Neural Networks - Part 3

Yuntian Deng

Lecture 21

Slides modified from Lecture 6 of CMSC 35246
(Shbhendu & Risi; University of Chicago)

Outline

Learning Goals

- Stochastic Gradient Descent
- Momentum Method and the Nesterov Variant
- Adaptive Learning Methods (AdaGrad, RMSProp)
- Adaptive Moments (Adam)

Learning Goals

Batched Gradient Descent

Momentum

Adaptive Method

Adam Optimizer

Optimization

- We've seen back-propagation as a method for computing gradients.
- Let's see a family of first-order optimization methods.

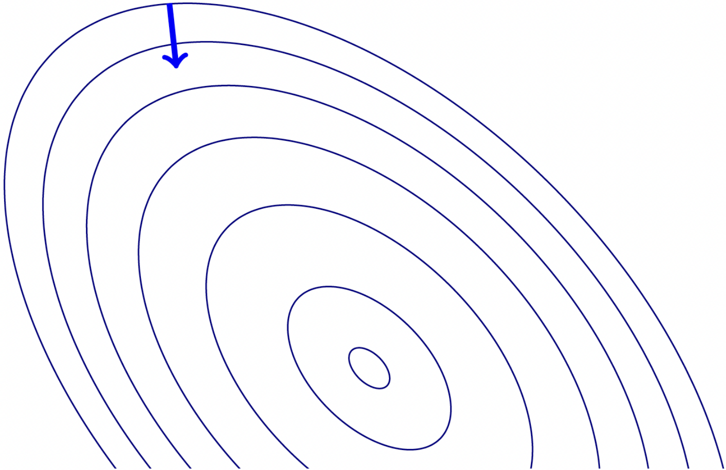
Gradient Descent

Algorithm 1 Batch Gradient Descent at Iteration k

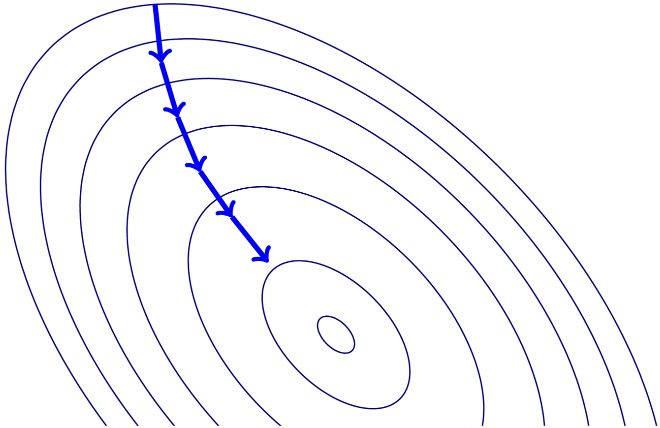
Require: Learning rate ϵ_k
Require: Initial Parameter θ
1: **while** stopping criteria not met **do**
2: Compute gradient estimate over N examples:
3: $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
4: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
5: **end while**

- Positive: Gradient Estimates are stable
- Negative: Need to compute the gradients over the entire training for one update.

Gradient Descent



Gradient Descent



Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ

```
while stopping criteria not met do
    Sample example (x^(i), y^(i)) from training set
    Compute gradient estimate:
    ĝ ← +∇_θ L(f(x^(i); θ), y^(i))
    Apply Update: θ ← θ − εĝ
end while
```

- ϵ_k is the learning rate.
- Sufficient Condition to guarantee convergence:

$$\sum_{k=1} \epsilon_k = \infty \ \& \ \sum_{k=1} \epsilon_k^2 < \infty$$

Stochastic Gradient Descent

- In practice the learning rate is decayed linearly till iteration τ

  $$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$$

  with $\alpha = \frac{k}{\tau}$

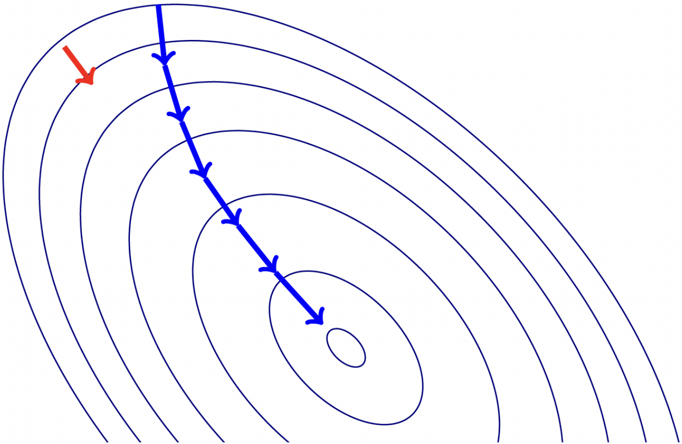
- τ is usually set to the number of iterations needed for a large number of passes through the data

- ϵ_τ should roughly be set to a small number.

