Lecture 6
Optimization for Deep Neural Networks

CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

April 12, 2017

- Things we will look at today
  - Stochastic Gradient Descent

- Things we will look at today
  - Stochastic Gradient Descent
  - Momentum Method and the Nesterov Variant

- Things we will look at today
  - Stochastic Gradient Descent
  - Momentum Method and the Nesterov Variant
  - Adaptive Learning Methods (AdaGrad, RMSProp, Adam)

- Things we will look at today
    - Stochastic Gradient Descent
    - Momentum Method and the Nesterov Variant
    - Adaptive Learning Methods (AdaGrad, RMSProp, Adam)
    - Batch Normalization

- Things we will look at today
  - Stochastic Gradient Descent
  - Momentum Method and the Nesterov Variant
  - Adaptive Learning Methods (AdaGrad, RMSProp, Adam)
  - Batch Normalization
  - Intialization Heuristics

- Things we will look at today
  - Stochastic Gradient Descent
  - Momentum Method and the Nesterov Variant
  - Adaptive Learning Methods (AdaGrad, RMSProp, Adam)
  - Batch Normalization
  - Intialization Heuristics
  - Polyak Averaging

- Things we will look at today
  - Stochastic Gradient Descent
  - Momentum Method and the Nesterov Variant
  - Adaptive Learning Methods (AdaGrad, RMSProp, Adam)
  - Batch Normalization
  - Intialization Heuristics
  - Polyak Averaging
  - On Slides but for self study: Newton and Quasi Newton Methods (BFGS, L-BFGS, Conjugate Gradient)

Optimization

- We've seen backpropagation as a method for computing gradients

Optimization

- We've seen backpropagation as a method for computing gradients
- Assignment: Was about implementation of SGD in conjunction with backprop

Optimization

- We've seen backpropagation as a method for computing gradients
- Assignment: Was about implementation of SGD in conjunction with backprop
- Let's see a family of first order methods

Batch Gradient Descent

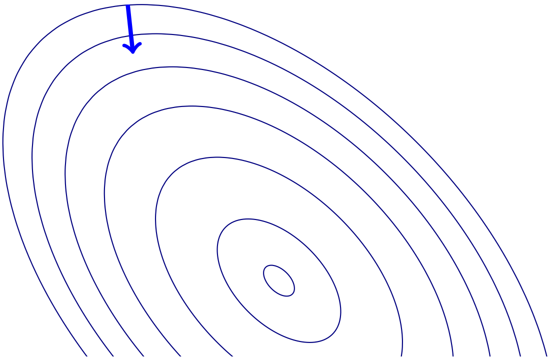
Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ

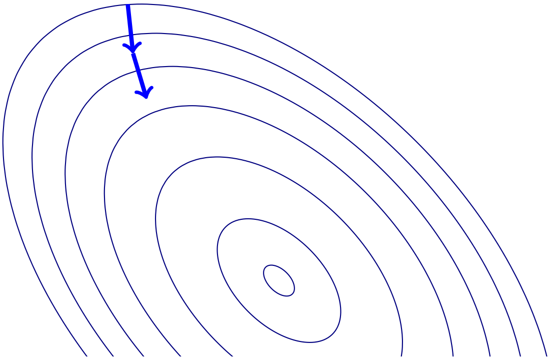
1: **while** stopping criteria not met **do**
2: Compute gradient estimate over N examples:
3: $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
4: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
5: **end while**

