Overview of various Optimizers in Neural Networks

Understand the relation between various optimizers along with their advantages and disadvantages.

Satyam Kumar Jun 8 · 8 min read ★

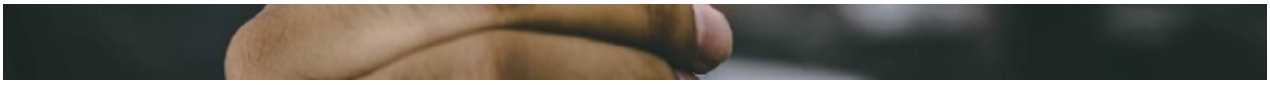


Photo by Hitesh Choudhary on Unsplash

Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function.

> ***How do Optimizers work?***

How you should change your weights or learning rates of your neural network to reduce the losses is defined by the optimizers you use. Optimization algorithms are responsible for reducing the losses and to provide the most accurate results possible.

The weight is initialized using some initialization strategies and is updated with each epoch according to the update equation.

$$W_{new} = W_{old} - lr * (\nabla_W L)_{W_{old}}$$

The above equation is the update equation using which weights are updated to reach the most accurate result. The best result can be achieved using some optimization strategies or algorithms called optimizers.

Various optimizers are researched within the last few couples of years each having its advantages and disadvantages. Read the

entire article to understand the working, advantages, and disadvantages of the algorithms.

Gradient Descent (GD):

Gradient descent is the most basic and first-order optimization algorithm which is dependent on the first-order derivative of a loss function. It calculates which way the weights should be altered so that the function can reach a minimum. Through backpropagation, the loss is transferred from one layer to another and the model's parameters also known as weights are modified depending on the losses so that the loss can be minimized.

$$W_{new} = W_{old} - lr * (\nabla_W L)_{W_{old}}$$

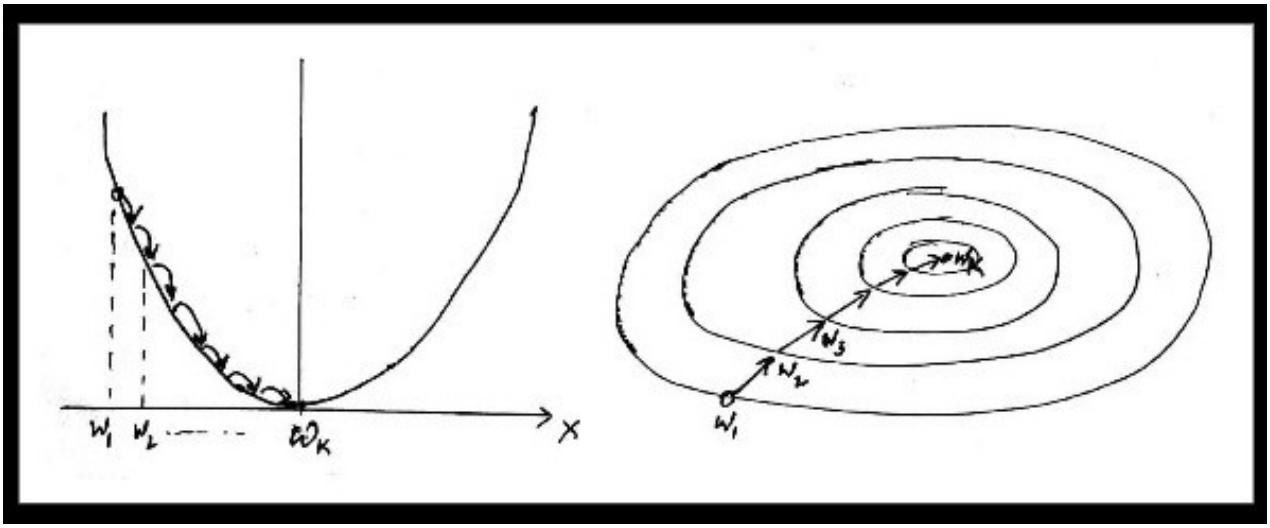


Image 1

From the image above (Image 1) it is observed that weights are updated to converge to the minima.

