the weights and biases of each node based on the output of the loss function.

An example:

Let us imagine a person hiking down the hill with no sense of direction. He doesn't know the right way to reach the valley, but, he can understand whether he is moving closer (going downhill) or further away (uphill) from his destination. If he keeps taking steps in the correct direction, he will reach the valley.

This is the intuition behind optimizers — to reach a global minima with respect to loss function.

Types of Optimizers

Let's discuss the most frequently used and appreciated optimizers in machine learning:

Gradient Descent

Well, of course we need to start off with the biggest star of our post — gradient descent. Gradient descent is an iterative optimization algorithm. It is dependent on the derivatives of the loss function for finding minima. Running the algorithm for numerous iterations and epochs helps to reach the global minima (or closest to it).

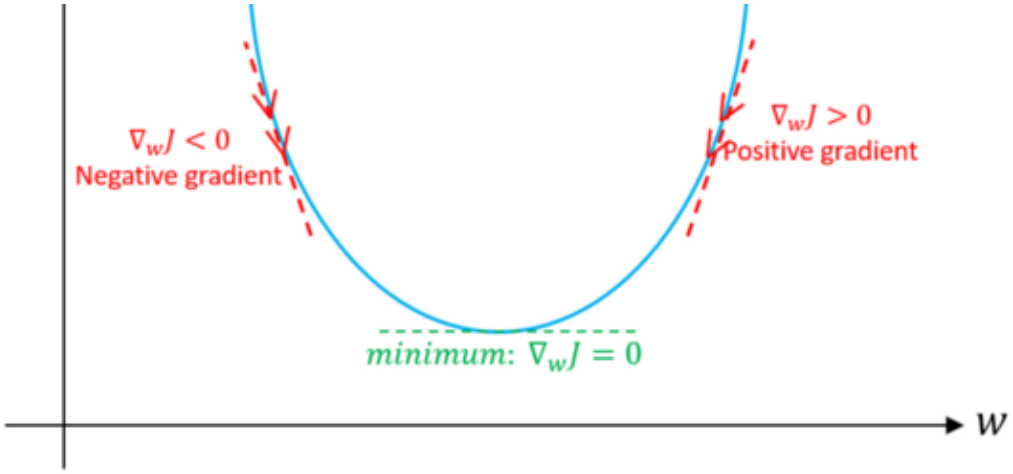


Image Source: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Role of Gradient:

Gradient refers to the slope of the equation in general. Gradients are partial derivatives and can be considered as the small change reflected in the loss function with respect to the small change in weights or parameters of the function. Now, this slight change can tell us what to do next to reduce the output of loss function — reduce this weight by 0.02 or increase this parameter by 0.005.

Learning Rate:

Learning rate is the size of the steps our algorithm takes to reach the global minima. Taking very large steps may skip the global minima, and the model will never reach the optimal value for loss function. On the other hand, taking very small steps will take forever to converge. Thus, size of the step is also dependent on the gradient value.

Gradient Descent

$$\Theta_j = \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta_0, \Theta_1)$$

Learning Rate

In the above formula, α is the learning rate, J is the cost function, and Θ is the parameter to be updated. As you can see, partial derivative of J with respect to Θj gives us the gradient. Note that, as we get closer to the global minima, the slope/gradient of the curve becomes less and less steep, which gives us a smaller value of derivative, which in turn reduces the step size automatically.

Instances of Gradient Descent

- **Vanilla/Batch Gradient Descent-** gradient calculation with all training samples (slow and heavy computation)
- **Stochastic Gradient Descent-** gradient calculation with each training sample (unstable and may shoot even after getting global minima, but is memory efficient)
- **Mini Batch Gradient Descent-** dividing the training data into mini batches (32 , 64… usually in the powers of 2) and gradient calculation for the same (fast and memory efficient with lesser variance)

Advantages of Gradient Descent

It is quite fast and is easy and intuitive to understand and compute.

Disadvantages of Gradient Descent

It may get stuck at a local minima and running gradient descent on a complete dataset takes too long if the dataset is very large (always use batches).

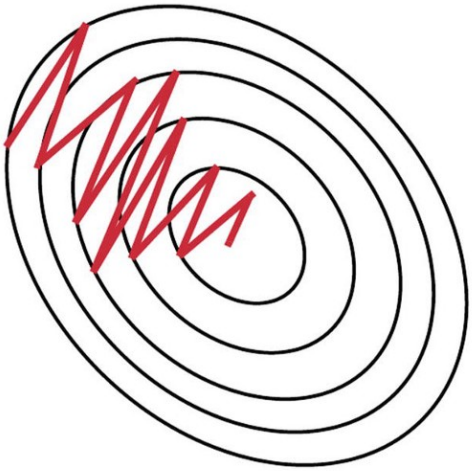
. . .

. . .