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

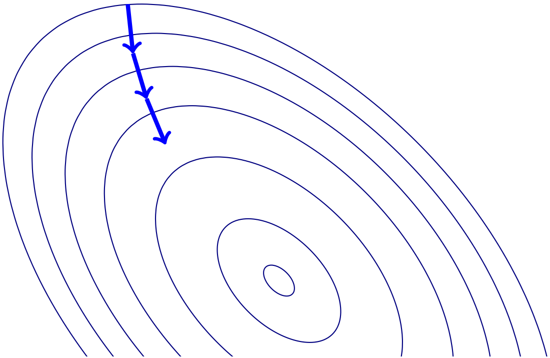
Gradient Descent



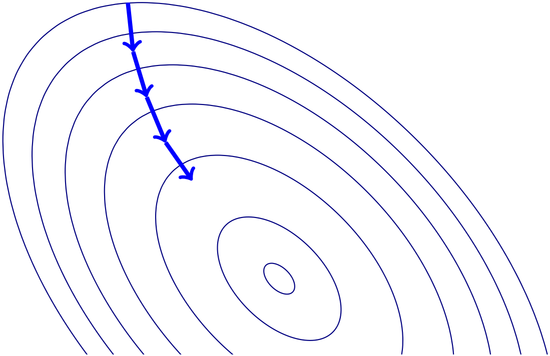
Gradient Descent



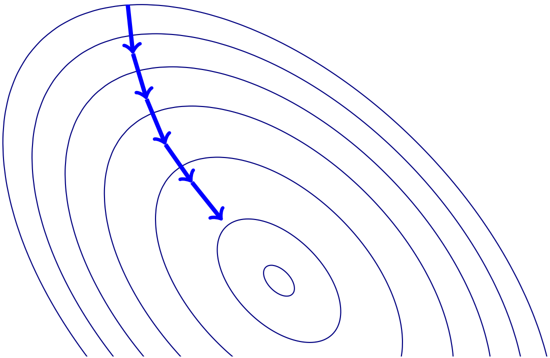
Gradient Descent



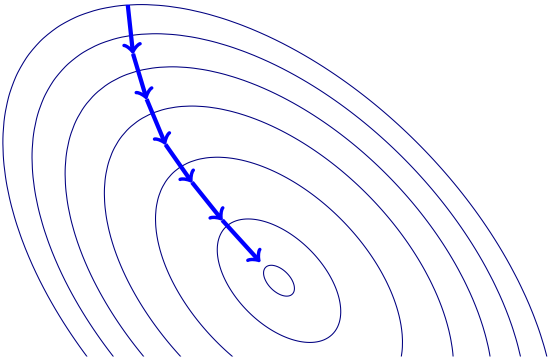
Gradient Descent



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
```

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

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 $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate:
    $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Apply Update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$
end while

- ϵ_k is learning rate at step k

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 $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set

  Compute gradient estimate:

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

  Apply Update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

end while

- ϵ_k is learning rate at step k
- Sufficient condition to guarantee convergence:

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

Learning Rate Schedule

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

Learning Rate Schedule

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

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \text{ with } \alpha = \frac{k}{\tau}$$

Learning Rate Schedule

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

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \text{ with } \alpha = \frac{k}{\tau}$$

- 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 1% of ϵ_0
- How to set ϵ_0?

Minibatching

- Potential Problem: Gradient estimates can be very noisy

Minibatching

- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use larger mini-batches

Minibatching

- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use larger mini-batches
- Advantage: Computation time per update does not depend on number of training examples N

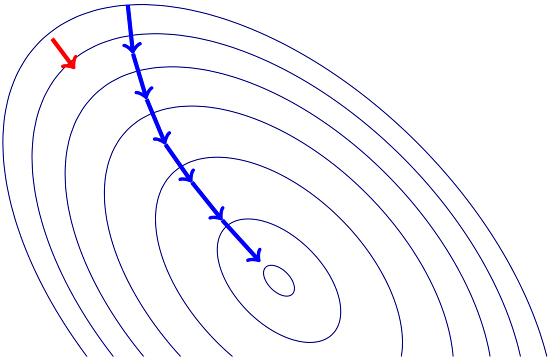
Minibatching

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

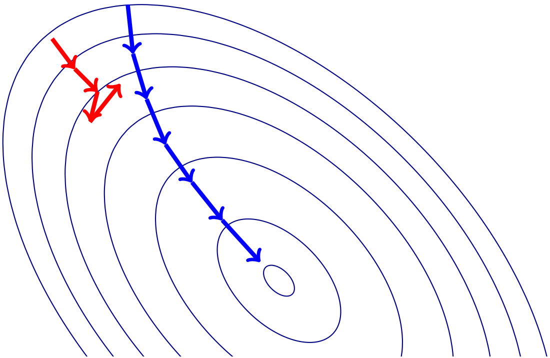
Minibatching

- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use larger mini-batches
- Advantage: Computation time per update does not depend on number of training examples N
- This allows convergence on extremely large datasets
- See: Large Scale Learning with Stochastic Gradient Descent by Leon Bottou

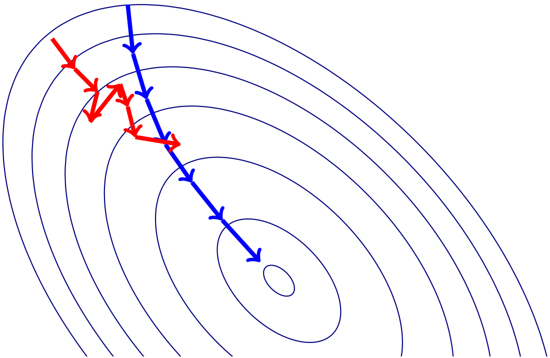
Stochastic Gradient Descent



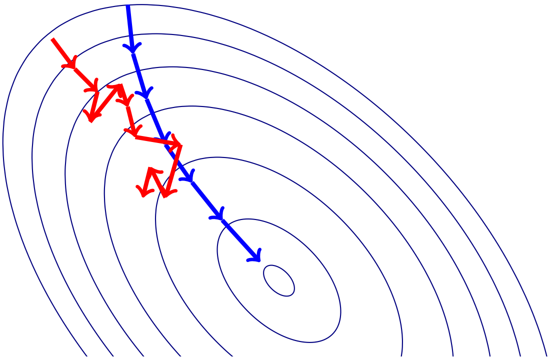
Stochastic Gradient Descent



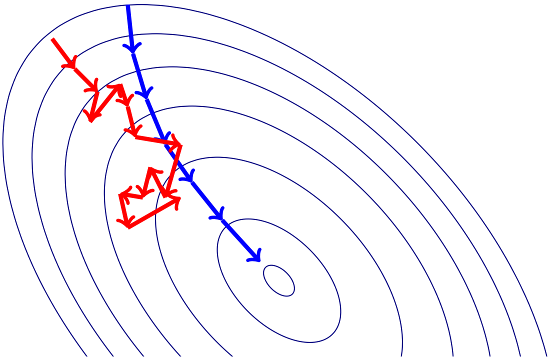
Stochastic Gradient Descent



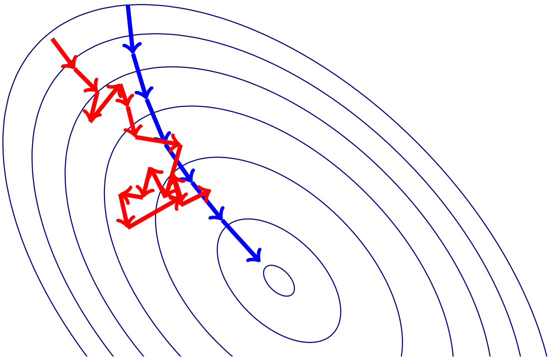
Stochastic Gradient Descent



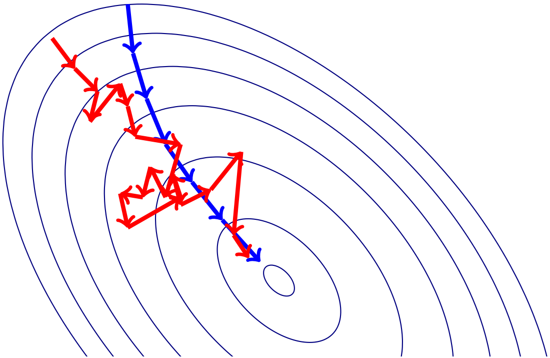
Stochastic Gradient Descent



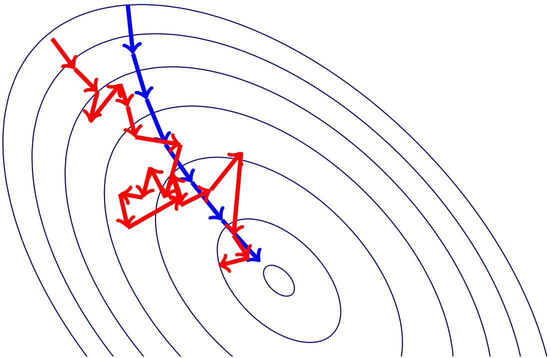
Stochastic Gradient Descent



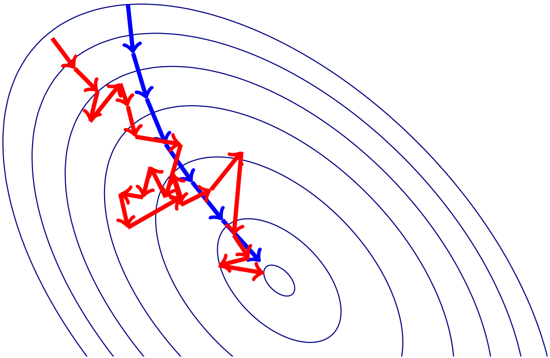
Stochastic Gradient Descent



Stochastic Gradient Descent



Stochastic Gradient Descent



So far..

- Batch Gradient Descent:

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

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

So far..

- Batch Gradient Descent:

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

- SGD:

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

Momentum

- The Momentum method is a method to accelerate learning using SGD

Momentum

- The Momentum method is a method to accelerate learning using SGD
- In particular SGD suffers in the following scenarios:
  - Error surface has high curvature

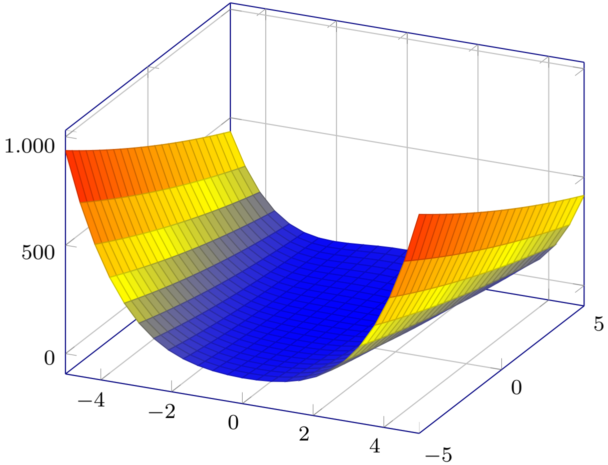
Momentum

- The Momentum method is a method to accelerate learning using SGD