Advantages:

1. Very simple to implement.

Disadvantages:

1. This algorithm takes an entire dataset of n-points at a time to compute the derivative to update the weights which require a lot of memory.
2. Minima is reached after a long time or is never reached.
3. This algorithm can be stuck at local minima or saddle point:

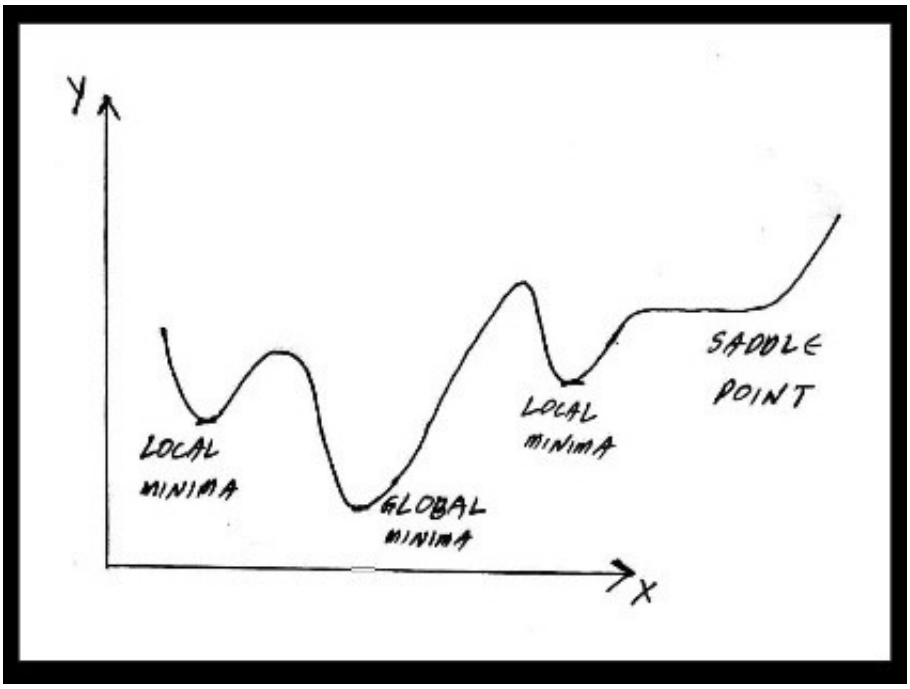


Image 2

In the image above (Image 2), gradient descent may get stuck at local minima or saddle point and con never converge to minima. To find the best solution the algorithm must reach global minima.

> ***Stochastic Gradient Descent (SGD):***

SGD algorithm is an extension of the GD algorithm and it overcomes some of the disadvantages of the GD algorithm. GD algorithm has a disadvantage that it requires a lot of memory to load the entire dataset of n-points at a time to computer derivative. In the case of the SGD algorithm derivative is computed taking one point at a time.