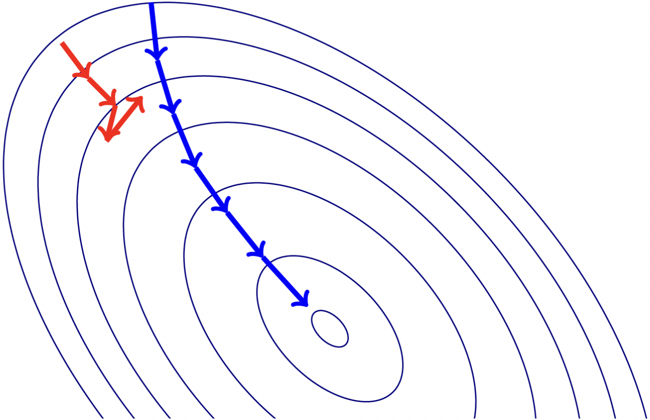
Stochastic Gradient Descent

- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use large mini-batches
- Advantage: Computation time per update does not depend on the number of training examples N
- This allows convergence on extremely large datasets

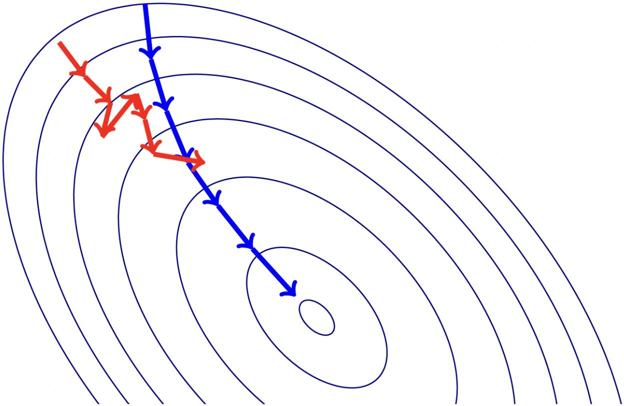
Stochastic Gradient Descent



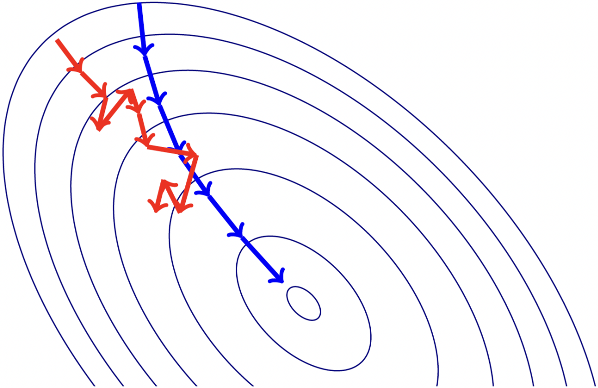
Stochastic Gradient Descent



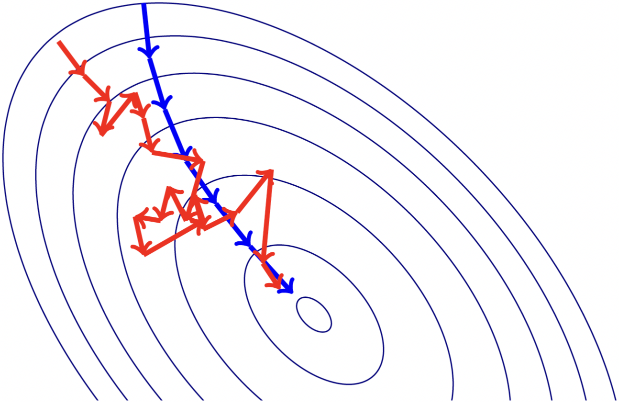
Stochastic Gradient Descent



Stochastic Gradient Descent



Stochastic Gradient Descent



Batch Gradient Descent

- Batch Gradient Descent:

$$\hat{g} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$$

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

- SGD:

$$\hat{g} \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$$

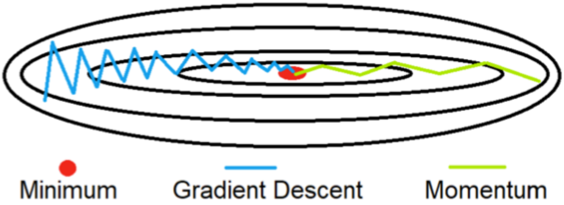
$$\theta \leftarrow \theta - \epsilon \hat{g}$$

