Overview of different Optimizers for neural networks

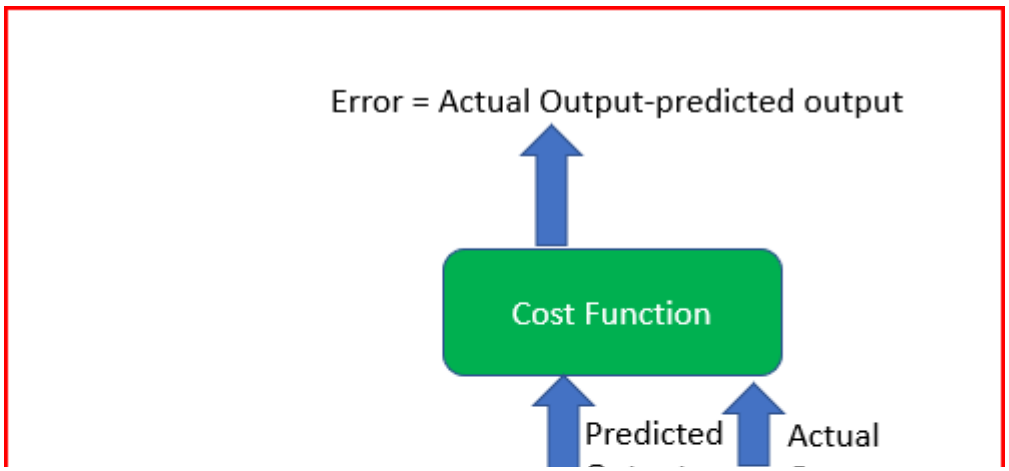
Renu Khandelwal Follow
Feb 3, 2019 · 6 min read

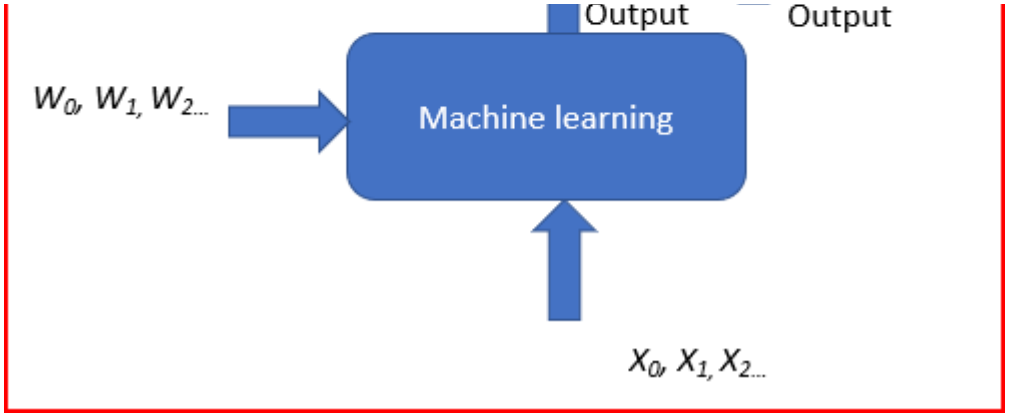
In this post, we will start to understand the objective of Machine Learning algorithms. How Gradient Descent helps achieve the goal of machine learning. Understand the role of optimizers in Neural networks. Explore different optimizers like Momentum, Nesterov, Adagrad, Adadelta, RMSProp, Adam and Nadam.

The objective of Machine Learning algorithm

The goal of machine learning and deep learning is to **reduce the difference between the predicted output and the actual output**. This is also called as a **Cost function(C) or Loss function**. Cost functions are convex functions.

As our goal is to minimize the cost function by finding the optimized value for weights. We also need to ensure that the algorithm generalizes well. This will help make a better prediction for the data that was not seen before





Gradient based learning

To achieve this we run multiple iterations with different weights. This helps to find the minimum cost. This is Gradient descent.

Gradient Descent

Gradient descent is an iterative machine learning optimization algorithm to reduce the cost function. This will help models to make accurate predictions.

