Lecture 6

Deep learning

Section 8.3, 8.5, 8.6.1

1

Chapter 8: Optimization for Training Deep Models

- One of the most important tasks in neural network training.
- It is very common to invest days to months of time on hundreds of machines in order to solve even a single instance of the neural network training problem.
- To address this problem a specialized set of optimization techniques have been developed for solving it.
- Optimization: finding the parameters θ of a neural network that significantly reduce a cost function $J(\theta)$,

2

Batch Gradient Descent (aka Vanilla Gradient Descend)

Algorithm: Batch Gradient Descent at iteration k

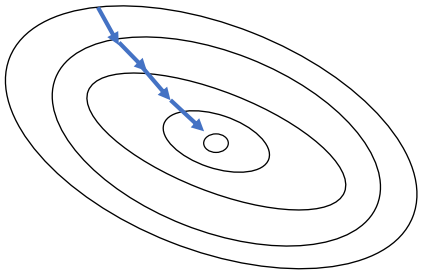
Require: Learning rate ϵ_k

Require: Initial Parameter θ

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

Goodfellow, Bengio, Courville 2016

3



Goodfellow, Bengio, Courville 2016

4

Stochastic Gradient Descent

Algorithm: Batch Gradient Descent at iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

1. **while** stopping criteria not met **do**
2. Sample example ($x^{(i)}$, $y^{(i)}$) from the training set
3. Compute gradient estimate :$\hat{g} \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$
4. Apply Update: $\theta \leftarrow \theta - \epsilon_k \hat{g}$
5. end while

Goodfellow, Bengio, Courville 2016

5

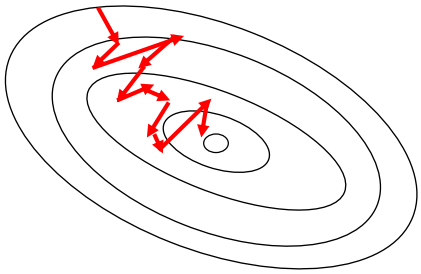
8.3.1 Mini-batch Gradient Descent

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

Goodfellow, Bengio, Courville 2016

6



Goodfellow, Bengio, Courville 2016

7

Learning rate

- In practice, it is necessary to gradually decrease the learning rate over time, so we now denote the learning rate at iteration *k* as ϵ_k.
- This is because the SGD gradient estimator introduces a source of noise (the random sampling of *m* training examples) that does not vanish even when we arrive at a minimum.
- it is common to decay the learning rate linearly until iteration τ:

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

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

Goodfellow, Bengio, Courville 2016

8

Learning rate

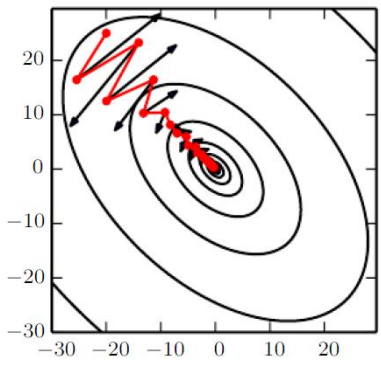
- The learning rate may be chosen by trial and error, but it is usually best to choose it by monitoring learning curves that plot the objective function as a function of time.
- Usually τ may be set to the number of iterations required to make a few hundred passes through the training set.
- $\epsilon_\tau$should be set to roughly 1% the value of ϵ_0.
- How to choose ϵ_0:
  - If it is too large, the learning curve will show violent oscillations, with the cost function often increasing significantly.
  - If the learning rate is too low, learning proceeds slowly, and if the initial learning rate is too low, learning may become stuck with a high cost value.

Goodfellow, Bengio, Courville 2016

9

Momentum

- Designed to accelerate learning, especially in the face of :
  - high curvature,
  - small but consistent gradients
  - noisy gradients.
- Accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.
- The momentum makes it keep going in the previous direction



10

Momentum

- Introduces a variable *v* that plays the role of velocity (typically initialized at zero)—it is the direction and speed at which the parameters move through parameter space.