What's wrong with SGD

- Momentum is an extension of gradient descent optimization, which builds inertia in a search direction to overcome local minima and oscillation of noisy gradients. It's based on the same concept of momentum in physics.
- With gradient descent, a weight update at time t is given by the learning rate and gradient at that exact moment. It means that the previous steps are not considered in the next iteration.
- Two issues:
  - Unlike convex functions, a non-convex function can have many local minima, the gradient becomes so small to get stuck
  - Gradient descent can be noisy with many oscillations which results in a larger number of iterations needed to reach convergence

Momentum

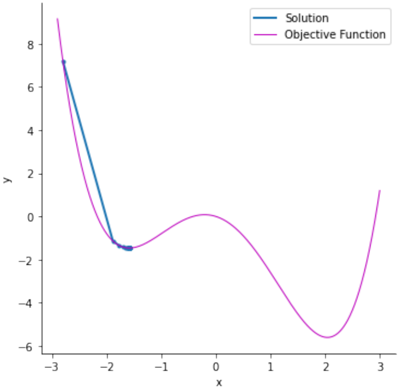
- Momentum is able to solve both of these issues by using an exponentially weighted average of the gradients to update the weights at each iteration.
- This method also prevents gradients of previous iterations to be weighted equally. With an exponentially weighted average, recent gradients are given more weight than previous ones.



Example

To demonstrate the use of momentum in the context of gradient descent, minimize the following function:

$$y = 0.3x^4 - 0.1x^3 - 2x^2 - 0.8x$$



Example cont.

$$g_i = \nabla_\theta f(\theta_{i-1}) = 1.2\theta_{i-1}^3 - 0.3\theta_{i-1}^2 - 4\theta_{i-1} - 0.8$$

$$\theta_i = \theta_{i-1} - \epsilon * g_i$$

Iteration	g_i	θ_i
1	-18.2	-1.885
2	-2.36	-1.76
3	-1.2	-1.70
4	-0.78	-1.66
...	...	...
99	0.0002	-1.586

Example cont.

- How do we try and solve this problem?
- Introduce a new variable v, the velocity
- We think of v as the direction, and speed by which the parameters move as the learning dynamics progress
- The velocity is an exponentially decaying moving average of the negative gradients:

$$v_i = \alpha v_{i-1} - \epsilon \nabla_\theta f(\theta_{i-1})$$

- $\alpha \in [0, 1)$, Update rule: $\theta_i \leftarrow \theta_{i-1} + v_i$

Example cont.

$$g_i = \nabla_\theta f_i(\theta_{i-1}) = 1.2\theta_{i-1}^3 - 0.3\theta_{i-1}^2 - 4\theta_{i-1} - 0.8$$

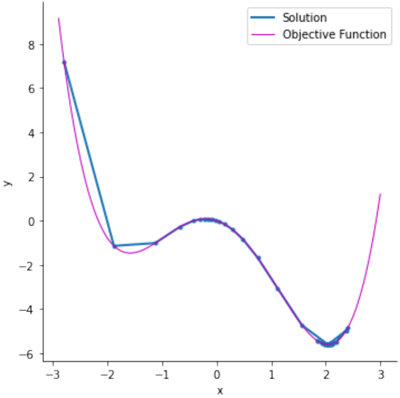
$$v_i = \alpha v_{i-1} - \epsilon g_i$$

$$\theta_i = \theta_{i-1} + v_i$$

Iteration	g_i	v_i	θ_i
1	-18.2	0	-1.885
2	-2.36	-15.1	-1.12
3	1.61	-9.0	-0.67
4	1.39	-9.2	-0.21
...	...	...	...
99	0.0002	0.0003	2.042

Example cont.