Gradient indicates the direction of increase. As we want to find the minimum point in the valley we need to go in the opposite direction of the gradient. **We update parameters in the negative gradient direction to minimize the loss**.

$$\theta = \theta - \eta \nabla J(\theta; x, y)$$

θ is the weight parameter, η is the learning rate and ∇J(θ;x,y) is the gradient of weight parameter θ

Types of Gradient Descent

Different types of Gradient descents are

- **Batch Gradient Descent or Vanilla Gradient Descent**
- **Stochastic Gradient Descent**
- **Mini batch Gradient Descent**

Read details of the different types of Gradient Descent here

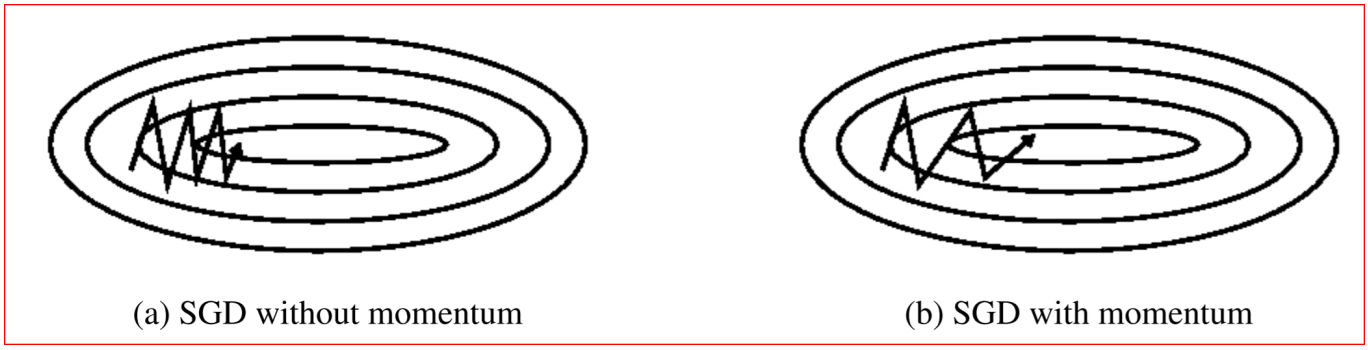
Role of an optimizer

Optimizers update the weight parameters to minimize the loss function. Loss function acts as guides to the terrain telling optimizer if it is moving in the right direction to reach the bottom of the valley, the global minimum.

Types of Optimizers

Momentum

Momentum is like a ball rolling downhill. The ball will gain momentum as it rolls down the hill.



(a) SGD without momentum (b) SGD with momentum

Source: Genevieve B. Orr

Momentum helps accelerate Gradient Descent(GD) when we have surfaces that curve more steeply in one direction than in another direction. It also dampens the oscillation as shown above

For updating the weights it takes the gradient of the current step as well as the gradient of the previous time steps. This helps us move faster towards convergence.

Convergence happens faster when we apply momentum optimizer to surfaces with curves.

