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

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

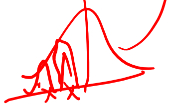
$\hat{p}_{data}$ distribution of training data

What we actually want

test dataset : p_{data}

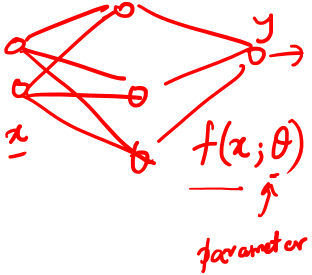
$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y).$$

p_{data} distribution of actual data



Empirical risk minimization

training dataset (x_i, y_i)



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

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

Batch and Minibatch algorithms

Loss function, weights, biases

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

Training by backpropagation

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

$$\overbrace{\min_{\theta} \frac{1}{m}\sum_{i=1}^{m} L(f(x^i;\theta), y^i)}^{J(\theta)}$$

$$\theta_{i+1} = \theta_i - \epsilon \nabla_\theta J(\theta)$$

step size | learning rate

$\nabla_\theta L$

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

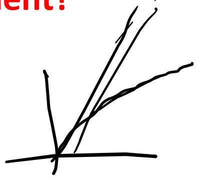
Is this efficient?

standard deviation

– ~~Variance~~ in the estimation with *m* samples - $\sigma/\sqrt{m}$

$\text{var} = \sigma^2/m$

$\text{std} = \sigma/\sqrt{m}$



– 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

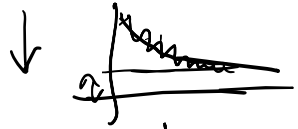
$$\nabla_\theta J(\theta) = \frac{1}{B} \sum_{i=1}^{B} \nabla_\theta L(f(x_i;\theta), y_i)$$

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

Algorithms for optimization

$\theta = \theta - \varepsilon \hat{g}$

$L(f(x;\theta), y)$



Stochastic Gradient Descent (SGD)

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

Require: Learning rate ϵ_k. ϵ

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.

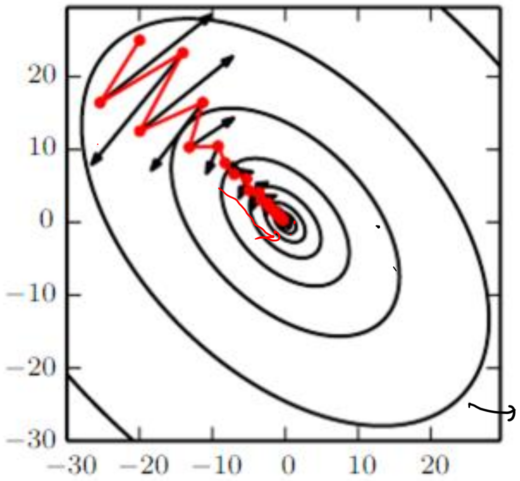


Figure showing effect of momentum
----- path with momentum
→ direction that SGD would take

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

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

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

$$\theta \leftarrow \theta + v.$$

$v \leftarrow \alpha v' - \varepsilon g$

$v_0 = 0$

$v_1 = \alpha v_0 - \varepsilon g_0$

$v_1 = -\varepsilon g_0$

$v_2 = \alpha v_1 - \varepsilon g_1$

$= -\alpha \varepsilon g_0 - \varepsilon g_1$

$v_3 = g_0, g_1, g_2$

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

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

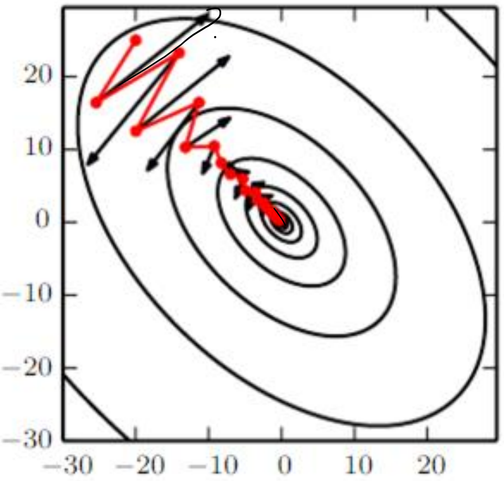


Figure showing effect of momentum
----- path with momentum
→ direction that SGD would take

$x + \alpha x + \alpha^2 x = \frac{x}{1-\alpha}$

*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

$v \leftarrow \alpha v - \epsilon g$

$g = \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^i; \theta), y^i) \right)$

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

Look ahead

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

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

---

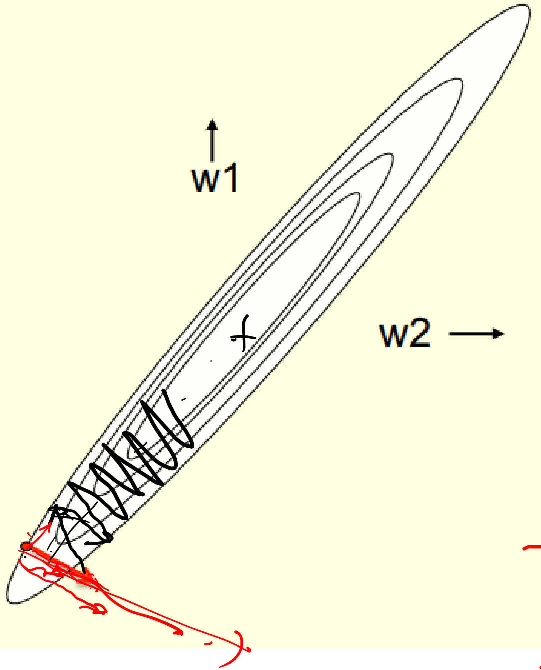
*Slide courtesy, Ian Goodfellow et al., deep learning book

Algorithms for optimization

$$f(x+h) = \boxed{f(x) + h f'(x)} + \frac{h^2}{2!} f''(x) + \dots$$

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)

$$g = \begin{pmatrix} g_1 \\ \vdots \\ g_n \end{pmatrix} \qquad g \odot g = \begin{pmatrix} g_1^2 \\ g_2^2 \\ \vdots \\ g_n^2 \end{pmatrix}$$

SGD $\rightarrow$ $\theta \leftarrow \theta + \Delta\theta$ $\quad \Delta\theta = -\epsilon g$

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 $\boldsymbol{r} = \mathbf{0}$
  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

$\rightarrow$ learning rate

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

$g = \frac{1}{N}\sum_{i=1}^{N} g_i$

$g_i = g + n_i \quad E[n_i] = 0 \quad E[\tilde{n}_i^2] = \sigma^2$

$\frac{1}{m}\sum_{i=1}^{m} g_i \sim g$

Unbiased est. $E\left[\frac{1}{m}\sum g_i\right] = \frac{1}{m}\sum E[g_i] = g$

$s_1 = (1-\rho_1)g_1$

$s_2 = \rho_1 s_1 + (1-\rho_1)g_2$

$= \rho_1(1-\rho_1)g_1 + (1-\rho_1)g_2$

$s_t = (1+\rho_1+\rho_1^2+\dots\rho^{t-1})(1-\rho_1)g$

$s_t \approx \frac{(1-\rho_1^t)}{(1-\rho_1)}(1-\rho_1)g$

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 $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$ → momentum

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

END