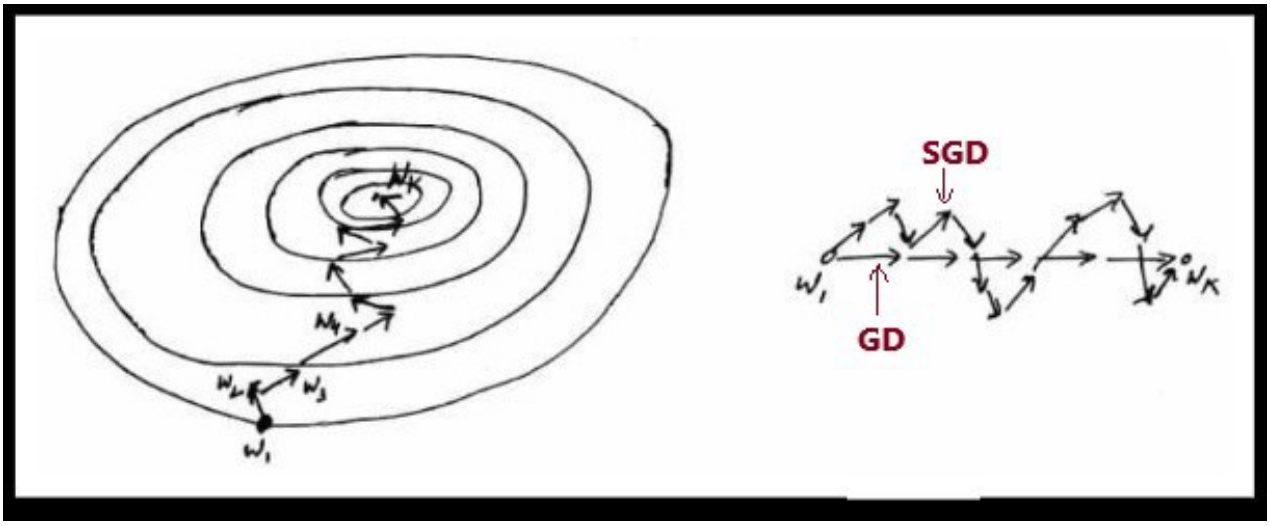


Image 3

From the above diagram (Image 3), it is observed that the updates take more number of iterations compared to gradient descent to reach minima. In the right part of Image 3, the GD algorithm takes fewer steps to reach minima but the SGD algorithm is noisier and takes more iterations.

Advantage:

1. Memory requirement is less compared to the GD algorithm as derivative is computed taking only 1 point at once.

Disadvantages:

1. The time required to complete 1 epoch is large compared to the GD algorithm.
2. Takes a long time to converge.
3. May stuck at local minima.

Mini Batch Stochastic Gradient Descent (MB-SGD):

MB-SGD algorithm is an extension of the SGD algorithm and it overcomes the problem of large time complexity in the case of the SGD algorithm. MB-SGD algorithm takes a batch of points or subset of points from the dataset to compute derivate.

It is observed that the derivate of the loss function for MB-SGD is almost the same as a derivate of the loss function for GD after some number of iterations. But the number of iterations to achieve minima is large for MB-SGD compared to GD and the cost of computation is also large.

The update of weight is dependent on the derivate of loss for a batch of points. The updates in the case of MB-SGD are much noisy because the derivative is not always towards minima.

Advantages:

1. Less time complexity to converge compared to standard SGD algorithm.

Disadvantages:

1. The update of MB-SGD is much noisy compared to the update of the GD algorithm.
2. Take a longer time to converge than the GD algorithm.
3. May get stuck at local minima.

SGD with momentum:

A major disadvantage of the MB-SGD algorithm is that updates of weight are very noisy. SGD with momentum overcomes this disadvantage by denoising the gradients. Updates of weight are dependent on noisy derivative and if we somehow denoise the derivatives then converging time will decrease.

The idea is to denoise derivative using exponential weighting

average that is to give more weightage to recent updates compared to the previous update.

SGD update equation:

$$W_{new} = W_{old} - lr * (\nabla_W L)_{W_{old}}$$

$$V_{new} = \alpha * V_{old} + lr * (\nabla_W L)_{W_{old}}$$

$$W_{new} = W_{old} - (\alpha * V_{old} + lr * (\nabla_W L)_{W_{old}})$$

$$0 < \alpha < 1$$

Momentum at time 't' is computed using all previous updates giving more weightage to recent updates compared to the previous update. This lead to speed up the convergence.

How momentum speeds up the convergence?