- In particular SGD suffers in the following scenarios:
  - Error surface has high curvature
  - We get small but consistent gradients

Momentum

- The Momentum method is a method to accelerate learning using SGD
- In particular SGD suffers in the following scenarios:
  - Error surface has high curvature
  - We get small but consistent gradients
  - The gradients are very noisy

Momentum



- Gradient Descent would move quickly down the walls, but very slowly through the valley floor

Momentum

- How do we try and solve this problem?

Momentum

- How do we try and solve this problem?
- Introduce a new variable $\mathbf{v}$, the velocity

Momentum

- How do we try and solve this problem?
- Introduce a new variable $\mathbf{v}$, the velocity
- We think of $\mathbf{v}$ as the direction and speed by which the parameters move as the learning dynamics progresses

Momentum

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

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

- $\alpha \in [0, 1)$Update rule: $\theta \leftarrow \theta + \mathbf{v}$

Momentum

- Let's look at the velocity term:

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

Momentum

- Let's look at the velocity term:

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

- The velocity accumulates the previous gradients

Momentum

- Let's look at the velocity term:

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

- 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

Momentum

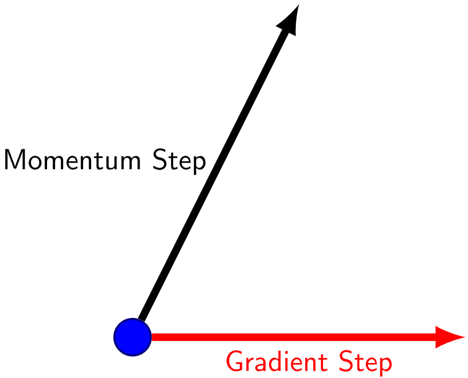
- Let's look at the velocity term:

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

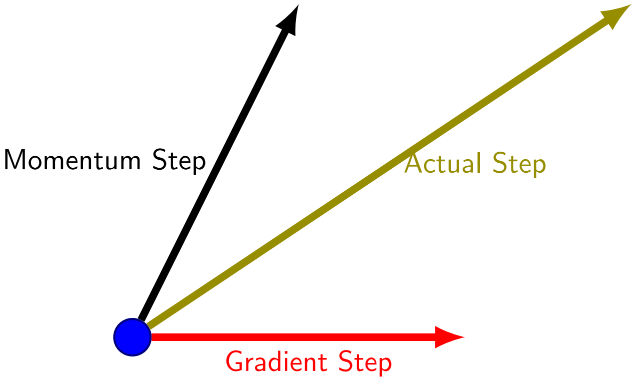
- The velocity accumulates the previous gradients
- What is the role of α?
  - If α is larger than ϵ the current update is more affected by the previous gradients
  - Usually values for α are set high $\approx 0.8, 0.9$

Momentum

Momentum



Momentum



Momentum: Step Sizes

- In SGD, the step size was the norm of the gradient scaled by the learning rate $\epsilon \|\mathbf{g}\|$. Why?

Momentum: Step Sizes

- In SGD, the step size was the norm of the gradient scaled by the learning rate $\epsilon\|\mathbf{g}\|$. Why?
- While using momentum, the step size will also depend on the norm and alignment of a sequence of gradients

Momentum: Step Sizes

- In SGD, the step size was the norm of the gradient scaled by the learning rate $\epsilon\|\mathbf{g}\|$. Why?
- While using momentum, the step size will also depend on the norm and alignment of a sequence of gradients
- For example, if at each step we observed $\mathbf{g}$, the step size would be (exercise!):

$$\epsilon\frac{\|\mathbf{g}\|}{1-\alpha}$$

Momentum: Step Sizes

- In SGD, the step size was the norm of the gradient scaled by the learning rate $\epsilon\|\mathbf{g}\|$. Why?
- While using momentum, the step size will also depend on the norm and alignment of a sequence of gradients
- For example, if at each step we observed $\mathbf{g}$, the step size would be (exercise!):

$$\epsilon\frac{\|\mathbf{g}\|}{1-\alpha}$$

- If $\alpha = 0.9 \implies$ multiply the maximum speed by 10 relative to the current gradient direction

Momentum

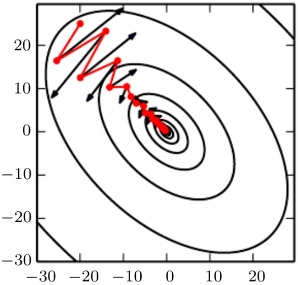


Illustration of how momentum traverses such an error surface better compared to Gradient Descent

SGD with Momentum

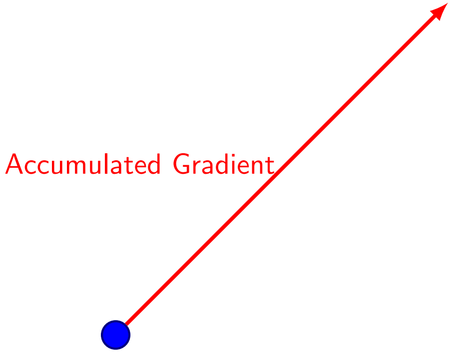
Algorithm 2 Stochastic Gradient Descent with Momentum

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 (x^(i), y^(i)) from training set
    Compute gradient estimate:
    ĝ ← +∇_θ L(f(x^(i); θ), y^(i))
    Compute the velocity update:
    v ← αv − εĝ
    Apply Update: θ ← θ + v
end while
```