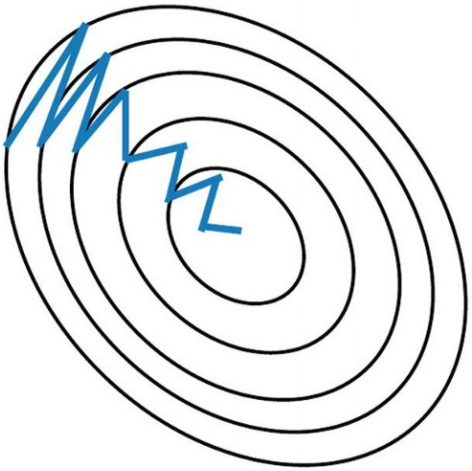
Momentum Based Optimizers

These optimizers help models converge by focusing on movement in the correct direction (downhill) and reduces the variance in every other insignificant direction. Thus, adding another parameter 'Y' to our optimization algorithm to move steeply in the relevant direction helps us move faster towards convergence. The momentum

parameter ‘Y’ typically has a value around 0.9. But, at the same time, one needs to hit and try multiple values and choose manually for the most optimal results.



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

Nesterov Accelerated Gradient (NAG)

If the value of the momentum is too high, our model can even cross the minima and again start going up. Thus, just like a ball, if it has too much energy, it would not stop on the flat surface after rolling down, but will continue to move up.

But, if our algorithm knows when to slow down, it will reach the minima. Thus, in NAG algorithms, the gradient for time step t also takes into account the estimated gradient for time step t+1, which gives a better idea whether to slow down or not. It works better than the conventional momentum algorithm.

$$\theta = \theta - v_t$$
$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta - \gamma v_{t-1})$$
$$\theta - \gamma v_{t-1} \text{ is the gradient of looked ahead}$$

In the above equations, (θ -ϒvt-1) is the next approximated values of the parameters and ∇J(θ -ϒvt-1) is the gradient with respect to the future values.

Advantages of NAG

It has an understanding of motion or future gradients. Thus, it can decrease momentum when the gradient value is lesser or the slope is low and can increase momentum when the slope is steep.

Disadvantages of NAG

NAG is not adaptive with respect to the parameter importance. Thus, all parameters are updated in a similar manner.

AdaGrad — Adaptive Gradient Algorithm

How cool would it be if our algorithm adapts different learning rates for different parameters? This is the intuition behind AdaGrad — some weights will have different learning rates based on how frequently or by how much their values are updated so as to perform optimally. Larger learning rate updates are made for less frequent parameters and smaller ones for the frequent parameters. The learning rate however, always decreases.

$$g_0 = 0$$
$$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

Here, as you can see, AdaGrad uses the sum of the square of past gradients to calculate the learning rate for each parameter (∇L(Θ) is the gradient or partial derivative of cost function with respect to Θ).

Advantages of AdaGrad

Works great for datasets which have missing samples or are sparse in nature.

Disadvantages of AdaGrad

The learning might be very slow, since, according to the formula above, division by bigger numbers (sum of past gradients become bigger and bigger with time) means that the learning rate is decreasing over time — therefore the pace of learning would also decrease.

RMS-Prop — Root Mean Square Propagation

RMS-Prop is almost similar to AdaGrad, with just a minute difference: It uses exponentially decaying average instead of the sum of gradients. Thus, RMS-Prop basically combines momentum with AdaGrad. Also, instead of using all the gradients for momentum, it takes into account only the most recent gradients for momentum calculation. This gives the model an understanding of whether it should increase or decrease its learning rate, given the current scenario.

$$g_0 = 0\,, \alpha \simeq 0.9$$

$$g_{t+1} \leftarrow \alpha \cdot g_t + (1-\alpha)\nabla_\theta \mathcal{L}(\theta)^2$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

Advantages of RMS-Prop

AdaGrad decreases the learning rate with each time step, but RMS-Prop can adapt to an increase or a decrease in the learning rate with each epoch.

Disadvantages of RMS-Prop

The learning can be pretty slow, same reason as in AdaGrad.

Adam — Adaptive Moment Estimation

Adam optimizer also uses adaptive learning rate technique to calculate current gradient from the first and second moments of the past gradients. Adam optimizer can be viewed as a combination of momentum and RMS-prop and is the most widely used optimizer for a wide range of problems.

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{v}_t &= \max(\hat{v}_{t-1}, v_t) \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} m_t
\end{aligned}
$$

Rather than influencing the learning rate with respect to first moments as in RMSProp, Adam also uses the average of the second moments of the gradients. Also, Adam calculates an exponential moving average of the gradient and the squared gradient. Thus, Adam can be considered as a combination of AdaGrad and RMS-Prop.

In the above formula, α is the learning rate, β1(usually ~0.9) is the exponential decay rate with respect to the first moments, β2 is the exponential decay rate with respect to the second moments (usually ~0.999). ∈ is just a small value to avoid division by zero.

Advantages of Adam

Adam optimizer is well suited for large datasets and is computationally efficient.

Disadvantages of Adam

There are few disadvantages as the Adam optimizer tends to converge faster, but other algorithms like the Stochastic gradient descent focus on the datapoints and generalize in a better manner. Thus, the performance depends on the type of data being provided and the speed/generalization trade-off.

