Optimization

Optimization difficulties, Minibatch optimization, Momentum, Nesterov's Momentum, Parameter initialization, Algorithms (SGD, Adam, AdaGrad)

How learning is different from pure *optimization*?

While training the model

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y),$$

$\hat{p}_{data}$ distribution of training data

What we actually want

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim p_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y).$$

p_{data} distribution of actual data

Empirical risk minimization

$$\mathbb{E}_{\boldsymbol{x},\mathrm{y}\sim\hat{p}_{\text{data}}(\boldsymbol{x},y)}[L(f(\boldsymbol{x};\boldsymbol{\theta}), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$$

*Slide courtesy, Ian Goodfellow et al., deep learning book

Batch and Minibatch algorithms

Loss function

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y),$$

Training by backpropagation

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(f(x_i;\theta), y_i)$$

It requires you to evaluate gradients w.r.t all the training examples for gradient estimation

Is this efficient?

- Variance in the estimation with m samples - $\sigma/\sqrt{m}$
- By calculating grads over all samples, we get only sub-linear performance

*Slide courtesy, Ian Goodfellow et al., deep learning book

Batch and Minibatch algorithms

Loss function

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y),$$

Training by backpropagation

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(f(x_i;\theta), y_i)$$

By calculating grads over all samples, we get only **sub-linear** performance

What is the alternative?

- Simple solution, don’t use all the samples for gradient estimation
- At each update iteration, randomly chose B samples and use them for estimating gradients **Minibatch training**
- Also, does as unbiased estimate of gradients

$$\nabla_\theta J(\theta) = \frac{1}{B} \sum_{i=1}^{B} \nabla_\theta L(f(x_i;\theta), y_i)$$

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization

Stochastic Gradient Descent (SGD)

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

---

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
  end while

---

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization

Stochastic Gradient Descent (SGD) with momentum

Parameter update step of SGD

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$

- Depending on ϵ, learning can be very slow or have drastic oscillations
- Momentum is designed to accelerate SGD
- The momentum algorithm accumulates a weighted avg. of past gradients and continues to move in their direction.

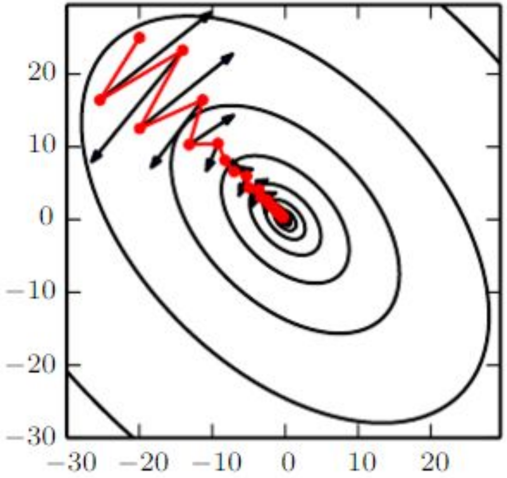


Figure showing effect of momentum
----- path with momentum
→ direction that SGD would take

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right),$$

Velocity $\boldsymbol{v}$ accumulates the past gradients

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

The larger α is relative to ϵ, the effect of past gradients is more

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization

Stochastic Gradient Descent (SGD) with momentum

Parameter update step now

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right),$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

- In SGD, update step size was ϵ ||g||
- With momentum, depends on how large and how aligned a *sequence* of gradients are
- Its largest, when successive gradients are same

If momentum repeatedly observes gradient as ***g***, it accelerates by a factor of $\frac{1}{1-\alpha}$, resulting in $\frac{\epsilon||\boldsymbol{g}||}{1-\alpha}.$

