An Overview of Gradient Descent Optimization Algorithms

Presenter: Ceyer Wakilpoor

Sebastian Ruder

Insight Centre for Data Analytics

June 2017

Outline

Outline

Introduction

- Gradient descent optimizers are commonly used as black-box optimizers
- The goal of this paper is to provide intuitions regarding the behaviour of different algorithms in practice
- Goes over the motivation behind different algorithms and their derivation

Outline

Basics

- Gradient descent is a way to minimize an objective function $J(\theta)$
- Updating θ in the direction opposite of the gradient of $J(\theta)$ w.r.t. θ
- There's a learning rate η that scales how far in the negative gradient direction you update the weights
- $\theta_+ = \theta_0 - \eta \cdot \frac{\partial J(\theta)}{\partial \theta}$

Outline

Outline

Batch Gradient Descent

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

- Most straight forward GD method, update parameters once per iteration of whole training dataset
- Intractable when whole data set can't fit in memory
- Can't train *online* (with new examples on-the-fly)
- Batch GD is guaranteed to converge to the global minimum for convex error surfaces and to local minimum otherwise

Stochastic Gradient Descent

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

- SGD performs an update for every training example, which means you can do online training
- SGD updates have much higher variance which causes the objective function to fluctuate heavily

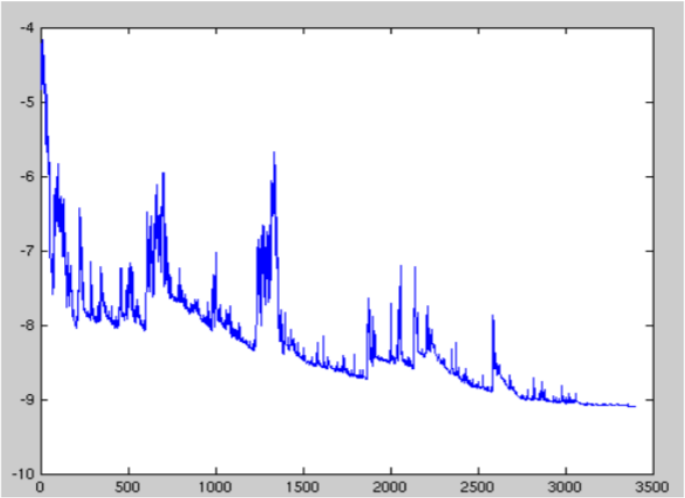


Figure 1: SGD fluctuation (Source: Wikipedia)

Stochastic Gradient Descent

- It is much faster since there are fewer repeated gradient computations; this happens because the weights are changed after every training example
- The large fluctuations can be useful in getting to better local minimum, but for convergence to an exact minimum it can be worse
  - Needed to decrease learning rate through steps in order to match the convergence claims of batch gradient descent

Mini-Batch Gradient Descent

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

- Best of both worlds: performs update for every mini-batch of n training examples
  - This will reduce the variance of the parameter updates, leading to a more stable convergence
  - Can avoid redundant computations and makes use of highly optimized matrix optimizations common state-of-the-art deep learning libraries
- Common mini-batch sized range between 50 and 256
- SGD is commonly used to refer to mini-batch GD as well
- Vanilla mini-batch GD does not guarantee good convergence

