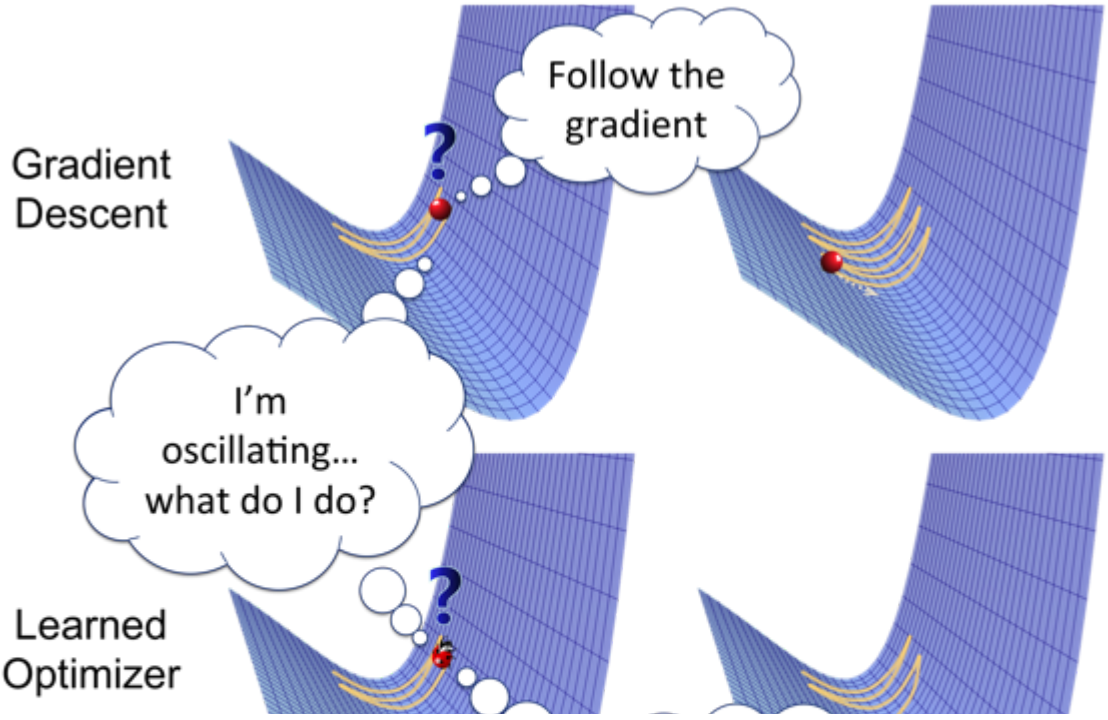
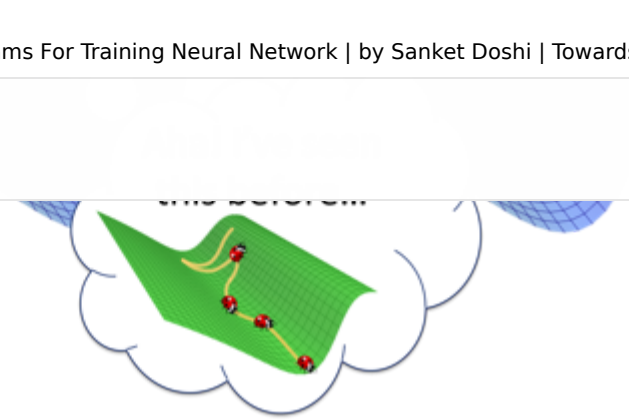
Various Optimization Algorithms For Training Neural Network

The right optimization algorithm can reduce training time exponentially.

Sanket Doshi Jan 13, 2019 · 7 min read

Many people may be using optimizers while training the neural network without knowing that the method is known as optimization. Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.





Optimizers help to get results faster

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 or strategies are responsible for reducing the losses and to provide the most accurate results possible.

We'll learn about different types of optimizers and their advantages:

Gradient Descent

Gradient Descent is the most basic but most used optimization algorithm. It's used heavily in linear regression and classification algorithms. Backpropagation in neural networks also uses a gradient descent algorithm.