Race to the Global Minima

Here is an animation of how different optimizers help the model reach the global minima for a particular data space:

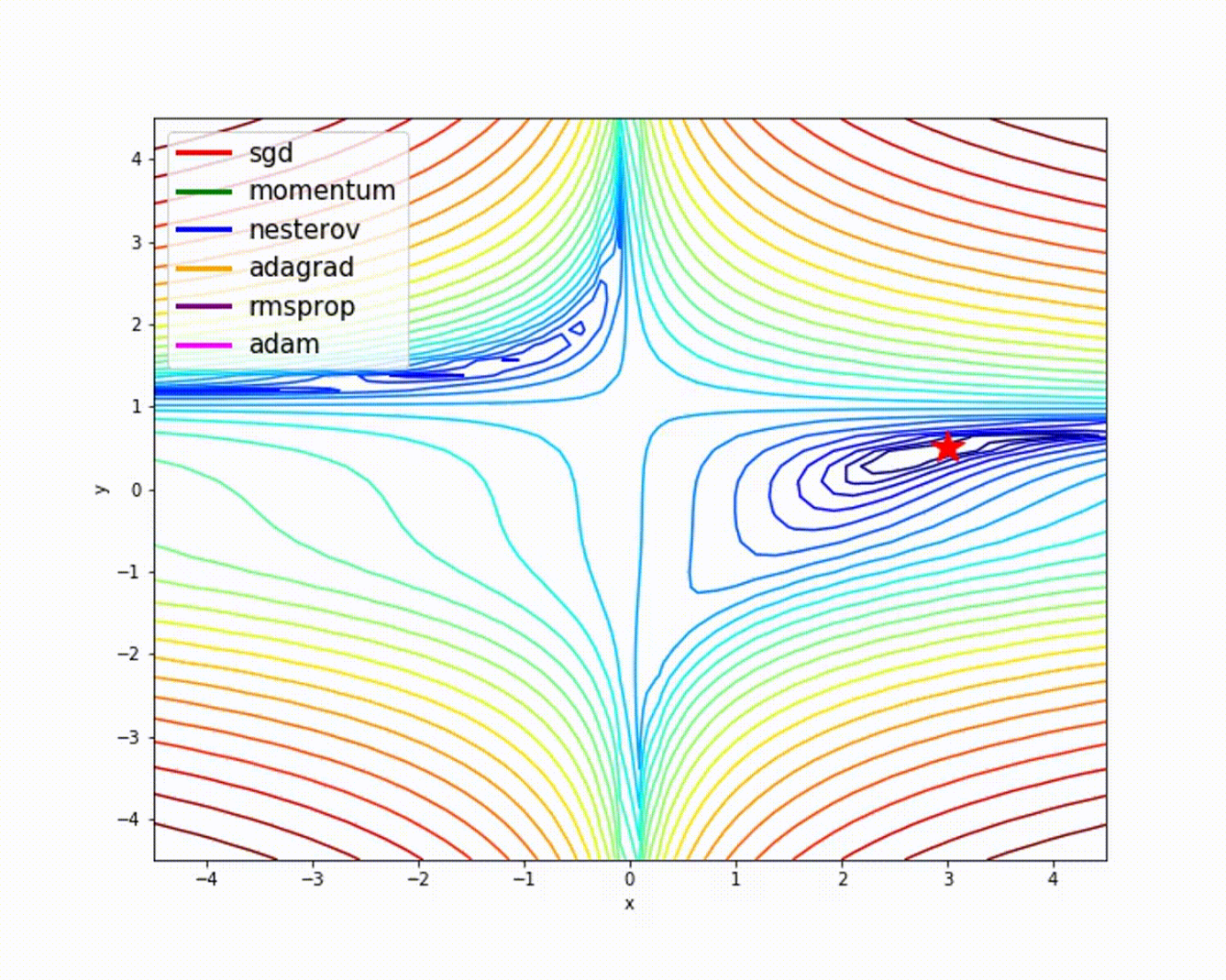


Image Source:https://github.com/ilguyi/optimizers.numpy

Conclusion

In this post we discussed about various optimizers like gradient descent and its variations, Nesterov accelerated gradient, AdaGrad, RMS-Prop, and Adam along with their primary use cases. These are the most widely-used optimizer functions and are essential for developing efficient neural networks. Choosing an optimizer is influential to building efficient and fast neural networks.

I hope this article has helped you learn and understand more about these fundamental ML concepts. All feedback is welcome. Please help me improve!

Until next time!

• • •

Editor's Note: ***Heartbeat*** *is a contributor-driven online publication and community dedicated to exploring the emerging intersection of mobile app development and machine learning. We're committed to supporting and inspiring developers and engineers from all walks of life.*

Editorially independent, Heartbeat is sponsored and published by ***Fritz AI****, the machine learning platform that helps developers teach devices to see, hear, sense, and think. We pay our contributors, and we don't sell ads.*

If you'd like to contribute, head on over to our ***call for contributors****. You can also sign up to receive our weekly newsletters (****Deep Learning Weekly*** *and the* ***Fritz AI Newsletter****), join us on* ***Slack****, and follow Fritz AI on* ***Twitter*** *for all the latest in mobile machine learning.*

Machine Learning | Optimization | Gradient Descent | Heartbeat | Data Science