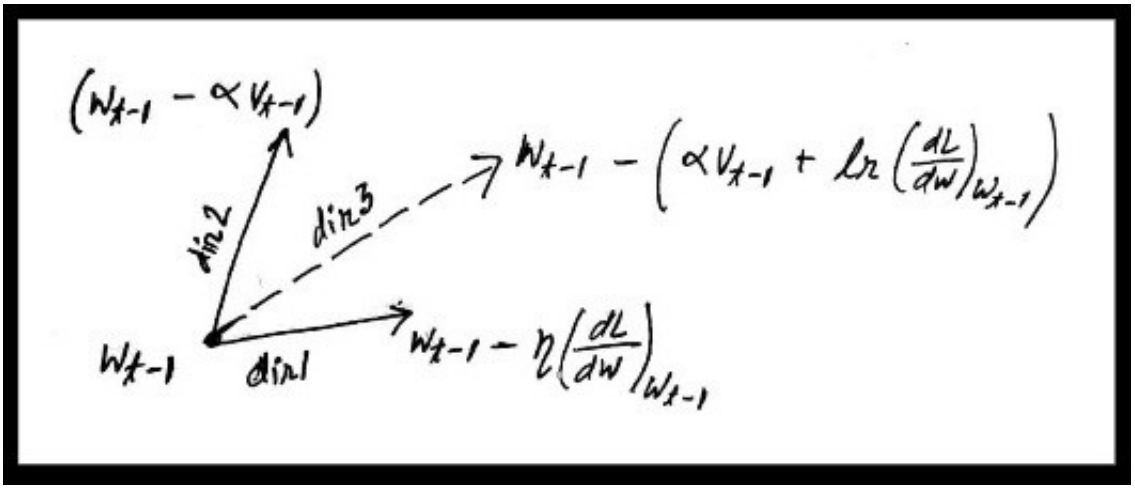


Image 4

When the new weight is calculated by subtracting only the gradient term with a previous weight the update moves in direction 1 and if new weight is calculated by subtracting momentum term with a previous weight then update moves in direction 2 (Image 5).

What if we combine both the equation and calculate the new weight by subtracting previous weight with the sum of momentum and gradient term then the update will move towards direction 3 which results in denoising the update.

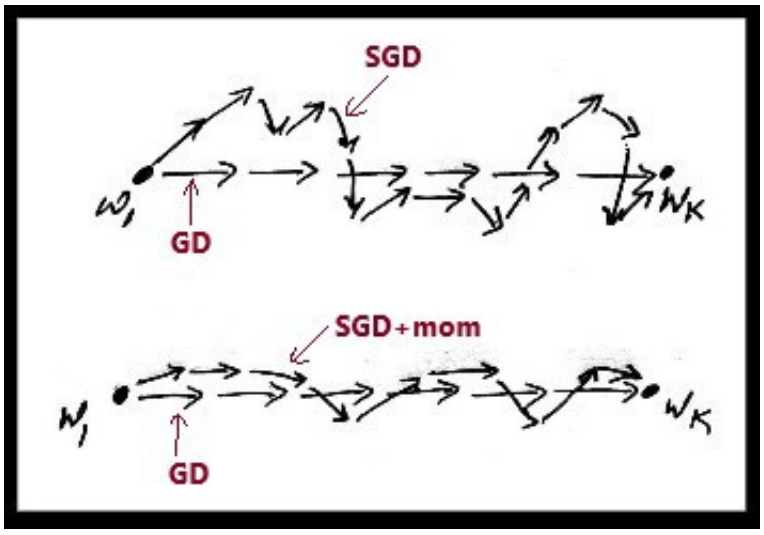


Image 5

The diagram above (Image 5) concludes SGD+momentum denoises the gradients and converges faster as compared to SGD.

Advantages:

1. Has all advantages of the SGD algorithm.
2. Converges faster than the GD algorithm.

Disadvantages:

1. We need to compute one more variable for each update.

> ***Nesterov Accelerated Gradient (NAG):***

The idea of the NAG algorithm is very similar to SGD with momentum with a slight variant. In the case of SGD with momentum algorithm, the momentum and gradient are computed on previous updated weight.

$$W_{dash} = W_{t-1} - \alpha * V_{t-1}$$

$$W_t = W_{dash} - (lr * \nabla_{W_{dash}} L)$$

$$W_t = W_{t-1} - ((momentum) + (lr * derivative))$$

$$W_t = W_{t-1} - ((\alpha * V_{t-1}) + (lr * \nabla_{W_{dash}} L))$$

According to the NAG algorithm firstly compute momentum V_t

for point W_(t-1) and move in that direction to reach W_dash, then compute the gradient at new updated weight W_dash and again move towards the gradient (Image 6). The net movement results in the direction towards the minima.