Gradient descent is a first-order optimization algorithm which is dependent on the first order derivative of a loss function. It calculates that which way the weights should be altered so that the function can reach a minima. 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.

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

Advantages:

1. Easy computation.
2. Easy to implement.
3. Easy to understand.

Disadvantages:

2. Weights are changed after calculating gradient on the whole dataset. So, if the dataset is too large than this may take years to converge to the minima.
3. Requires large memory to calculate gradient on the whole dataset.

Stochastic Gradient Descent

It's a variant of Gradient Descent. It tries to update the model's parameters more frequently. In this, the model parameters are altered after computation of loss on each training example. So, if the dataset contains 1000 rows SGD will update the model parameters 1000 times in one cycle of dataset instead of one time as in Gradient Descent.

$\theta = \theta - \alpha \cdot \nabla J(\theta; x(i); y(i))$ **, where {x(i) ,y(i)} are the training examples**.

As the model parameters are frequently updated parameters have high variance and fluctuations in loss functions at different intensities.

Advantages:

1. Frequent updates of model parameters hence, converges in less time.
2. Requires less memory as no need to store values of loss functions.
3. May get new minima's.

Disadvantages:

1. High variance in model parameters.
2. May shoot even after achieving global minima.
3. To get the same convergence as gradient descent needs to slowly reduce the value of learning rate.

Mini-Batch Gradient Descent

It's best among all the variations of gradient descent algorithms. It is an improvement on both SGD and standard gradient descent. It updates the model parameters after every

$\theta = \theta - \alpha \cdot \nabla J(\theta; B(i))$, where {B(i)} are the batches of training examples.

Advantages:

1. Frequently updates the model parameters and also has less variance.
2. Requires medium amount of memory.

All types of Gradient Descent have some challenges:

1. Choosing an optimum value of the learning rate. If the learning rate is too small than gradient descent may take ages to converge.
2. Have a constant learning rate for all the parameters. There may be some parameters which we may not want to change at the same rate.
3. May get trapped at local minima.

Momentum

Momentum was invented for reducing high variance in SGD and softens the convergence. It accelerates the convergence towards the relevant direction and reduces the fluctuation to the irrelevant direction. One more hyperparameter is used in this method known as momentum symbolized by 'γ'.

$$V(t) = \gamma V(t-1) + \alpha . \nabla J(\theta)$$

Now, the weights are updated by $\theta = \theta - V(t)$.

The momentum term γ is usually set to 0.9 or a similar value.

Advantages:

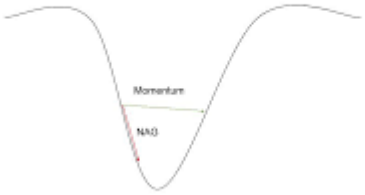
1. Reduces the oscillations and high variance of the parameters.
2. Converges faster than gradient descent.

1. One more hyper-parameter is added which needs to be selected manually and accurately.

Nesterov Accelerated Gradient

Momentum may be a good method but if the momentum is too high the algorithm may miss the local minima and may continue to rise up. So, to resolve this issue the NAG algorithm was developed. It is a look ahead method. We know we'll be using $\gamma V(t-1)$ for modifying the weights so, $\theta - \gamma V(t-1)$ approximately tells us the future location. Now, we'll calculate the cost based on this future parameter rather than the current one.

$V(t) = \gamma V(t-1) + \alpha . \nabla J(\theta - \gamma V(t-1))$ and then update the parameters using $\theta = \theta - V(t)$.



NAG vs momentum at local minima

Advantages:

1. Does not miss the local minima.
2. Slows if minima's are occurring.

Disadvantages:

1. Still, the hyperparameter needs to be selected manually.

Adagrad

One of the disadvantages of all the optimizers explained is that the learning rate is constant for all parameters and for each cycle. This optimizer changes the learning rate.

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}),$$

A derivative of loss function for given parameters at a given time t.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$

Update parameters for given input i and at time/iteration t

η is a learning rate which is modified for given parameter **θ(i)** at a given time based on previous gradients calculated for given parameter **θ(i).**

We store the sum of the squares of the gradients w.r.t. **θ(i)** up to time step **t**, while **ϵ** is a smoothing term that avoids division by zero (usually on the order of 1e−8). Interestingly, without the square root operation, the algorithm performs much worse.