Escaping the local minima with momentum, and then settling down to the global minima.



Momentum

- Velocity Term:

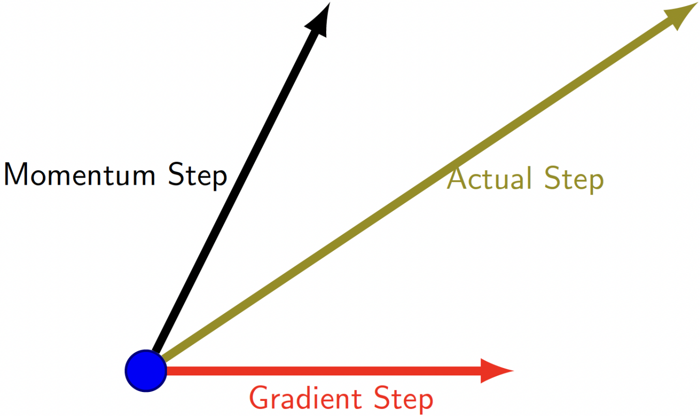
$$\boldsymbol{v} = \alpha \boldsymbol{v} - \epsilon \nabla_\theta (L(f(x^{(I)}; \theta), y^{(i)}))$$

- Update Term:

$$\theta_i = \theta_{i-1} + \boldsymbol{v}$$

- The velocity accumulates the previous gradients
- What is the role of α?
  - If α is larger than ϵ the current update is more affected by the previous gradients.
  - Usually values for α are set high

Momentum



Momentum

- In SGD, the step size was the norm of the gradient scaled by the learning rate, which is $\epsilon||g||$.
- While using momentum, the step size will also depend on the norm of a sequence of gradients.
- The step size becomes:

$$\epsilon||g_1|| + \alpha\epsilon||g_2|| + \alpha^3\epsilon||g_3|| + \cdots + \alpha^K\epsilon||g_K||$$

- Therefore, the stepsize is roughly $\epsilon\frac{||\hat{g}||}{1-\alpha}$
- If $\alpha = 0.9$, multiply the maximum speed by 10 relative to the current gradient direction.

SGD with Momentum

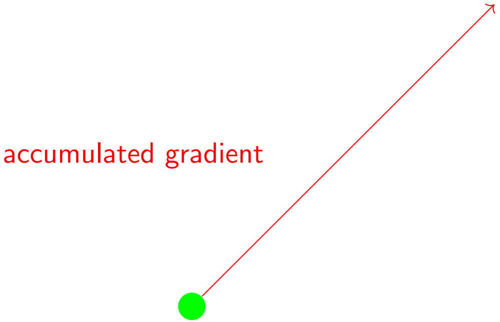
Algorithm 2 Stochastic Gradient Descent with Momentum

Require: Learning rate ϵ_k
Require: Momentum Parameter α
Require: Initial Parameter θ
Require: Initial Velocity $\mathbf{v}$

while stopping criteria not met **do**
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate:
    $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Compute the velocity update:
    $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$
    Apply Update: $\theta \leftarrow \theta + \mathbf{v}$
end while

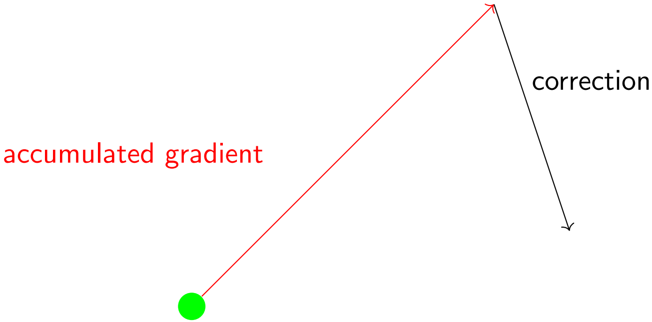
Nesterov Momentum

- Another approach: First take a step in the direction of the accumulated gradient
- Then calculate the gradient and make a correction



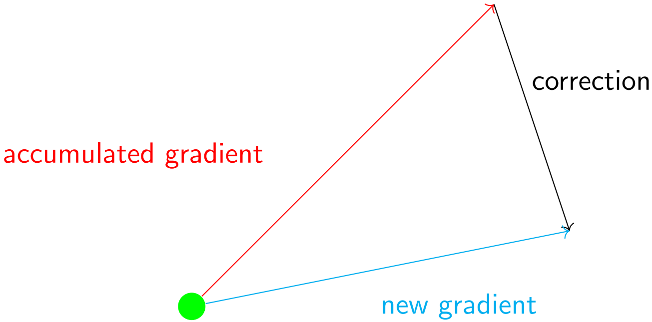
Nesterov Momentum

- Another approach: First take a step in the direction of the accumulated gradient
- Then calculate the gradient and make a correction