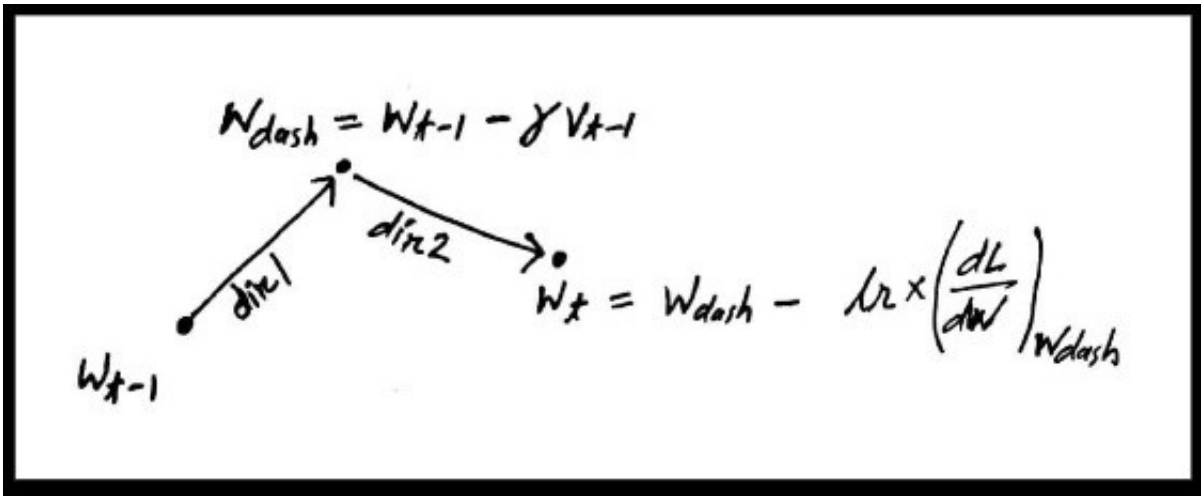


Image 6

Both NAG and SGD with momentum algorithms work equally well and share the same advantages and disadvantages.

Adaptive Gradient (AdaGrad):

For all the previously discussed algorithms the learning rate remains constant. So the key idea of AdaGrad is to have an adaptive learning rate for each of the weights. The learning rate for weight will be decreasing with the number of iteration.

$$W_t = W_{t-1} - (lr_t * (\nabla_{W_{t-1}} L))$$

$$g_t = \nabla_{W_{t-1}} L$$

$$W_t = W_{t-1} - (lr_t * g_t)$$

$$lr_t = \frac{lr}{\sqrt{\alpha_{t-1} + \varepsilon}}$$

$$\alpha_{t-1} = \sum_{i=1}^{t-1} g_i^2$$

So with the increase in the number of the iteration (t) learning rate alpha increasing, which results in an adaptively decrease in the learning rate.

Advantage:

1. No need to update the learning rate manually as it changes adaptively with iterations.

Disadvantage:

1. As the number of iteration becomes very large learning rate decreases to a very small number which leads to slow convergence.

AdaDelta:

Problem with previous algorithm AdaGrad was learning rate becomes very small with a large number of iterations which lead to slow convergence. To avoid this, the AdaDelta algorithm has an idea to take an exponentially decaying average.

$$W_t = W_{t-1} - (lr_t * g_t)$$

$$lr_t = \frac{lr}{\sqrt{eda_{t-1} + \varepsilon}}$$

$$eda_{t-1} = (\gamma eda_{t-2}) + ((1 - \gamma) * g_{t-1}^2)$$

Adam:

In the case of the AdaDelta algorithm, we were storing exponential decaying averages of the square of gradients to modify the learning rate. For Adam optimizer, the idea is to store both 1st order of moment (g_t) and 2nd order moment of the gradient (square of g_t).