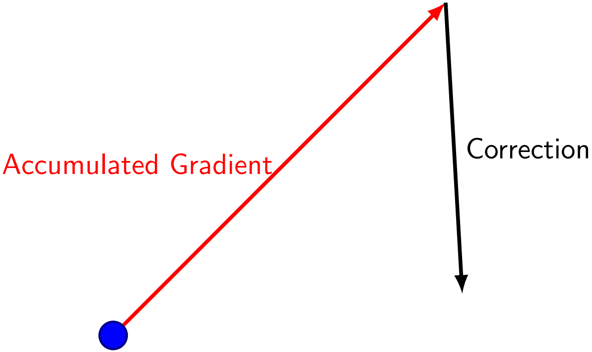
Nesterov Momentum

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



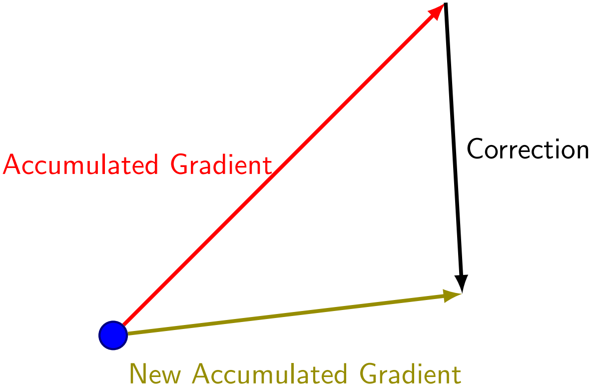
Nesterov Momentum

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



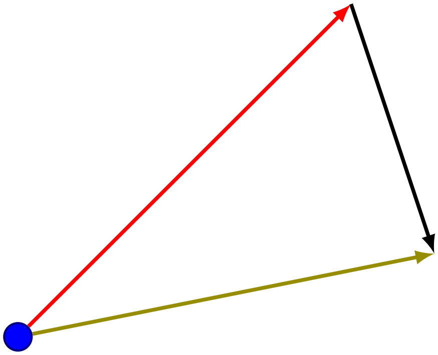
Nesterov Momentum

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



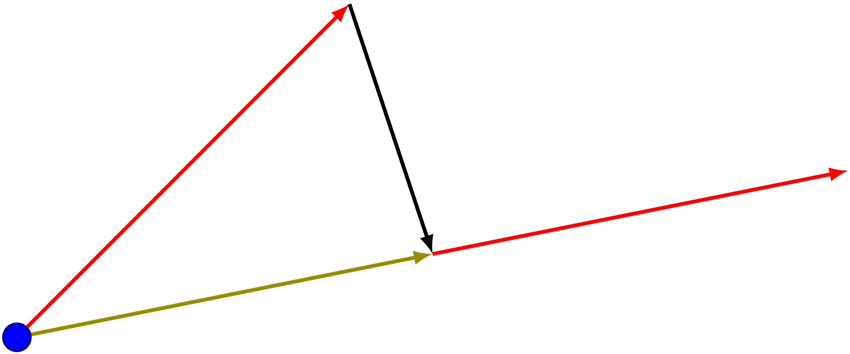
Nesterov Momentum

Next Step



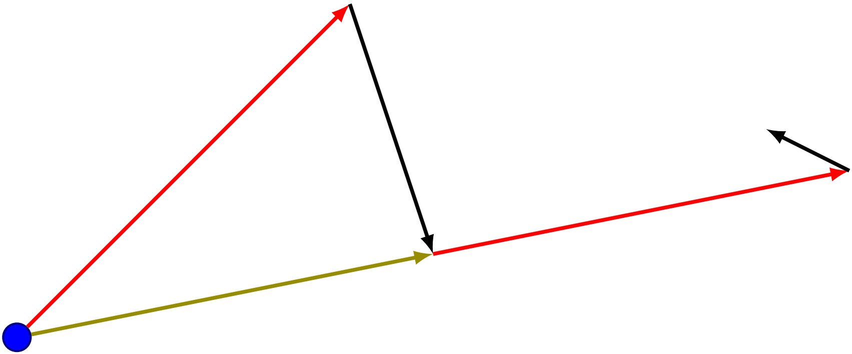
Nesterov Momentum

Next Step



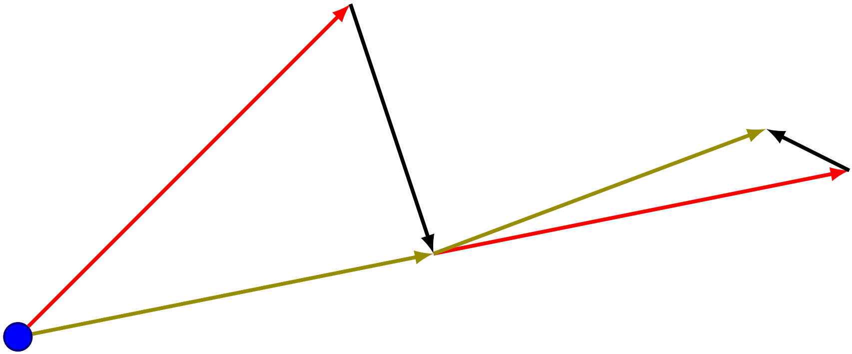
Nesterov Momentum

Next Step



Nesterov Momentum

Next Step



Let's Write it out..

- Recall the velocity term in the Momentum method:

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

Let's Write it out..

- Recall the velocity term in the Momentum method:

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

- Nesterov Momentum:

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

Let's Write it out..

- Recall the velocity term in the Momentum method:

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

- Nesterov Momentum:

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

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

SGD with Nesterov 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 (x^(i), y^(i)) from training set
    Update parameters: θ̃ ← θ + αv
    Compute gradient estimate:
    ĝ ← +∇_θ̃ L(f(x^(i); θ̃), y^(i))
    Compute the velocity update: v ← αv − εĝ
    Apply Update: θ ← θ + v
end while
```

- **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**

Adaptive Learning Rate Methods

Motivation

- Till now we assign the same learning rate to all features

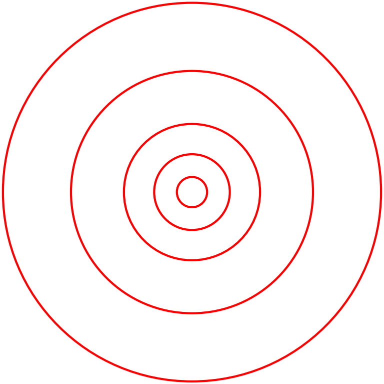
Motivation

- Till now we assign the same learning rate to all features
- If the features vary in importance and frequency, why is this a good idea?

Motivation

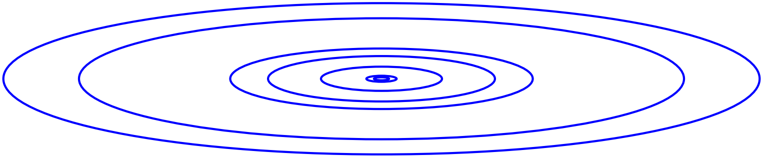
- Till now we assign the same learning rate to all features
- If the features vary in importance and frequency, why is this a good idea?
- It's probably not!

Motivation



Nice (all features are equally important)

Motivation



Harder!

AdaGrad

- Idea: Downscale a model parameter by square-root of sum of squares of all its historical values

AdaGrad

- Idea: Downscale a model parameter by square-root of sum of squares of all its historical values
- Parameters that have large partial derivative of the loss – learning rates for them are rapidly declined

AdaGrad

- Idea: Downscale a model parameter by square-root of sum of squares of all its historical values
- Parameters that have large partial derivative of the loss – learning rates for them are rapidly declined
- Some interesting theoretical properties

AdaGrad

Algorithm 4 AdaGrad

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

Initialize $\mathbf{r} = 0$

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

RMSProp

- AdaGrad is good when the objective is convex.

RMSProp

- AdaGrad is good when the objective is convex.
- AdaGrad might shrink the learning rate too aggressively, we want to keep the history in mind

RMSProp

- 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 non-convex settings by accumulating an exponentially decaying average of the gradient

RMSProp

- 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 non-convex settings by accumulating an exponentially decaying average of the gradient
- This is an idea that we use again and again in Neural Networks

RMSProp

- 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 non-convex settings by accumulating an exponentially decaying average of the gradient
- This is an idea that we use again and again in Neural Networks
- Currently has about 500 citations on scholar, but was proposed in a slide in Geoffrey Hinton's coursera course

RMSProp

Algorithm 5 RMSProp

Require: Global Learning rate ϵ, decay parameter ρ, δ

Initialize $\mathbf{r} = 0$