Nesterov Momentum

- Another approach: First take a step in the direction of the accumulated gradient
- Then calculate the gradient and make a correction



Nestorv Momentum

- Recall the velocity term in the Momentum method:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_\theta (L(f(x^{(i)}; \theta), y^{(i)}))$$

- Nesterov Momentum:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_\theta (L(f(x^{(i)}; \theta + \alpha \boldsymbol{v}), y^{(i)}))$$

- Update: $\theta \leftarrow \theta + \boldsymbol{v}$

SGD with Nestorv Momentum

Algorithm 3 SGD with Nesterov Momentum

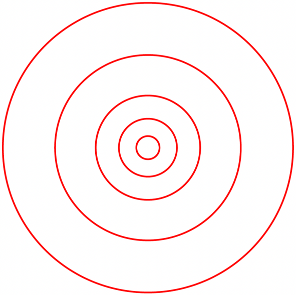
Require: Learning rate ϵ
Require: Momentum Parameter α
Require: Initial Parameter θ
Require: Initial Velocity $\mathbf{v}$

while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
  Update parameters: $\tilde{\theta} \leftarrow \theta + \alpha\mathbf{v}$
  Compute gradient estimate:
  $\hat{\mathbf{g}} \leftarrow +\nabla_{\tilde{\theta}} L(f(\mathbf{x}^{(i)}; \tilde{\theta}), \mathbf{y}^{(i)})$
  Compute the velocity update: $\mathbf{v} \leftarrow \alpha\mathbf{v} - \epsilon\hat{\mathbf{g}}$
  Apply Update: $\theta \leftarrow \theta + \mathbf{v}$
end while

Motivation

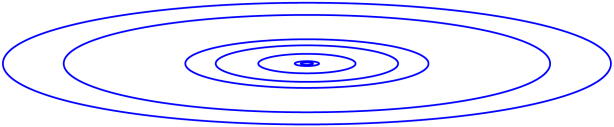
- Till now we assign the same learning rate to all the features
- If the features vary in importance and frequency, why is this a good idea?
- It's probably not!

Motivation



Nice (all features are equally important)

Motivation



Harder!

Motivation

- Downscale a model parameter by the square root of the sum of squares of all its historical values
- Parameters that have larger partial derivatives of the loss - learning rates for them rapidly declined
- The algorithm assigns higher learning rates to infrequent features, which ensures that the parameter updates rely less on frequency and more on relevance

AdaGrad (Adaptative Gradient)

Algorithm 1 Adaptative Gradient

Require: Global Learning rate ϵ, Initial Parameter θ, δ

```
Initialize r = 0
while stopping criteria not met do
    Sample example (x^(i), y^(i)) from training set
    Compute gradient estimate: ĝ ← +∇_θ L(f(x^(i); θ), y^(i))
    Accumulate: r ← r + ĝ ⊙ ĝ
    Compute update: Δθ ← −ε/(δ+√r) ⊙ ĝ
    Apply Update: θ ← θ + Δθ
```

Initialize $\boldsymbol{r} = 0$

while stopping criteria not met **do**

Sample example $(x^{(i)}, y^{(i)})$ from training set

Compute gradient estimate: $\hat{g} \leftarrow +\nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$

Accumulate: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \hat{g} \odot \hat{g}$

Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \hat{g}$

Apply Update: $\theta \leftarrow \theta + \Delta\theta$

RMSProp (Root Mean Square)

- AdaGrad is good when the objective is convex
- AdaGrad might shrink the learning rate too aggressively, we want to keep the history in mind.
- We can adapt it to perform better in a non-convex setting by accumulating an exponentially decaying average of the gradient
- This is an idea that we use again and again in Neural Networks

RMSProp (Root Mean Square)

