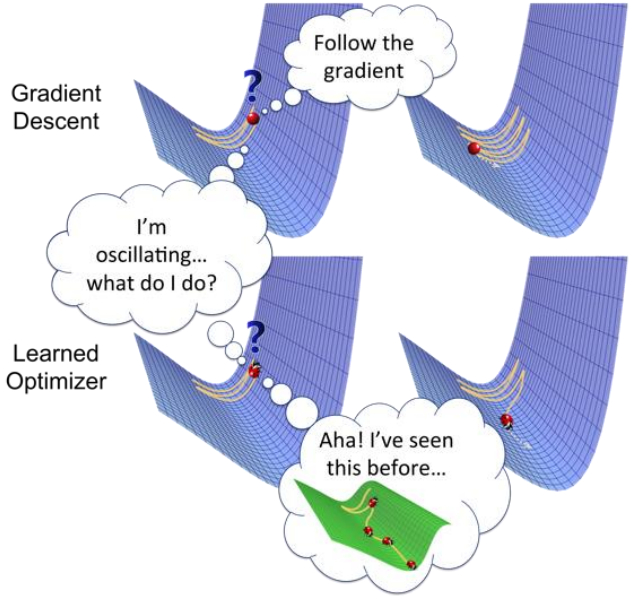
# Optimization in Deep Learning

Kokoy Siti Komariah
09.07.2020

# Introduction

- Optimizers are algorithms or methods used to change the attributes of neural network such as weights and learning rate in order to reduce the losses.
- And how you should change your weights or learning rates of your neural network to reduce the losses is defined by the optimizers you use. Optimizers are responsible for reducing the losses and to provide the most accurate results possible.
- One of the example of optimization algorithm is Gradient Descent.
- It is the most basic but most used optimizer. Gradient Descent is an iterative machine learning optimization algorithm to reduce the cost function. This will help models to make accurate predictions.



# Gradient Descent

- Gradient descent is a way to minimize an object function $J(\theta)$.
- It start with a random point on the function and move in the negative direction of the gradient of the function to reach the **local/global minima**.
- Updates parameters in opposite direction of gradient.

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

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

- It 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.

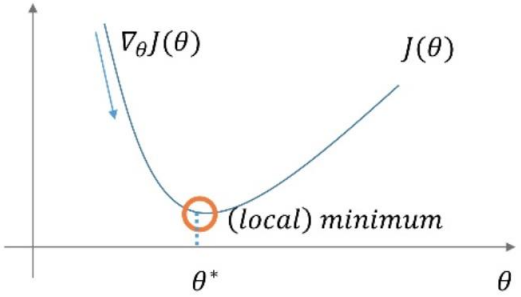



Figure 1: Optimization with GD (source: Sebastian Ruder)

# Gradient Descent

## Advantage:

- ➔ Easy computation.
- ➔ Easy to implement.
- ➔ Easy to understand.

## Disadvantage:

- ➔ May trap at local minima.
- ➔ 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.
- ➔ Requires large memory to calculate gradient on the whole dataset.

# 3 Variants of GD

- Batch Gradient Descent
- Stochastic Gradient Descent (SGD)
- Mini-batch Gradient Descent

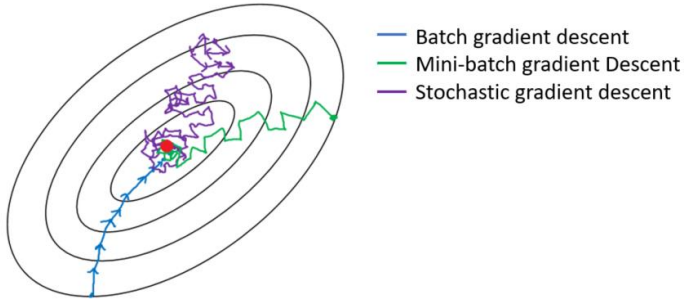
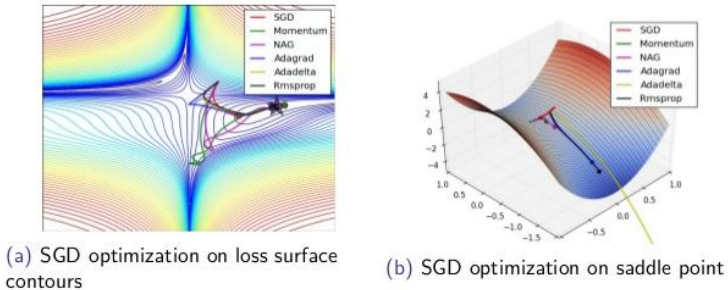



Figure 2: Gradient descent variants' trajectory towards minimum

# SGD Variants

- SGD with Momentum
- Nesterov Accelerated Gradient (NAG)
- AdaGrad
- AdaDelta
- RMSprop
- Adam




(a) SGD optimization on loss surface contours

(b) SGD optimization on saddle point

Figure 3: Visualization of algorithms (source: Alec Radford)

# SGD with 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 '$\gamma$'.

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

Momentum Gradient descent takes gradient of previous time steps into consideration

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

Figure 4: Source: Genevieve B. Orr

- Now, the weights are updated by $\theta = \theta - v_t$ .
- The momentum term $\gamma$ is usually set to ***0.9*** or a similar value.