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

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

$\alpha \in [0; 1)$

- The velocity accumulates the previous gradients. The velocity is set to an exponentially decaying average of the negative gradient.
- The larger α is relative to ϵ, the more previous gradients affect the current direction.

Goodfellow, Bengio, Courville 2016

11

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: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Goodfellow, Bengio, Courville 2016

12

Nesterov Momentum

- Presented by Ilya sutskever (2013) and it is inspired by Nesterov's accelerated gradient method (Nesterov, 1983, 2004)
- In most cases it works better than standard momentum
- The standard method of momentum first computes the gradient at the current location then makes a jump in the direction of updated accumulated gradient
- In Nestrov: first make a jump in the direction of previous accumulated gradient then make a correction

Goodfellow, Bengio, Courville 2016

13

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

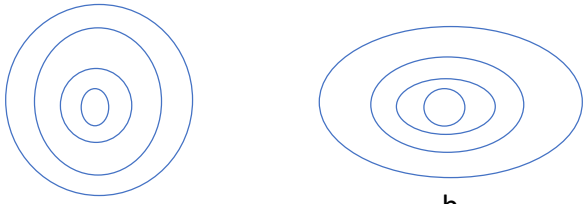
Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
  Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
  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

Goodfellow, Bengio, Courville 2016

14

Adaptive Learning Rate Methods

- Previous techniques use the same learning rates for all features
- Consider the following loss functions
- Shall we assign the same learning rates to all features in case b



Goodfellow, Bengio, Courville 2016

15

AdaGrad

- Proposed by Duchi *et al.*, 2011
- performs well for some but not all deep learning models.
- individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values
- The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives (infrequent update) have a relatively smaller decrease in their learning rate.
- The net effect is greater progress in the more gently sloped directions of parameter space.

Goodfellow, Bengio, Courville 2016

16

AdaGrad

- Enjoys some desirable theoretical properties.
- Empirically it has been found that—for training deep neural network models—the accumulation of squared gradients *from the beginning of training* can result in a premature and excessive decrease in the effective learning rate (stops the learning early).

Goodfellow, Bengio, Courville 2016

17

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

Note: Element-wise product or **Hadamard product**, and is denoted as A $\odot$ B- matrix containing the product of the individual elements

Goodfellow, Bengio, Courville 2016

18

RMSProp

- The **RMSProp** algorithm (Hinton, 2012) modifies AdaGrad to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average.
- Currently has about 2107 citations on scholar (proposed in a slide in Georey Hinton's coursera course)
- AdaGrad is designed to converge rapidly when applied to a convex function.
- When applied to a non-convex function to train a neural network, the learning trajectory may pass through many different structures and eventually arrive at a region that is a locally convex bowl. AdaGrad shrinks the learning rate according to the entire history of the squared gradient and may have made the learning rate too small before arriving at such a convex structure.
- RMSProp uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex bowl, as if it were an instance of the AdaGrad algorithm initialized within that bowl.

Goodfellow, Bengio, Courville 2016

19

RMSProp

- Empirically, RMSProp has been shown to be an effective and practical optimization algorithm for deep neural networks. It is currently one of the go-to optimization methods being employed routinely by deep learning practitioners.
- Compared to AdaGrad, the use of the moving average introduces a new hyperparameter, ρ, that controls the length scale of the moving average.

Goodfellow, Bengio, Courville 2016

20

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

Goodfellow, Bengio, Courville 2016

21

RMSProp with Nesterov

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
Initialize 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 interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Goodfellow, Bengio, Courville 2016

22

Adam

- Presented by Kingma and Ba, 2014
- The name "Adam" derives from the phrase "adaptive moments."
- In the context of the earlier algorithms, it is perhaps best seen as a variant on the combination of RMSProp and momentum with a few important distinctions.
- First, in Adam, momentum is incorporated directly as an estimate of the first order moment.
- Second, Adam includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin

Goodfellow, Bengio, Courville 2016

23

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 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 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