```
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 ← ρr + (1 − ρ)ĝ ⊙ ĝ
    Compute update: Δθ ← −ε/(δ+√r) ⊙ ĝ
    Apply Update: θ ← θ + Δθ
end while
```

while stopping criteria not met **do**

- Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
- Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
- Accumulate: $\mathbf{r} \leftarrow \rho\mathbf{r} + (1-\rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
- Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta+\sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
- Apply Update: $\theta \leftarrow \theta + \Delta\theta$

end while

RMSProp with Nesterov

Algorithm 6 RMSProp with Nesterov

Require: Global Learning rate ϵ, decay parameter ρ, δ, α, $\mathbf{v}$

Initialize $\mathbf{r} = 0$

while stopping criteria not met **do**

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

  Compute Update: $\tilde{\theta} \leftarrow \theta + \alpha \mathbf{v}$

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

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

  Compute Velocity: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \frac{\epsilon}{\sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$

  Apply Update: $\theta \leftarrow \theta + \mathbf{v}$

end while

Adam

- We could have used RMSProp with momentum

Adam

- We could have used RMSProp with momentum
- Use of Momentum with rescaling is not well motivated

Adam

- We could have used RMSProp with momentum
- Use of Momentum with rescaling is not well motivated
- Adam is like RMSProp with Momentum but with bias correction terms for the first and second moments

Adam: ADAptive Moments

Algorithm 7 RMSProp with Nesterov

Require: ϵ (set to 0.0001), decay rates ρ_1 (set to 0.9), ρ_2 (set to 0.9), θ, δ

Initialize moments variables $\mathbf{s} = 0$ and $\mathbf{r} = 0$, time step t = 0

while stopping criteria not met **do**

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

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

  $t \leftarrow t + 1$

  Update: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\hat{\mathbf{g}}$

  Update: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

  Correct Biases: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t}$

  Compute Update: $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$

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

end while

All your GRADs are belong to us!

SGD: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

Momentum: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \mathbf{v}$

Nesterov: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta + \alpha \mathbf{v}), \mathbf{y}^{(i)}) \right)$ then $\theta \leftarrow \theta + \mathbf{v}$

AdaGrad: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$ then $\Delta\theta - \leftarrow \dfrac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ then $\theta \leftarrow \theta + \Delta\theta$

RMSProp: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ then $\Delta\theta \leftarrow -\dfrac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \Delta\theta$

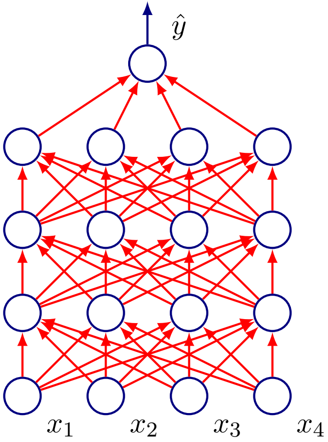
Adam: $\hat{\mathbf{s}} \leftarrow \dfrac{\mathbf{s}}{1 - \rho_1^t}, \hat{\mathbf{r}} \leftarrow \dfrac{\mathbf{r}}{1 - \rho_2^t}$ then $\Delta\theta = -\epsilon \dfrac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$ then $\theta \leftarrow \theta + \Delta\theta$

Batch Normalization

A Difficulty in Training Deep Neural Networks

A deep model involves composition of several functions

$$\hat{y} = W_4^T(\tanh(W_3^T(\tanh(W_2^T(\tanh(W_1^T\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3))))$$



A Difficulty in Training Deep Neural Networks

- We have a recipe to compute gradients (Backpropagation), and update every parameter (we saw half a dozen methods)

A Difficulty in Training Deep Neural Networks

- We have a recipe to compute gradients (Backpropagation), and update every parameter (we saw half a dozen methods)
- Implicit Assumption: Other layers don't change i.e. other functions are fixed

A Difficulty in Training Deep Neural Networks

- We have a recipe to compute gradients (Backpropagation), and update every parameter (we saw half a dozen methods)
- Implicit Assumption: Other layers don't change i.e. other functions are fixed
- In Practice: We update all layers simultaneously

A Difficulty in Training Deep Neural Networks

- We have a recipe to compute gradients (Backpropagation), and update every parameter (we saw half a dozen methods)
- Implicit Assumption: Other layers don't change i.e. other functions are fixed
- In Practice: We update all layers simultaneously
- This can give rise to unexpected difficulties

A Difficulty in Training Deep Neural Networks

- We have a recipe to compute gradients (Backpropagation), and update every parameter (we saw half a dozen methods)
- Implicit Assumption: Other layers don't change i.e. other functions are fixed
- In Practice: We update all layers simultaneously
- This can give rise to unexpected difficulties
- Let's look at two illustrations

Intuition

- Consider a second order approximation of our cost function (which is a function composition) around current point $\theta^{(0)}$:

$$J(\theta) \approx J(\theta^{(0)}) + (\theta - \theta^{(0)})^T \mathbf{g} + \frac{1}{2}(\theta - \theta^{(0)})^T H (\theta - \theta^{(0)})$$

Intuition

- Consider a second order approximation of our cost function (which is a function composition) around current point $\theta^{(0)}$:

$$J(\theta) \approx J(\theta^{(0)}) + (\theta - \theta^{(0)})^T \mathbf{g} + \frac{1}{2}(\theta - \theta^{(0)})^T H(\theta - \theta^{(0)})$$

- $\mathbf{g}$ is gradient and H the Hessian at $\theta^{(0)}$

Intuition

- Consider a second order approximation of our cost function (which is a function composition) around current point $\theta^{(0)}$:

$$J(\theta) \approx J(\theta^{(0)}) + (\theta - \theta^{(0)})^T \mathbf{g} + \frac{1}{2}(\theta - \theta^{(0)})^T H (\theta - \theta^{(0)})$$

- $\mathbf{g}$ is gradient and H the Hessian at $\theta^{(0)}$
- If ϵ is the learning rate, the new point

$$\theta = \theta^{(0)} - \epsilon \mathbf{g}$$