For α = 0.9, the descent is 10 times normal SGD

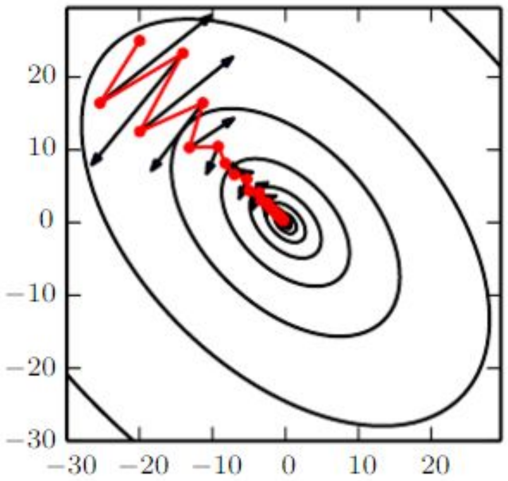


Figure showing effect of momentum
----- path with momentum
→ direction that SGD would take

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization

Stochastic Gradient Descent (SGD) with momentum

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

---

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $\boxed{\text{Compute velocity update: } \boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

---

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization

Nesterov momentum

Parameter update

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boxed{\boldsymbol{\theta} + \alpha \boldsymbol{v}}), \boldsymbol{y}^{(i)} \right) \right],$$

Look ahead

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization

Nesterov momentum

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

---

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
    Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$ ***Look ahead step***
    Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

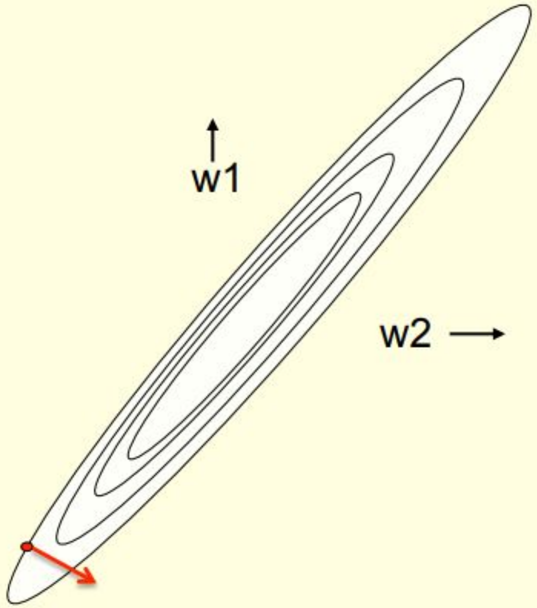
---

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization

Why learning can be slow

- If the ellipse is very elongated, the direction of steepest descent is almost perpendicular to the direction towards the minimum!
  - The red gradient vector has a large component along the short axis of the ellipse and a small component along the long axis of the ellipse.
  - This is just the opposite of what we want.



*Slide courtesy, G. Hinton's course on cousera week 3

Algorithms for optimization - adaptive learning rate

AdaGrad (Duchi et al., 2011)

Parameter update

Scales the learning rate with square root of sum of past gradients

– Larger partial derivatives - reduced learning rates (viceversa)

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Algorithms for optimization - adaptive learning rate

RMSProp(Hinton et al., 2012)

Parameter update

Scales the learning rate with weighted average of square of past gradients

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.

Initialize accumulation variables $\boldsymbol{r} = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

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

Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

Algorithms for optimization - adaptive learning rate

Adam (Kingma et al., 2014)

Parameter update

Combines RMSProp and momentum methods

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$

Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

end while

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization - approx. second order methods

Newton's method

Taylor expansion of J:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0),$$

$\boldsymbol{H}$ is the Hessian of J with respect to $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_0$

Solution for critical points:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

If higher order terms are included - iterate

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization - approx. second order methods

Newton's method

Algorithm 8.8 Newton's method with objective $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$.