$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta; x, y)$$

$$\theta = \theta - v_t$$

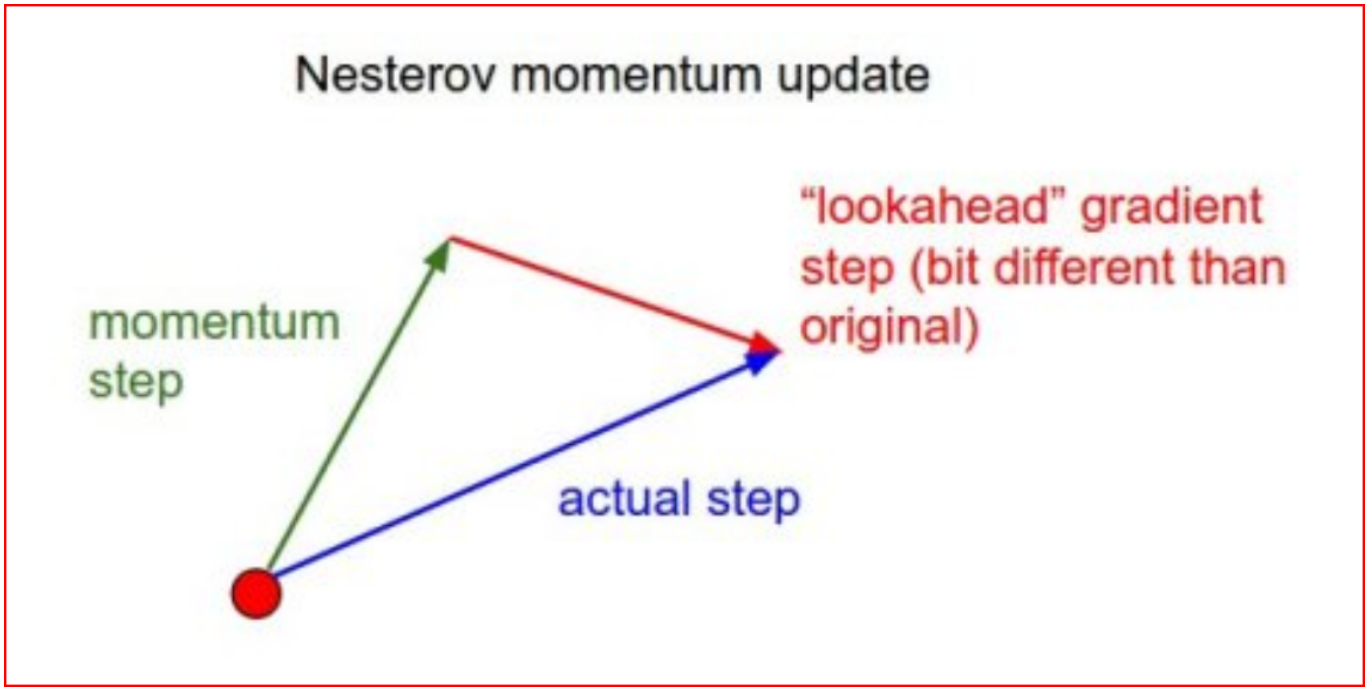
$$\theta = \theta - v_t$$

Momentum Gradient descent takes gradient of previous time steps into consideration

Nesterov accelerated gradient(NAG)

Nesterov acceleration optimization is like a ball rolling down the hill but knows exactly when to slow down before the gradient of the hill increases again.

We calculate the gradient not with respect to the current step but with respect to the future step. We evaluate the gradient of the looked ahead and based on the importance then update the weights.



source:http://cs231n.github.io/neural-networks-3/

NAG is like you are going down the hill where we can look ahead in the future. This way we can optimize our descent faster. Works slightly better than standard Momentum.

$$\theta = \theta - v_t$$

$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta - \gamma v_{t-1})$$

$\theta - \gamma v_{t-1}$ is the gradient of looked ahead

Nesterov Accelerated Gradient

Adagrad — Adaptive Gradient Algorithm

We need to tune the learning rate in Momentum and NAG which is an expensive process.

Adagrad is an adaptive learning rate method. In Adagrad we adopt the learning rate to the parameters. **We perform larger updates for infrequent parameters and smaller updates for frequent parameters.**

It is **well suited when we have sparse data as in large scale neural networks. GloVe word embedding uses adagrad where infrequent words required a greater update and frequent words require smaller updates**.

For SGD, Momentum, and NAG we update for all parameters θ at once. We also use the same learning rate η. In Adagrad we use different learning rate for every parameter θ for every time step t

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot g_t$$

G_t is sum of the squares of the past gradients w.r.t. to all parameters θ

Adagrad

Adagrad eliminates the need to manually tune the learning rate.

In the denominator, we accumulate the sum of the square of the past gradients. Each term is a positive term so it keeps on growing to make the learning rate η infinitesimally small to the point that algorithm is no longer able learning. **Adadelta, RMSProp, and adam tries to resolve Adagrad's radically diminishing learning rates**.