Code

- Batch GD

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

- SGD

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

- Mini-batch GD

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(loss_function, batch, params)
    params = params - learning_rate * params_grad
```

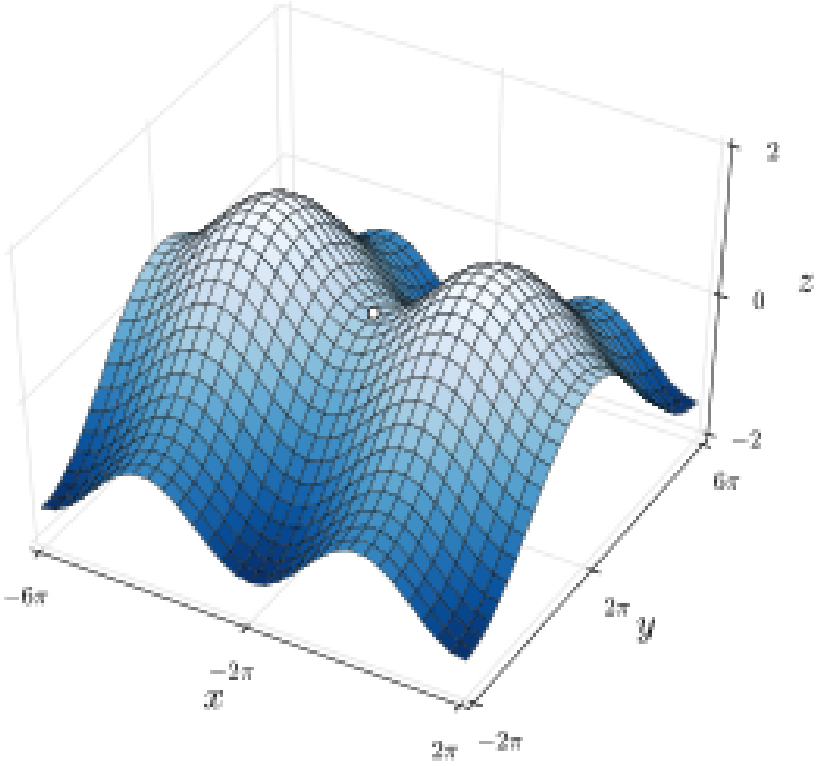
Outline

Limitations

- Choosing a proper learning rate: too small will take too long and too large can lead to divergence
- Constant learning rate through learning process usually is not ideal, so need to schedule learning rate changes in a predefined way
  - Fail to take into account properties of data, may want to update more for rarely occurring features
- Deep learning leads to very complex non-convex error functions
  - Get stuck in local minima, or more commonly in saddle points

Saddle Points

- A point where one dimension slopes up while another slopes down, usually surrounded by a plateau of about equal error
- Regardless of the direction GD goes, it is difficult to escape because the surrounds gradients are usually around zero

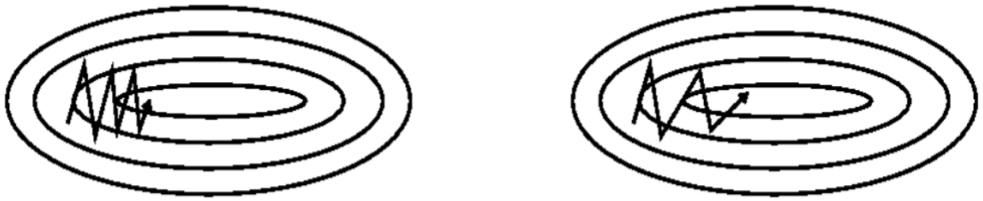


Outline

Momentum

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

- γ is usually selected to be around .9
- SGD has trouble navigating areas where the surface curves more steeply in one dimension than in another (ravines)
- This is common around local minima
- Like a ball rolling down a valley, increase gradient for dimension that stays constant and decrease for the dimension that changes direction
- Leads to faster convergence and fewer oscillations



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

Figure 2: Source: Genevieve B. Orr

Nesterov Accelerated Gradient

- Notion of ball knowing to slow down when hill slopes up again
- Only difference is the gradient is computed of the predicted next parameter values (looking ahead)

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

$$\theta = \theta - v_t$$

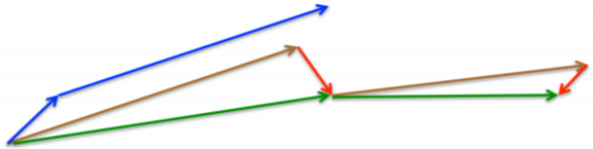


Figure 3: Nesterov update (Source: G. Hinton's lecture 6c)

- Prevents us from going too fast and results in significant increase in performance of RNNs

Adagrad

- Allows individualized parameter update depending on importance, larger updates for infrequent parameters and smaller updates for frequent ones
- Well suited for sparse data, uses different learning rate for every parameter at every time step

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

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

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

Adagrad

- $G_t \in \mathbb{R}^{dxd}$ where each diagonal element i, i is the sum of the squared for the gradients w.r.t. θ
- ϵ is the smoothing term to avoid dividing by zero
- Learning rate usually by default set to .01
- Accumulates squared gradients in the denominator so can stop learning eventually

Adadelta

- Works to solve the issue of monotonically decreasing learning rate
- Fixes sum of gradients window to some fixed size w
- In order to avoid storing all the gradients, just use a decaying average:

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

- keep γ usually around .9 similar to momentum
- Simply replace our previous Adagrad update rule with an exponentially decaying average

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

Adadelta

- They wanted the update units to match the parameter so they used the square of the parameters instead of the gradients in lieu of the learning rate:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{(E[\Delta\theta^2]_t + \epsilon}$$

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

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

RMSProp

- Identical to the first steps of derivation for Adadelta

$$E[g^t]_t = .9E[g^2]_{t-1} + .1g_t^2$$

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

- Suggested γ of .9 and η of .001
- Avoids radically diminishing learning rate from Adagrad

Adam

- Adaptive Moment Estimation
- In addition to storing exponentially decaying average of past squared gradients, v_t, also keeps exponentially decaying average of past gradients, m_t

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

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

- Estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradient respectively

Adam

- Avoid zero bias, especially at initial time steps:

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

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

- Yields final update rule:

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

- Estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradient respectively
- Suggested default values of .9 for β_1, .999 for β_2, and 10^{-8} for ϵ
- Shown to work better than other methods

AdaMax

- Scales gradient inversely proportionally to the l_2 norm of the past gradients

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

- Using l_∞:

$$u_t = \beta_2^\infty v_{t-1} + (1 - \beta_2^\infty)|g_t|^\infty$$
$$= max(\beta_2 \cdot v_{t-1}, |g_t|)$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t}\hat{m}_t$$

- using the max operation avoids bias towards zero
- Good default values are $\eta = .002$, $\beta_1 = .9$, and $\beta_2 = .999$

Nadam

- Combine NAG and Adam, first incorporate NAG into Adam:

$$g_t = \nabla_{\theta_t} J(\theta_t - \gamma m_{t-1})$$

$$m_t = \gamma m_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - m_t$$

- To avoid extra computation, we can use m_t to look ahead instead of computing the momentum for t-1 and t

$$g_t = \nabla_{\theta_t} J(\theta_t)$$

$$m_t = \gamma m_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - (\gamma m_t + \eta g_t)$$

Nadam

- Including Nesterov momentum to Adam, first take our previous derivation of Adam and expand the terms:

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

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

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

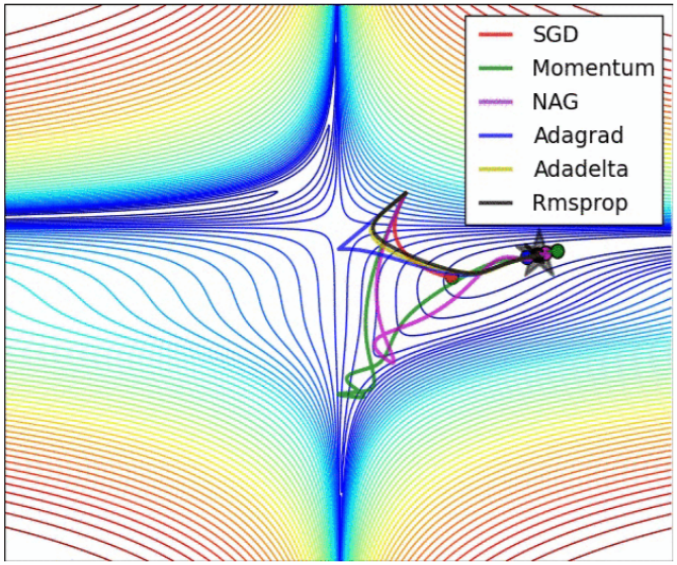
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} (\beta_1 \hat{m}_{t-1} + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t})$$

- Look ahead like we did on the previous slide:

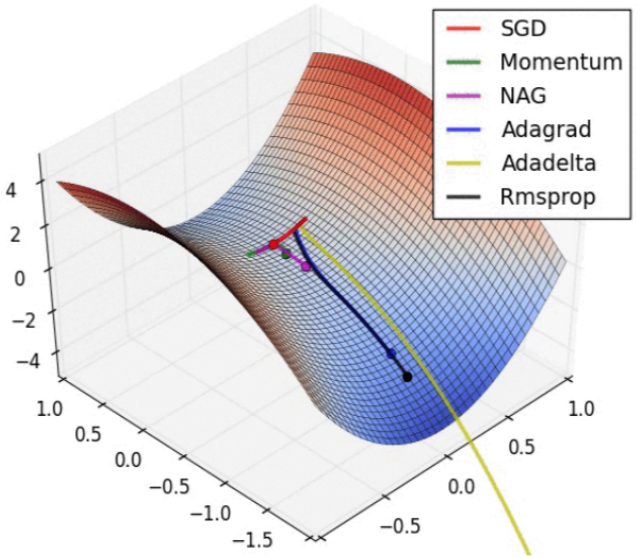
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} (\beta_1 \hat{m}_t + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t})$$

Outline

Visualization



(a) SGD optimization on loss surface contours



(b) SGD optimization on saddle point

Figure 4: Source and full animations: Alec Radford

Outline

What to Use

- For sparse data use an adaptive learning-rate method
- RMSprop is an extension of Adagrad but fixes the diminishing learning rate issue
- Adadelta is like RMSprop uses RMS of parameter updates in in numerator of update rule
- Adam adds bias-correction and momentum to RMSProp, Adam generally performs slightly better especially towards the end as gradients become sparser
- Vanilla SGD can be effective, but take a long time and is sensitive to annealing and initialization
- For faster convergence and for deep, complex neural networks use one of the adaptive learning rate methods

Outline

Parallelizing and Distributing SGD

- Hogwild!
  - Processors are allowed to access shared memory without locking parameters, works well with sparse data, allows SGD updates in parallel on CPUs
  - Achieves almost optimal rate of convergence, as it is unlikely that processors will overwrite useful info
- DownpourSGD
  - Multiple replicas of model ran in parallel
  - Risk of divergence from each other since information isn't shared
  - Each device solves subset
- TensorFlow
  - Utilizes computation graph which uses Send/Receive node pairs between devices
- Elastic Averaging SGD
  - Central server for parameters, meant to keep local variables further from center variable

Outline

Additional Strategies

- Shuffling and Curriculum Learn
  - Shuffle data in between epochs
  - For certain difficult problems, training examples can be presented in meaningful order: Curriculum Learning
- Batch Normalization
  - Normalize initial values of parameters by initializing them with zero mean and unit variance, but we lose normalization as we train
  - Reestablish normalization for every mini-batch, can avoid the need of dropout
- Early Stopping
  - Should be done when validation error stops improving
- Gradient Noise
  - Adding noise that follow Gaussian distribution $N(0, \sigma_t^2)$ to each update

$$g_{t,i} = g_{t,i} + N(0, \frac{\eta}{(1+t)^\gamma})$$

Citations

https://en.wikipedia.org/wiki/Saddle_point