It makes big updates for less frequent parameters and a small step for frequent parameters.

Advantages:

1. Learning rate changes for each training parameter.
2. Don't need to manually tune the learning rate.
3. Able to train on sparse data.

Disadvantages:

1. Computationally expensive as a need to calculate the second order derivative.
2. The learning rate is always decreasing results in slow training.

AdaDelta

window of accumulated past gradients to some fixed size **w**. In this exponentially moving average is used rather than the sum of all the gradients.

$$\mathbf{E[g^2](t) = \gamma . E[g^2](t-1) + (1-\gamma) . g^2(t)}$$

We set **γ** to a similar value as the momentum term, around 0.9.

$$\mathbb{E}[g^2]_t = \gamma \mathbb{E}[g^2]_{t-1} + (1-\gamma) g_t^2,$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_t + \epsilon}} \cdot g_t.$$

Update the parameters

Advantages:

1. Now the learning rate does not decay and the training does not stop.

Disadvantages:

1. Computationally expensive.

Adam

Adam (Adaptive Moment Estimation) works with momentums of first and second order. The intuition behind the Adam is that we don't want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search. In addition to storing an exponentially decaying average of past squared gradients like **AdaDelta**, ***Adam*** also keeps an exponentially decaying average of past gradients **M(t).**

M(t) and V(t) are values of the first moment which is the ***Mean*** and the second moment which is the ***uncentered variance*** of the gradients respectively.

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

First and second order of momentum

Here, we are taking mean of **M(t)** and **V(t)** so that **E[m(t)]** can be equal to **E[g(t)]** where, **E[f(x)]** is an expected value of **f(x)**.

To update the parameter:

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

Update the parameters

The values for β1 is 0.9 , 0.999 for β2, and (10 x exp(-8)) for '**∈**'.

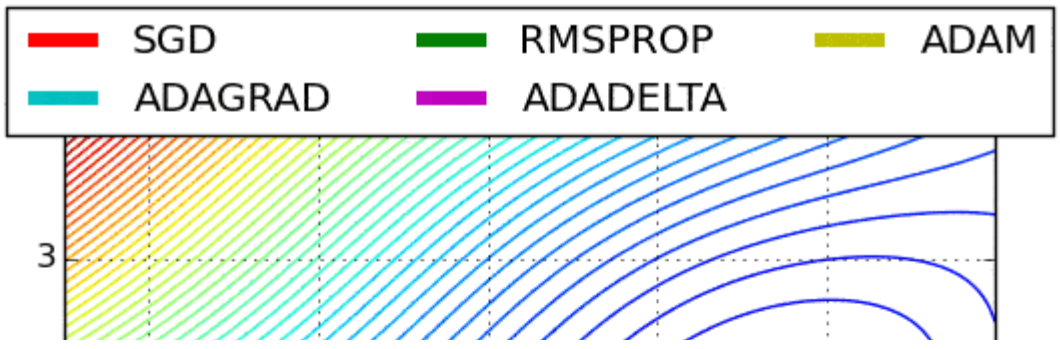
Advantages:

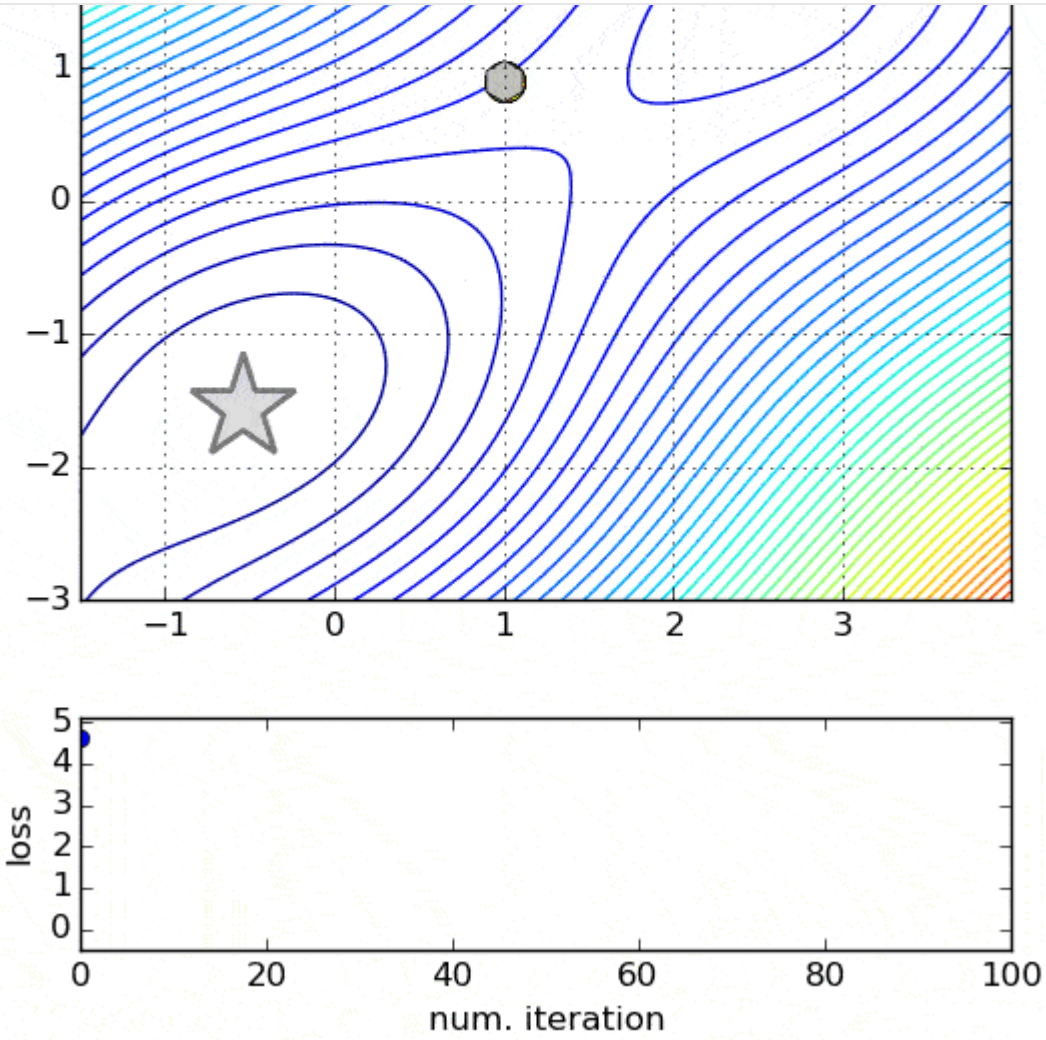
1. The method is too fast and converges rapidly.
2. Rectifies vanishing learning rate, high variance.

Disadvantages:

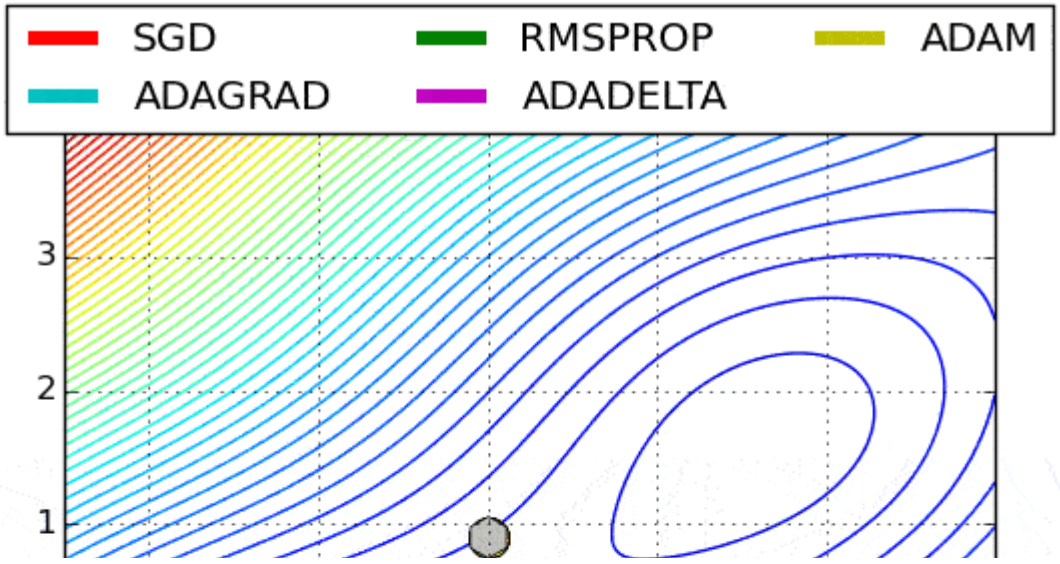
Computationally costly.