Require: Initial parameter $\boldsymbol{\theta}_0$
Require: Training set of m examples
  while stopping criterion not met **do**
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian inverse: $\boldsymbol{H}^{-1}$
    Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization - approx. second order methods

Learning can be slow with steepest descent

Let d_{t-1} be the previous search direction

- At the minimum in direction d_{t-1},

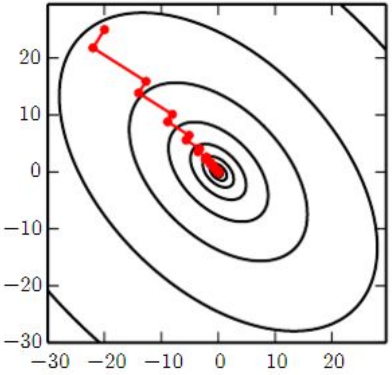
$$\nabla_\theta J(\theta).d_{t-1} = 0$$

- The current search direction of descent,

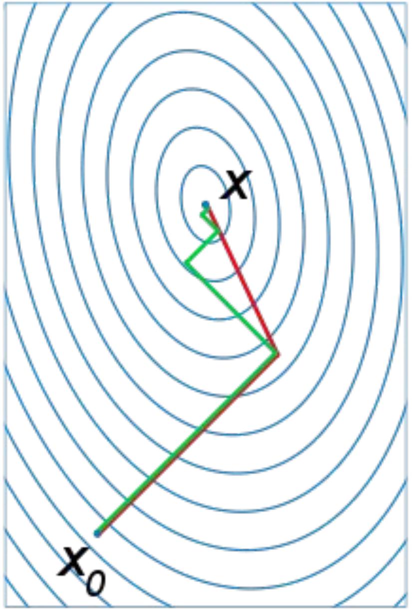
$$d_t = \nabla_\theta J(\theta)$$

d_t is *orthogonal* to previous d_{t-1}

- The current direction doesn't preserve minima along previous search direction



Steepest descent



Conjugate gradient
Steepest descent

Conjugate descent tries to address this

$$d_t = \nabla_\theta J(\theta) + \beta_t d_{t-1}$$

β_t controls previous search direction contribution

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization - approx. second order methods

Conjugate descent tries to address this

$$d_t = \nabla_\theta J(\theta) + \beta_t d_{t-1}$$

For conjugacy, $\boldsymbol{d}_t$ and $\boldsymbol{d}_{t-1}$

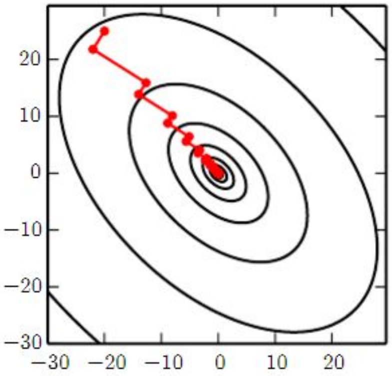
$$\boldsymbol{d}_t^\top \boldsymbol{H} \boldsymbol{d}_{t-1} = 0,$$

Without the need for $\boldsymbol{H}$, β_t can be evaluated using

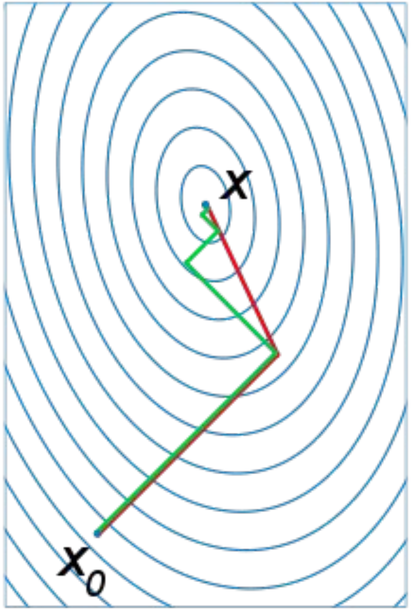
$$\beta_t = \frac{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$$

or

$$\beta_t = \frac{(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}))^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$$



Steepest descent



Conjugate gradient
Steepest descent

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization - approx. second order methods