Goodfellow, Bengio, Courville 2016

24

Approximate Second-Order Methods - Newton's Method

- make use of second derivatives to improve optimization.
- The most widely used second-order method is Newton's method.
- the Newton parameter update rule:

$$\theta^* = \theta_0 - H^{-1}\nabla_\theta J(\theta_0)$$

- where *H* is the Hessian of *J (*square matrix of second-order partial derivatives of the function*)* with respect to θevaluated at θ_0
- Intuitively, second derivatives describes the local curvature of the loss function, which allows us to perform a more efficient update (take wider steps in case of shallow curvatures and smaller steps in case of steep curvatures)

Goodfellow, Bengio, Courville 2016

25

Newton's Method

- Absence of learning rate hyperparameters has a large advantage over first order methods.
- Computing (and inverting) the Hessian in its explicit form is a very costly process in both space and time.
- A large variety of *quasi-Newton* methods have been developed that seek to bring some of the advantages of Newton's method without the computational burden by approximating the calculation of the Hessian matrix. Among these, the most popular is L-BFGS.
- Applying L-BFGS on mini-batches is tricky an active area of research.

Goodfellow, Bengio, Courville 2016

26

Hyperparameter Optimization

- The most common hyperparameters in context of Neural Networks include:
  - the initial learning rate
  - learning rate decay schedule (such as the decay constant)
  - regularization strength (L2 penalty, dropout strength)
- Larger Neural Networks typically require a long time to train, so performing hyperparameter search can take many days/weeks.
- In most cases a single validation set of respectable size substantially simplifies the code base, without the need for cross-validation with multiple folds.

http://cs231n.github.io/neural-networks-3/#sgd

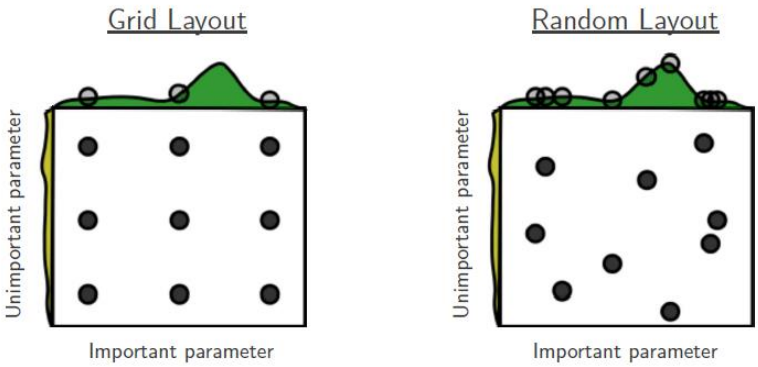
27

Hyperparameter Optimization

- learning rate and regularization strength have multiplicative effects on the training dynamics.
- Therefore, it is intuitive to search for these hyperparameters on log scale. For example, adding 0.01 to a learning rate has huge effects on the dynamics if the learning rate is 0.001, but nearly no effect if the learning rate when it is 10.
- Prefer random search to grid search: easier to implement, more effective, compared with pure grid search, random search finds better models using a fraction of the computing time [Ref. *Random Search for Hyper-Parameter Optimization by James Bergstra and Yoshua Bengio*]

http://cs231n.github.io/neural-networks-3/#sgd

28



Random Search for Hyper-Parameter Optimization by James Bergstra and Yoshua Bengio

29

Hyperparameter Optimization

- **Careful with best values on border**: If your best parameters are on border, you may be missing more optimal hyperparameter setting beyond the interval. Consider changing your range.
- **Stage your search from coarse to fine:** sometimes it is helpful to start your search in coarse ranges, then, based on the best results, narrow the range
- Algorithms to navigate the space of hyperparameters more efficiently is an active area of research. Examples of libraries that uses Bayesian Hyperparameter Optimization (Spearmint (https://github.com/JasperSnoek/spearmint),

http://cs231n.github.io/neural-networks-3/#sgd

30