# SGD with Momentum

## Advantage:

- ➔ Reduces the oscillations and high variance of the parameters.
- ➔ Converges faster than gradient descent.

## Disadvantage:

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

# Nesterov Accelerated Gradient (NAG)

- 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.
- In Nesterov momentum, instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate gradient at this "looked ahead" position.

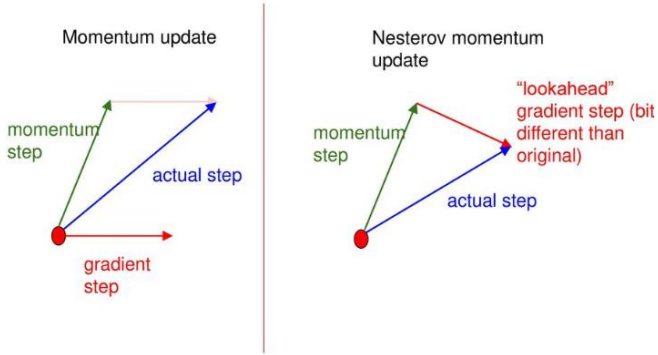



Figure 5: Nesterov momentum or Nesterov Accelerated Gradient (NAG)

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

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

Nesterov Accelerated Gradient

**Advantage:**

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

**Disadvantage:**

- Still, the hyperparameter needs to be selected manually.

# AdaGrad

- One of the disadvantages of all the optimizers explained above is that the learning rate is constant for all parameters and for each cycle.
- This optimizer changes the learning rate. It changes the learning rate '$\eta$' for each parameter and at every time step '$t$'. It's a type second order optimization algorithm. It works on the derivative of an error function.
- It makes big updates for less frequent parameters and a small step for frequent parameters.

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

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

Adagrad

**Advantage:**

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

**Disadvantage:**

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

# AdaDelta

- **Adadelta** is an extension of Adagrad and it also tries to reduce Adagrad's aggressive, monotonically reducing the learning rate.
- Instead of accumulating all previously squared gradients, Adadelta limits the 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.
- 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.

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

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

**Advantage:**

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

**Disadvantage:**

- Computationally expensive.

# RMSprop

- **Root Mean Square Propogation (RMSprop)** is an unpublished, adaptive learning rate method proposed by Geoff Hinton. It was developed at the same time with AdaDelta.
- 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

- Adam (Adaptive Moment Estimation) works with momentums of first and second order. It is combines the heuristics of both *Momentum* and *RMSProp.*
- 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.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$m_t$ and $v_t$ are estimates of first and second moment respectively

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{V_t}{1-\beta_2^t}$$

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

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

# Conclusions

- Optimizers are a crucial part of the neural network, understanding how they work would help you to choose which one to use for your application.
- Adam is the best optimizers. If one wants to train the neural network in less time and more efficiently.
- For sparse data use the optimizers with dynamic learning rate such as AdaGrad in GloVe embedding.
- If you want to use gradient descent algorithm than mini-batch gradient descent is the best option.

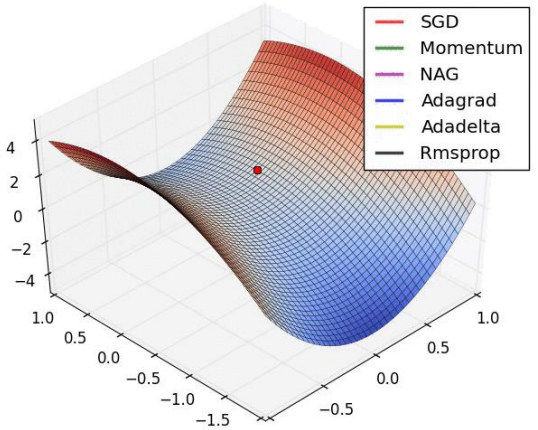



Figure 6: Long Valley Problem

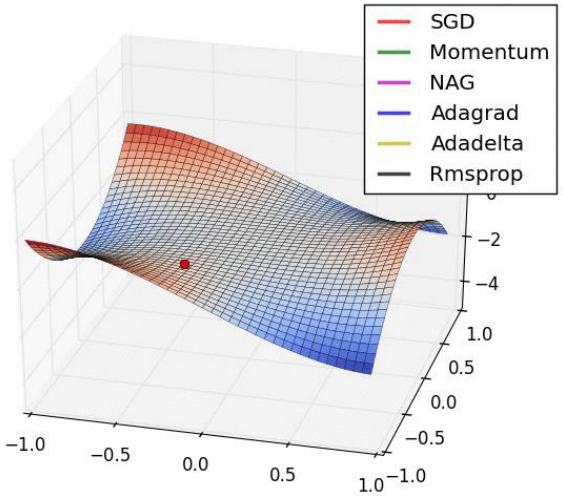



Figure 7: Saddle Point Problem

# Thank You!