Conjugate gradient

Algorithm 8.9 The conjugate gradient method

Require: Initial parameters $\boldsymbol{\theta}_0$
Require: Training set of m examples
  Initialize $\boldsymbol{\rho}_0 = \mathbf{0}$
  Initialize $g_0 = 0$
  Initialize $t = 1$
  while stopping criterion not met **do**
    Initialize the gradient $\boldsymbol{g}_t = \mathbf{0}$
    Compute gradient: $\boldsymbol{g}_t \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak-Ribière)
    (Nonlinear conjugate gradient: optionally reset β_t to zero, for example if t is a multiple of some constant k, such as $k = 5$)
    Compute search direction: $\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$
    Perform line search to find: $\epsilon^* = \operatorname{argmin}_\epsilon \frac{1}{m} \sum_{i=1}^m L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_t + \epsilon \boldsymbol{\rho}_t), \boldsymbol{y}^{(i)})$
    (On a truly quadratic cost function, analytically solve for ϵ^* rather than explicitly searching for it)
    Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$
    $t \leftarrow t + 1$
  end while

Slide courtesy, Ian Goodfellow et al., deep learning book

END

Algorithms for optimization - strategies

Batch Normalization (Ioffe et al., 2015)

Optimization challenges

Ill conditioning

Local Minima

Saddle points

*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization - Stochastic Gradient Descent

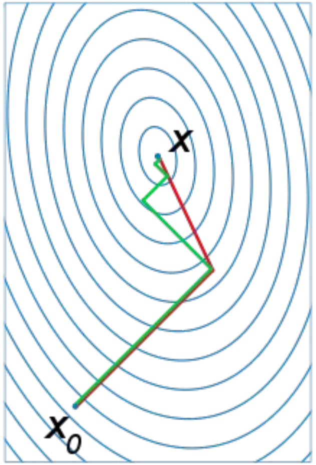
SGD; Momentum; Nesterov Momentum

Parameter initialization

Learning rate scheduling; AdaGrad, RMSProp, Adam

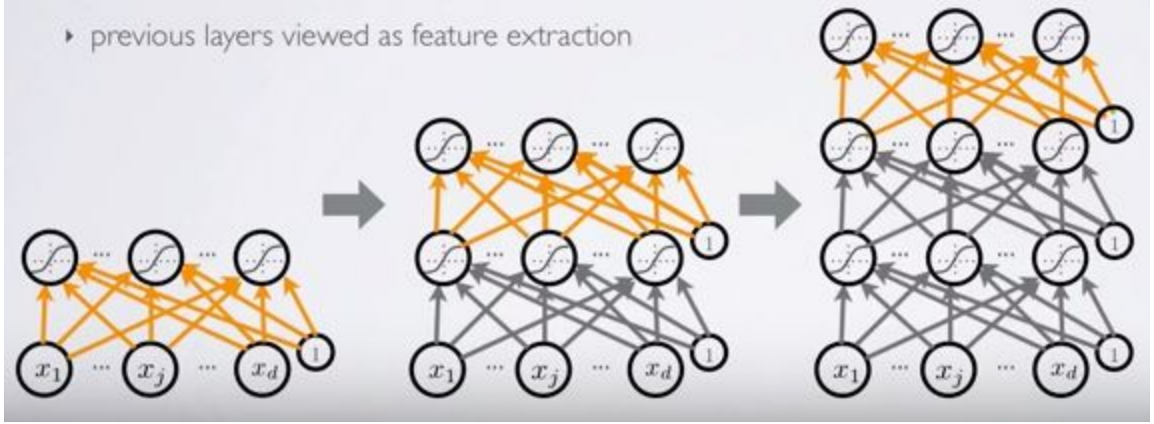
2nd order methods, Conjugate gradient

Batch Normalization



Unsupervised pretraining

Each layer is trained greedily fixing previous layers' weights



*Slide courtesy, Hugo larochelle