Intuition

- Plugging our new point, $\theta = \theta^{(0)} - \epsilon \mathbf{g}$ into the approximation:

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

Intuition

- Plugging our new point, $\theta = \theta^{(0)} - \epsilon \mathbf{g}$ into the approximation:

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

- There are three terms here:

Intuition

- Plugging our new point, $\theta = \theta^{(0)} - \epsilon \mathbf{g}$ into the approximation:

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

- There are three terms here:
  - Value of function before update

Intuition

- Plugging our new point, $\theta = \theta^{(0)} - \epsilon \mathbf{g}$ into the approximation:

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2}\mathbf{g}^T H \mathbf{g}$$

- There are three terms here:
  - Value of function before update
  - Improvement using gradient (i.e. first order information)

Intuition

- Plugging our new point, $\theta = \theta^{(0)} - \epsilon \mathbf{g}$ into the approximation:

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

- There are three terms here:
  - Value of function before update
  - Improvement using gradient (i.e. first order information)
  - Correction factor that accounts for the curvature of the function

Intuition

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

Intuition

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

- Observations:

Intuition

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2}\mathbf{g}^T H \mathbf{g}$$

- Observations:
  - $\mathbf{g}^T H \mathbf{g}$ too large: Gradient will start moving upwards

Intuition

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2}\mathbf{g}^T H \mathbf{g}$$

- Observations:
  - $\mathbf{g}^T H \mathbf{g}$ too large: Gradient will start moving upwards
  - $\mathbf{g}^T H \mathbf{g} = 0$: J will decrease for even large ϵ

Intuition

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

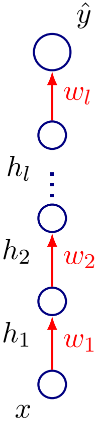
- Observations:
  - $\mathbf{g}^T H \mathbf{g}$ too large: Gradient will start moving upwards
  - $\mathbf{g}^T H \mathbf{g} = 0$: J will decrease for even large ϵ
  - Optimal step size $\epsilon^* = \mathbf{g}^T \mathbf{g}$ for zero curvature, $\epsilon^* = \frac{\mathbf{g}^T \mathbf{g}}{\mathbf{g}^T H \mathbf{g}}$ to take into account curvature

Intuition

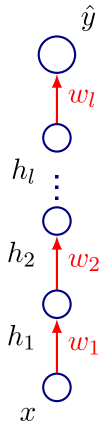
$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

- Observations:
  - $\mathbf{g}^T H \mathbf{g}$ too large: Gradient will start moving upwards
  - $\mathbf{g}^T H \mathbf{g} = 0$: J will decrease for even large ϵ
  - Optimal step size $\epsilon^* = \mathbf{g}^T \mathbf{g}$ for zero curvature, $\epsilon^* = \frac{\mathbf{g}^T \mathbf{g}}{\mathbf{g}^T H \mathbf{g}}$ to take into account curvature
- Conclusion: Just neglecting second order effects can cause problems (remedy: second order methods). What about higher order effects?

Higher Order Effects: Toy Model

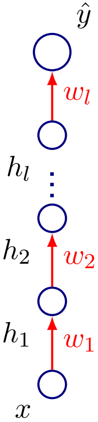


Higher Order Effects: Toy Model



- Just one node per layer, no non-linearity

Higher Order Effects: Toy Model



- Just one node per layer, no non-linearity
- $\hat{y}$ is linear in x but non-linear in w_i

Higher Order Effects: Toy Model

- Suppose $\delta = 1$, so we want to decrease our output $\hat{y}$

Higher Order Effects: Toy Model

- Suppose $\delta = 1$, so we want to decrease our output $\hat{y}$
- Usual strategy:
  - Using backprop find $\mathbf{g} = \nabla_{\mathbf{w}}(\hat{y} - y)^2$

Higher Order Effects: Toy Model

- Suppose $\delta = 1$, so we want to decrease our output $\hat{y}$
- Usual strategy:
  - Using backprop find $\mathbf{g} = \nabla_{\mathbf{w}}(\hat{y} - y)^2$
  - Update weights $\mathbf{w} := \mathbf{w} - \epsilon\mathbf{g}$

Higher Order Effects: Toy Model

- Suppose $\delta = 1$, so we want to decrease our output $\hat{y}$
- Usual strategy:
  - Using backprop find $\mathbf{g} = \nabla_{\mathbf{w}}(\hat{y} - y)^2$
  - Update weights $\mathbf{w} := \mathbf{w} - \epsilon \mathbf{g}$
- The first order Taylor approximation (in previous slide) says the cost will reduce by $\epsilon \mathbf{g}^T \mathbf{g}$

Higher Order Effects: Toy Model

- Suppose $\delta = 1$, so we want to decrease our output $\hat{y}$
- Usual strategy:
  - Using backprop find $\mathbf{g} = \nabla_{\mathbf{w}}(\hat{y} - y)^2$
  - Update weights $\mathbf{w} := \mathbf{w} - \epsilon\mathbf{g}$
- The first order Taylor approximation (in previous slide) says the cost will reduce by $\epsilon\mathbf{g}^T\mathbf{g}$
- If we need to reduce cost by 0.1, then learning rate should be $\frac{0.1}{\mathbf{g}^T\mathbf{g}}$

Higher Order Effects: Toy Model

- The new $\hat{y}$ will however be:

$$\hat{y} = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)\dots(w_l - \epsilon g_l)$$

Higher Order Effects: Toy Model

- The new $\hat{y}$ will however be:

$$\hat{y} = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)\dots(w_l - \epsilon g_l)$$

- Contains terms like $\epsilon^3 g_1 g_2 g_3 w_4 w_5 \dots w_l$

Higher Order Effects: Toy Model

- The new $\hat{y}$ will however be:

$$\hat{y} = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2) \dots (w_l - \epsilon g_l)$$

- Contains terms like $\epsilon^3 g_1 g_2 g_3 w_4 w_5 \dots w_l$
- If weights $w_4, w_5, \dots, w_l$ are small, the term is negligible. But if large, it would explode

Higher Order Effects: Toy Model

- The new $\hat{y}$ will however be:

$$\hat{y} = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)\dots(w_l - \epsilon g_l)$$

- Contains terms like $\epsilon^3 g_1 g_2 g_3 w_4 w_5 \dots w_l$
- If weights $w_4, w_5, \dots, w_l$ are small, the term is negligible. But if large, it would explode
- Conclusion: Higher order terms make it very hard to choose the right learning rate

Higher Order Effects: Toy Model

- The new $\hat{y}$ will however be:

$$\hat{y} = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)\dots(w_l - \epsilon g_l)$$

- Contains terms like $\epsilon^3 g_1 g_2 g_3 w_4 w_5 \dots w_l$
- If weights $w_4, w_5, \dots, w_l$ are small, the term is negligible. But if large, it would explode