EDA for 1st order moment:

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

EDA for 2nd order moment:

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

Bias Correction:

$$m_t^* = \frac{m_t}{1 - \beta_1^t}$$

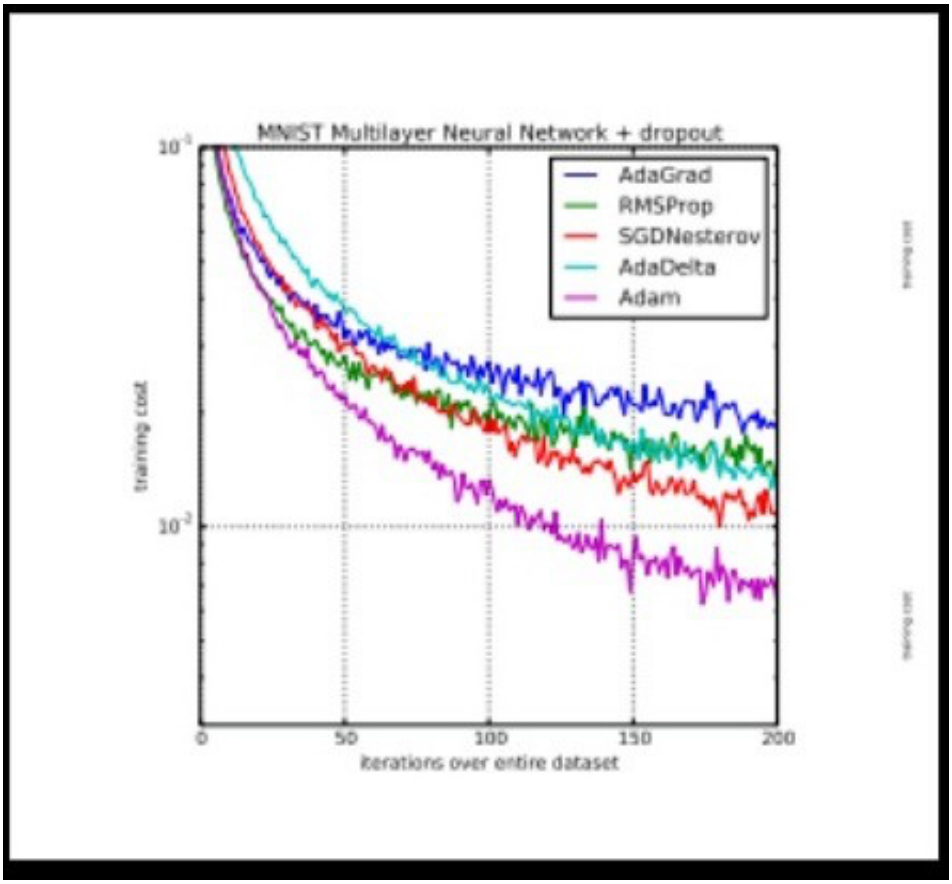
$$v_t^* = \frac{v_t}{1 - \beta_2^t}$$

Update Function:

$$W_t = W_{t-1} - \alpha * \frac{m_t^*}{\sqrt{v_t^* + \varepsilon}}$$

$$0 < \beta_1, \beta_2, \alpha < 1$$

Conclusion and Comparison of Optimizers:

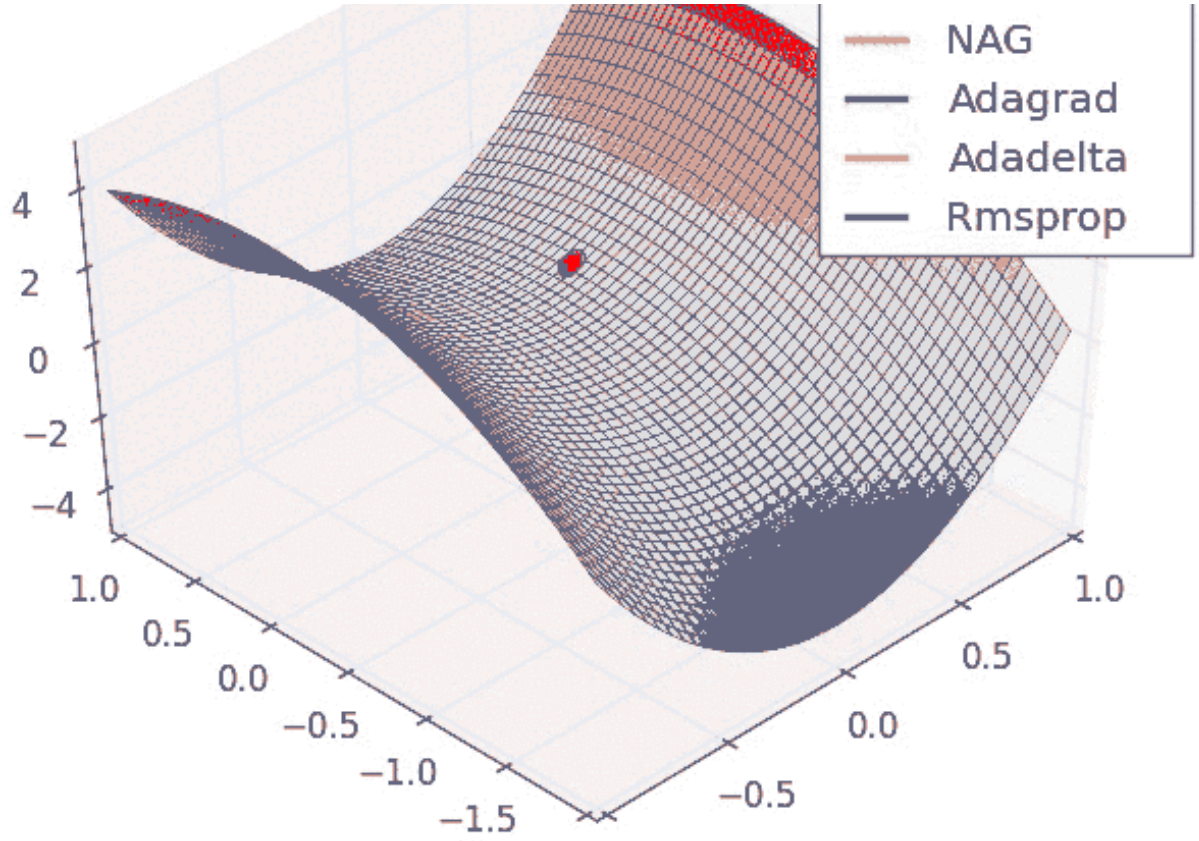


Source: https://arxiv.org/pdf/1412.6980.pdf

The image above is a plot for the number of iteration vs training loss of algorithms discussed in this article.

For a given number of iteration (suppose 100), it is observed from the plot that adam converges the fastest amongst all other algorithms.

Which algorithm to choose when?



Source: https://ruder.io/content/images/2016/09/saddle_point_evaluation_optimizers.gif

- In the animation above it is observed that the SGD algorithm (red) is stuck at a saddle point. So SGD algorithm can only be used for shallow networks.
- All the other algorithms except SGD finally converges one after the other, AdaDelta being the fastest followed by momentum algorithms.
- AdaGrad and AdaDelta algorithm can be used for sparse data.
- Momentum and NAG work well for most cases but is slower.
- Animation for Adam is not available but from the plot above it is observed that it is the fastest algorithm to converge to minima.
- Adam is considered the best algorithm amongst all the

algorithms discussed above.

Thank You For Reading!