Adadelta

- Adadelta is an extension of Adagrad and it also tries to reduce Adagrad's aggressive, monotonically reducing the learning rate
- It does this by **restricting the window of the past accumulated gradient to some fixed size of w. Running average at time *t* then depends on the previous average and the current gradient**
- In Adadelta we do not need to set the default learning rate as we take the ratio of the running average of the previous time steps to the current gradient

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

$$\Delta\theta = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g_t]} \cdot g_t$$

RMSProp

- RMSProp is Root Mean Square Propagation. It was devised by Geoffrey Hinton.
- RMSProp tries to resolve Adagrad's radically diminishing learning rates by **using a moving average of the squared gradient**. It utilizes the magnitude of the recent gradient descents to normalize the gradient.
- In RMSProp learning rate gets adjusted automatically and it chooses a different learning rate for each parameter.
- RMSProp divides the learning rate by the average of the exponential decay of squared gradients

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(1-\gamma){g^2}_{t-1} + \gamma g_t + \varepsilon}} \cdot g_t$$

γ is the decay term that takes value from 0 to 1. gt is moving average of squared gradients

Adam — Adaptive Moment Estimation

- Another method that **calculates the individual adaptive learning rate for each parameter from estimates of first and second moments of the gradients.**
- It also reduces the radically diminishing learning rates of Adagrad
- Adam can be viewed as a **combination of Adagrad, which works well on sparse gradients and RMSprop which works well in online and nonstationary settings**.
- **Adam implements the exponential moving average of the gradients to scale the learning rate instead of a simple average as in Adagrad. It keeps an exponentially decaying average of past gradients**
- Adam is computationally efficient and has very little memory requirement
- **Adam optimizer is one of the most popular gradient descent optimization algorithms**

Adam algorithm first updates the exponential moving averages of the gradient(mt) and the squared gradient(vt) which is the estimates of the first and second moment.

Hyper-parameters β1, β2 ∈ [0, 1) control the exponential decay rates of these moving averages as shown below

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

m_t and v_t are estimates of first and second moment respectively

Moving averages are initialized as 0 leading to moment estimates that are biased around 0 especially during the initial timesteps. This initialization bias can be easily counteracted resulting in bias-corrected estimates

$$\hat{m}_t = \frac{m_t}{}$$

$$\hat{m}_t = \frac{M_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{V_t}{1-\beta_2^t}$$

$\hat{m}_t$ and $\hat{v}_t$ are bias corrected estimates of first and second moment respectively

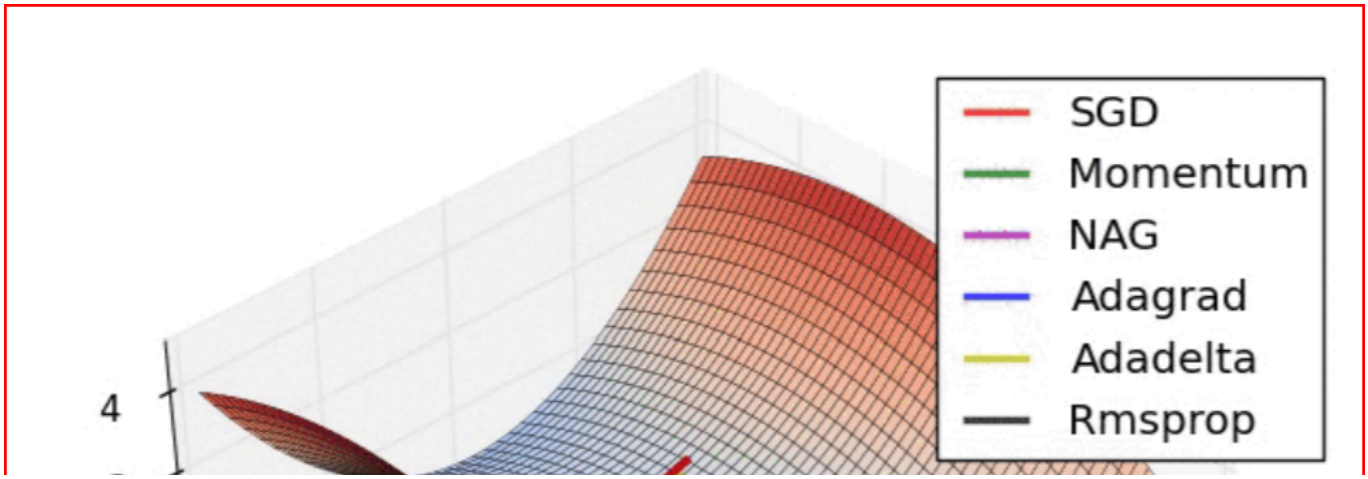
Finally, we update the parameter as shown below