Algorithm 2 Root Mean Square Propagation

Require: Global Learning rate ϵ, Initial Parameter ρ, θ, δ

1: Initialize $\boldsymbol{r}$=0
2: **while** stopping criteria not met **do**
3: Sample example $(x^{(i)}, y^{(i)})$ from training set
4: Compute gradient estimate: $\hat{g} \leftarrow +\nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$
5: Accumulate: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\hat{g} \odot \hat{g}$
6: Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta+\sqrt{\boldsymbol{r}}} \odot \hat{g}$
7: Apply Update: $\theta \leftarrow \theta + \Delta\theta$

AdaDelta (Adative Delta)

- It is similar to RMSProp as an improvement over AdaGrad
- It completely removes the usage of hand-set learning rate
- Using the difference between current weight and the newly updated weight as the learning rate

AdaDelta (Adative Delta)

Algorithm 3 Root Mean Square Propagation

Require: Initial Parameter ρ, θ, δ

```
Initialize r = 0, d = 0
while stopping criteria not met do
    Sample example (x^(i), y^(i)) from training set
    Compute gradient estimate: ĝ ← +∇_θ L(f(x^(i); θ), y^(i))
    Accumulate: r ← ρr + (1 − ρ) ĝ ⊙ ĝ
    Accumulate: d ← ρd + (1 − ρ)[Δθ]^2
    Compute update: Δθ ← −(δ+√d)/(δ+√r) ⊙ ĝ
    Apply Update: θ ← θ + Δθ
```

Initialize $\boldsymbol{r} = 0$, $\boldsymbol{d} = 0$

while stopping criteria not met **do**

Sample example $(x^{(i)}, y^{(i)})$ from training set

Compute gradient estimate: $\hat{g} \leftarrow +\nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$

Accumulate: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\hat{g} \odot \hat{g}$

Accumulate: $\boldsymbol{d} \leftarrow \rho \boldsymbol{d} + (1-\rho)[\Delta\theta]^2$

Compute update: $\Delta\theta \leftarrow -\frac{\delta+\sqrt{\boldsymbol{d}}}{\delta+\sqrt{\boldsymbol{r}}} \odot \hat{g}$

Apply Update: $\theta \leftarrow \theta + \Delta\theta$

Learning Goals

Batched Gradient Descent

Momentum

Adaptive Method

Adam Optimizer

Adam

The inspiration of Adam optimizer:

- AdaGrad (Adaptive Gradient Algorithm) maintains a per-parameter learning rate that improves the performance on problems with sparse gradients
- RMSProp (Root Mean Square Propagation) also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight.
- Momentum Method can maintain a velocity term to keep track of the history gradients.

Adam: ADAptive Moments

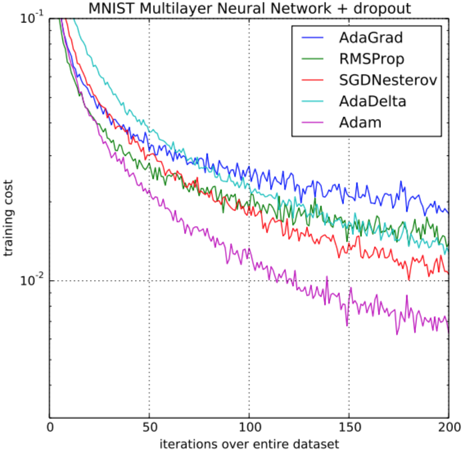
Algorithm 4 ADAptive Moments

Require: Learning Rate ϵ, Decay rates $\rho_1, \rho_2, \theta, \delta$

1: Initialize $\boldsymbol{s} = 0$, $\boldsymbol{r} = 0$, time step $t = 0$
2: **while** stopping criteria not met **do**
3: Sample example $(x^{(i)}, y^{(i)})$ from training set
4: Compute gradient estimate: $\hat{g} \leftarrow +\nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$
5: $t \leftarrow t + 1$
6: Update: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\hat{g}$
7: Update: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\hat{g} \odot \hat{g}$
8: Correct Biases: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}, \hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
9: Compute Update: $\Delta\theta = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$
10: Apply Update: $\theta \leftarrow \theta + \Delta\theta$

Performance

Adam optimizer is by far one of the most successful optimizers to achieve great performance. A standard benchmark to evaluate optimizer performance is MNIST:



Revisiting Learning Goals

- Stochastic Gradient Descent
- Momentum Method and the Nesterov Variant
- Adaptive Learning Methods (AdaGrad, RMSProp)
- Adaptive Moments (Adam)