- Conclusion: Higher order terms make it very hard to choose the right learning rate
- Second Order Methods are already expensive, nth order methods are hopeless. Solution?

Batch Normalization

- Method to reparameterize a deep network to reduce co-ordination of update across layers

Batch Normalization

- Method to reparameterize a deep network to reduce co-ordination of update across layers
- Can be applied to input layer, or any hidden layer

Batch Normalization

- Method to reparameterize a deep network to reduce co-ordination of update across layers
- Can be applied to input layer, or any hidden layer
- Let H be a design matrix having activations in any layer for m examples in the mini-batch

Batch Normalization

- Method to reparameterize a deep network to reduce co-ordination of update across layers
- Can be applied to input layer, or any hidden layer
- Let H be a design matrix having activations in any layer for m examples in the mini-batch

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1k} \\ h_{21} & h_{22} & h_{23} & \dots & h_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{m1} & h_{m2} & h_{m3} & \dots & h_{mk} \end{bmatrix}$$

Batch Normalization

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1k} \\ h_{21} & h_{22} & h_{23} & \dots & h_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \\ h_{m1} & h_{m2} & h_{m3} & \dots & h_{mk} \end{bmatrix}$$

- Each row represents all the activations in layer for one example

Batch Normalization

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1k} \\ h_{21} & h_{22} & h_{23} & \dots & h_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \\ h_{m1} & h_{m2} & h_{m3} & \dots & h_{mk} \end{bmatrix}$$

- Each row represents all the activations in layer for one example
- Idea: Replace H by H' such that:

$$H' = \frac{H - \mu}{\sigma}$$

Batch Normalization

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1k} \\ h_{21} & h_{22} & h_{23} & \dots & h_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{m1} & h_{m2} & h_{m3} & \dots & h_{mk} \end{bmatrix}$$

- Each row represents all the activations in layer for one example
- Idea: Replace H by H' such that:

$$H' = \frac{H - \mu}{\sigma}$$

- μ is mean of each unit and σ the standard deviation

Batch Normalization

- μ is a vector with μ_j the column mean

Batch Normalization

- μ is a vector with μ_j the column mean
- σ is a vector with σ_j the column standard deviation

Batch Normalization

- μ is a vector with μ_j the column mean
- σ is a vector with σ_j the column standard deviation
- $H_{i,j}$ is normalized by subtracting μ_j and dividing by σ_j

Batch Normalization

- During training we have:

$$\mu = \frac{1}{m} \sum_j H_{:,j}$$

Batch Normalization

- During training we have:

$$\mu = \frac{1}{m} \sum_j H_{:,j}$$

$$\sigma = \sqrt{\delta + \frac{1}{m} \sum_j (H - \mu)_j^2}$$

- We then operate on H' as before $\implies$ we backpropagate *through* the normalized activations

Why is this good?

- The update will never act to only increase the mean and standard deviation of any activation

Why is this good?

- The update will never act to only increase the mean and standard deviation of any activation
- Previous approaches added penalties to cost or per layer to encourage units to have standardized outputs

Why is this good?

- The update will never act to only increase the mean and standard deviation of any activation
- Previous approaches added penalties to cost or per layer to encourage units to have standardized outputs
- Batch normalization makes the reparameterization easier

Why is this good?

- The update will never act to only increase the mean and standard deviation of any activation
- Previous approaches added penalties to cost or per layer to encourage units to have standardized outputs
- Batch normalization makes the reparameterization easier
- At test time: Use running averages of μ and σ collected during training, use these for evaluating new input $\mathbf{x}$

An Innovation

- Standardizing the output of a unit can limit the expressive power of the neural network

An Innovation

- Standardizing the output of a unit can limit the expressive power of the neural network
- Solution: Instead of replacing H by H', replace it will $\gamma H' + \beta$

An Innovation

- Standardizing the output of a unit can limit the expressive power of the neural network
- Solution: Instead of replacing H by H', replace it will $\gamma H' + \beta$
- γ and β are also learned by backpropagation

An Innovation

- Standardizing the output of a unit can limit the expressive power of the neural network
- Solution: Instead of replacing H by H', replace it will $\gamma H' + \beta$
- γ and β are also learned by backpropagation
- Normalizing for mean and standard deviation was the goal of batch normalization, why add γ and β again?

Initialization Strategies

- In convex problems with good ϵ no matter what the initialization, convergence is guaranteed

- In convex problems with good ϵ no matter what the initialization, convergence is guaranteed
- In the non-convex regime initialization is much more important

- In convex problems with good ϵ no matter what the initialization, convergence is guaranteed
- In the non-convex regime initialization is much more important
- Some parameter initialization can be unstable, not converge

- In convex problems with good ϵ no matter what the initialization, convergence is guaranteed
- In the non-convex regime initialization is much more important
- Some parameter initialization can be unstable, not converge
- Neural Networks are not well understood to have principled, mathematically nice initialization strategies

- In convex problems with good ϵ no matter what the initialization, convergence is guaranteed
- In the non-convex regime initialization is much more important
- Some parameter initialization can be unstable, not converge
- Neural Networks are not well understood to have principled, mathematically nice initialization strategies
- What is known: Initialization should break symmetry (quiz!)