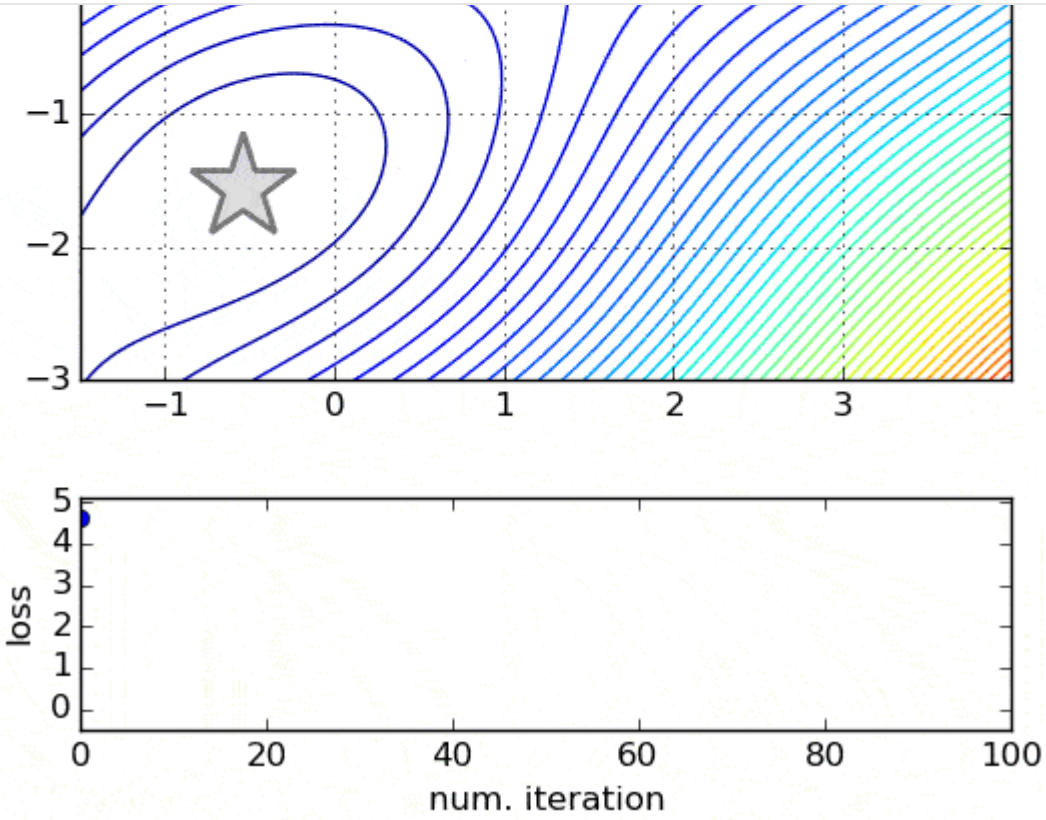
Comparison between various optimizers





Comparison 1





comparison 2

Conclusions

Adam is the best optimizers. If one wants to train the neural network in less time and more efficiently than Adam is the optimizer.

For sparse data use the optimizers with dynamic learning rate.

If, want to use gradient descent algorithm than min-batch gradient descent is the best option.

I hope you guys liked the article and were able to give you a good intuition towards the different behaviors of different Optimization Algorithms.