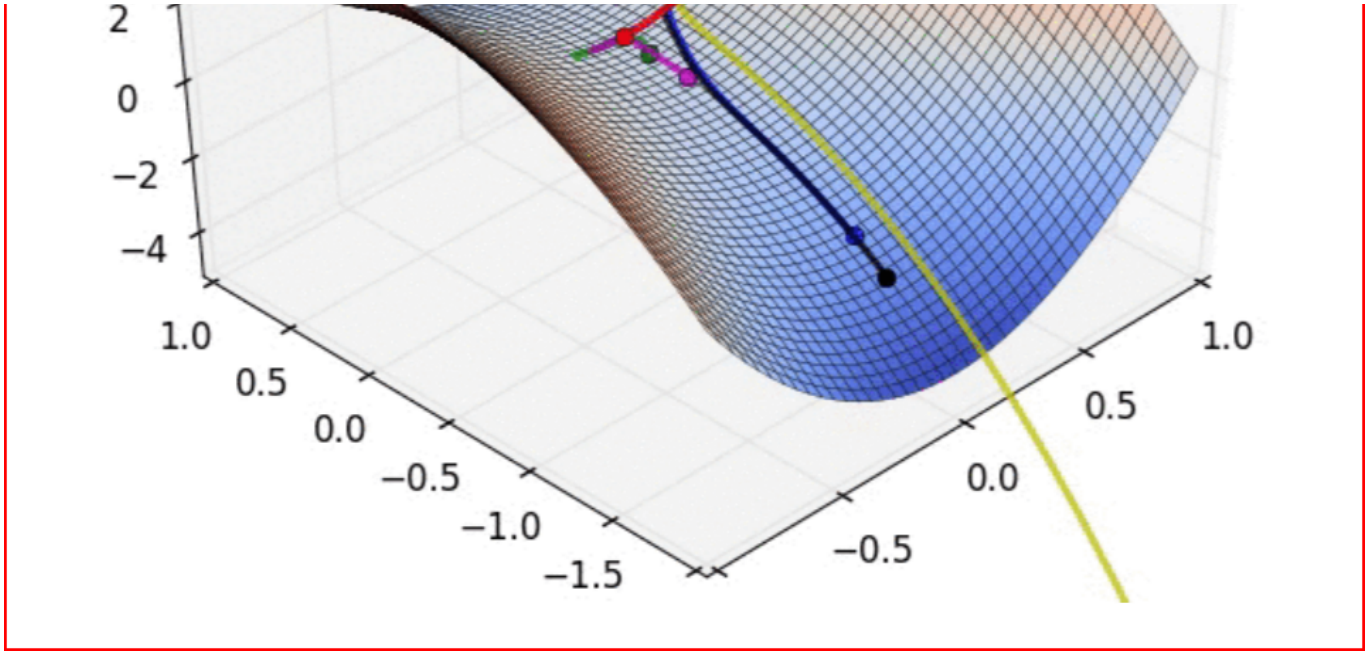
$$\theta_{t+1} = \theta_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

Nadam- Nesterov-accelerated Adaptive Moment Estimation

- Nadam combines NAG and Adam
- Nadam is employed for noisy gradients or for gradients with high curvatures
- The learning process is accelerated by summing up the exponential decay of the moving averages for the previous and current gradient

In the diagram below we see can see how different optimizer will converge to the minimum. Adagrad, Adadelta, and RMSprop headed off immediately in the right direction and converge. Momentum and NAG were led off-track, evoking the image of a ball rolling down the hill. NAG corrected itself quickly





Source Source and full animations: Alec Radford

Clap if you liked the article

References:

Adam: A Method for Stochastic Optimization by Diederik P. Kingma, Jimmy Ba

http://cs231n.github.io/neural-networks-3/

https://arxiv.org/pdf/1609.04747.pdf

http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

. . .

Related Stories from DDI:

Deep Learning Explained in 7 Steps

with cats

medium.com

Machine Learning for Stock Market Investing

When a friend of yours uploads your new beach-body photo on Facebook and the platform suggests to tag your face, it is...

medium.com

Deep Learning Optimizer Adam Momentum Neural Networks