- In convex problems with good ϵ no matter what the initialization, convergence is guaranteed
- In the non-convex regime initialization is much more important
- Some parameter initialization can be unstable, not converge
- Neural Networks are not well understood to have principled, mathematically nice initialization strategies
- What is known: Initialization should break symmetry (quiz!)
- What is known: Scale of weights is important

- In convex problems with good ϵ no matter what the initialization, convergence is guaranteed
- In the non-convex regime initialization is much more important
- Some parameter initialization can be unstable, not converge
- Neural Networks are not well understood to have principled, mathematically nice initialization strategies
- What is known: Initialization should break symmetry (quiz!)
- What is known: Scale of weights is important
- Most initialization strategies are based on intuitions and heuristics

Some Heuristics

- For a fully connected layer with m inputs and n outputs, sample:

$$W_{ij} \sim U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$$

Some Heuristics

- For a fully connected layer with m inputs and n outputs, sample:

$$W_{ij} \sim U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$$

- Xavier Initialization: Sample

$$W_{ij} \sim U(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}})$$

Some Heuristics

- For a fully connected layer with m inputs and n outputs, sample:

$$W_{ij} \sim U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$$

- Xavier Initialization: Sample

$$W_{ij} \sim U(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}})$$

- Xavier initialization is derived considering that the network consists of matrix multiplications with no nonlinearites

Some Heuristics

- For a fully connected layer with m inputs and n outputs, sample:

$$W_{ij} \sim U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$$

- Xavier Initialization: Sample

$$W_{ij} \sim U(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}})$$

- Xavier initialization is derived considering that the network consists of matrix multiplications with no nonlinearites
- Works well in practice!

More Heuristics

- Saxe *et al.* 2013, recommend initialzing to random orthogonal matrices, with a carefully chosen gain g that accounts for non-linearities

More Heuristics

- Saxe *et al.* 2013, recommend initialzing to random orthogonal matrices, with a carefully chosen gain g that accounts for non-linearities
- If g could be divined, it could solve the vanishing and exploding gradients problem (more later)

More Heuristics

- Saxe *et al.* 2013, recommend initialzing to random orthogonal matrices, with a carefully chosen gain g that accounts for non-linearities
- If g could be divined, it could solve the vanishing and exploding gradients problem (more later)
- The idea of choosing g and initializing weights accordingly is that we want norm of activations to increase, and pass back strong gradients

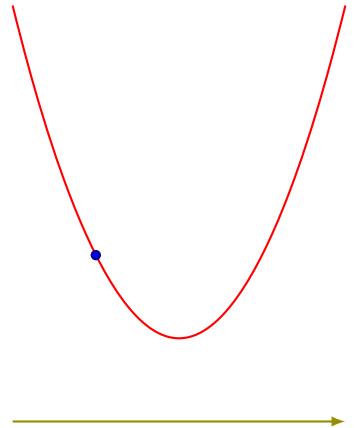
More Heuristics

- Saxe *et al.* 2013, recommend initialzing to random orthogonal matrices, with a carefully chosen gain g that accounts for non-linearities
- If g could be divined, it could solve the vanishing and exploding gradients problem (more later)
- The idea of choosing g and initializing weights accordingly is that we want norm of activations to increase, and pass back strong gradients
- Martens 2010, suggested an initialization that was sparse: Each unit could only receive k non-zero weights

More Heuristics

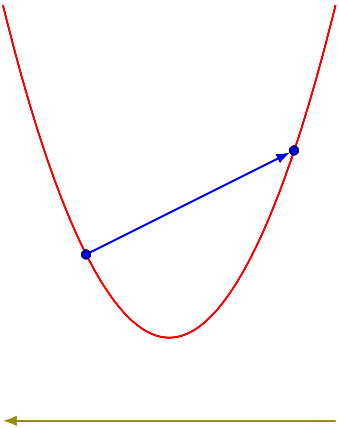
- Saxe *et al.* 2013, recommend initialzing to random orthogonal matrices, with a carefully chosen gain g that accounts for non-linearities
- If g could be divined, it could solve the vanishing and exploding gradients problem (more later)
- The idea of choosing g and initializing weights accordingly is that we want norm of activations to increase, and pass back strong gradients
- Martens 2010, suggested an initialization that was sparse: Each unit could only receive k non-zero weights
- Motivation: Ir is a bad idea to have all initial weights to have the same standard deviation $\frac{1}{\sqrt{m}}$

Polyak Averaging: Motivation

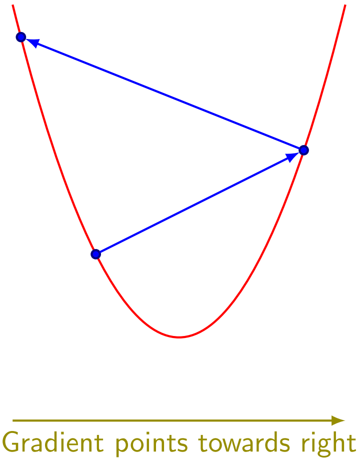


- Consider gradient descent above with high step size ϵ

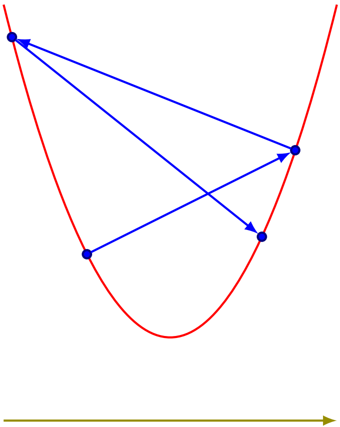
Polyak Averaging: Motivation



Polyak Averaging: Motivation

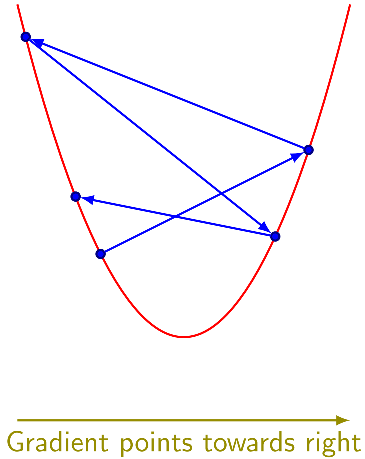


Polyak Averaging: Motivation



Gradient points towards left

Polyak Averaging: Motivation



A Solution: Polyak Averaging

- Suppose in t iterations you have parameters $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(t)}$

A Solution: Polyak Averaging

- Suppose in t iterations you have parameters $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(t)}$
- Polyak Averaging suggests setting $\hat{\theta}^{(t)} = \frac{1}{t}\sum_i \theta^{(i)}$

A Solution: Polyak Averaging

- Suppose in t iterations you have parameters $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(t)}$
- Polyak Averaging suggests setting $\hat{\theta}^{(t)} = \frac{1}{t}\sum_i \theta^{(i)}$
- Has strong convergence guarantees in convex settings

A Solution: Polyak Averaging

- Suppose in t iterations you have parameters $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(t)}$
- Polyak Averaging suggests setting $\hat{\theta}^{(t)} = \frac{1}{t} \sum_i \theta^{(i)}$
- Has strong convergence guarantees in convex settings
- Is this a good idea in non-convex problems?

Simple Modification

- In non-convex surfaces the parameter space can differ greatly in different regions

Simple Modification

- In non-convex surfaces the parameter space can differ greatly in different regions
- Averaging is not useful

Simple Modification

- In non-convex surfaces the parameter space can differ greatly in different regions
- Averaging is not useful
- Typical to consider the exponentially decaying average instead:

$$\hat{\theta}^{(t)} = \alpha\hat{\theta}^{(t-1)} + (1-\alpha)\hat{\theta}^{(t)} \text{ with } \alpha \in [0,1]$$

Next time

- Convolutional Neural Networks