Introduction to AI for postgraduate students

Lecture Note 8
Optimization for Training Deep Models

Agenda

- **How learning differs from pure optimization**
- Challenges in neural network optimization
- Basic algorithms
- Parameter initialization strategies
- Algorithms with adaptive learning rates

How Learning Differs from Pure Optimization

- ML usually acts indirectly.
- In classical optimization, we optimize the performance measure P.
- In ML, P is often intractable, and is defined with respect to unobserved test sets.
- In ML, we reduce a different cost function $J(\boldsymbol{\theta})$ in the hope that doing so will improve P.

- The focus is on the unregularized and supervised learning case, where J is typically determined:

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

- However, the ultimate goal is to minimize the generalization error, which is defined by:

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

Empirical Risk Minimization

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

Empirical risk

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

Risk

- The training process minimizing an empirical risk is very similar to straightforward optimization.
- But often, we use different approach in minimizing the empirical risk with ML.
  - Early stopping, regularization, etc.
  - Cf) In classical optimization, the optimization ends typically when the gradient is very small.
- It's because, the ultimate goal of ML is to minimize the risk (generalization error), not the empirical risk.

Surrogate Loss Function

- Sometimes the loss function we actually care about is not one that can be optimized efficiently.
  - Exactly minimizing 0-1 loss is typically intractable.
- Surrogate loss function
  - Acts as a proxy of the original goal but has advantages.
  - E.g., negative log-likelihood is used as a surrogate for the 0-1 loss.
  - We can improve the robustness of the classifier by further with a surrogate loss by pushing the classes apart from each other

Batch and Minibatch Algorithms

- One major difference from the classical optimization in ML is that we generally use a minibatch algorithm.
  - The full batch provides a more accurate estimation of the gradient, but it is not the best for ML training.

- For instance, in maximum likelihood problems, our training goal is to minimize

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}, y; \boldsymbol{\theta})$$

  who has the gradient as $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} \log p_{\text{model}}(\boldsymbol{x}, y; \boldsymbol{\theta})$

- We typically employ the sample mean theory to find the solution on $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

Batch and Minibatch Algorithms

- Why we have to use a minibatch?
  - Full batch may be computationally prohibited.
  - Generalization error may be suppressed by using minibatches, because of the increased noise in the gradient estimation.
- Considerations needed to determine the batch size
  - Larger batches provide a more accurate estimate of the gradient, but with less than linear returns.
  - Multicore architectures are usually underutilized by extremely small batches.
  - If all examples in the batch are to be processed in parallel (as is typically the case), then the amount of memory scales with the batch size.
  - When using GPUs, it is common for power of 2 batch sizes to offer better runtime. Typical power of 2 batch sizes range from 32 to 256, with 16 sometimes being attempted for large models.
  - Small batches can offer a regularizing effect, perhaps due to the noise they add to the learning process.
  - Training with such a small batch size might require a small learning rate to maintain stability because of the high variance in the estimate of the gradient.
  - Minibatches should be selected randomly, and they should be independent as much as possible.

Agenda

- How learning differs from pure optimization
- **Challenges in neural network optimization**
- Basic algorithms
- Parameter initialization strategies
- Algorithms with adaptive learning rates

Challenge: Ill -Conditioning

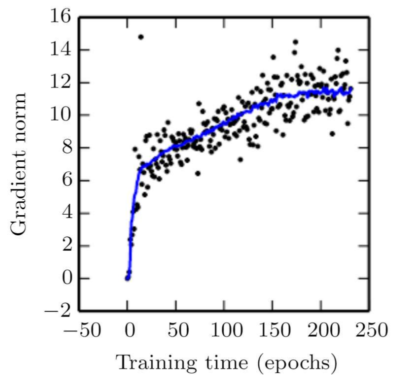
- In Chapter 4 we learnt the followings:
  - The Taylor approximation of $f(\boldsymbol{x})$ around the current point $\boldsymbol{x}^{(0)}$:

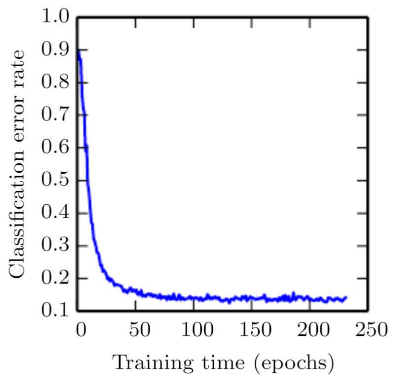
$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)})$$

  - The gradient descent update gives us $\boldsymbol{x} = \boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}$, at which the cost function becomes

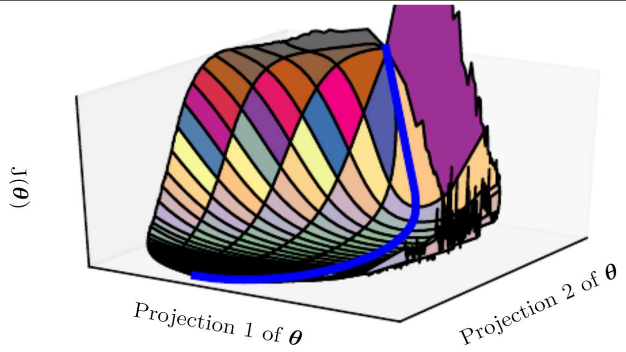
$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$$

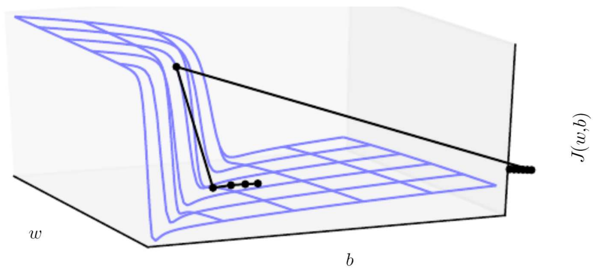
- We expect $f(\boldsymbol{x}^{(0)}) > f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g})$, so there will be a problem if $\frac{1}{2}\epsilon^2 \boldsymbol{g}^\mathrm{T} \boldsymbol{H} \boldsymbol{g} > \epsilon \boldsymbol{g}^\mathrm{T} \boldsymbol{g}$.
- In many cases, $\boldsymbol{g}^\mathrm{T} \boldsymbol{H} \boldsymbol{g}$ grows easily.
  - Learning becomes very slow despite the presence of a strong gradient.





Challenges: Others





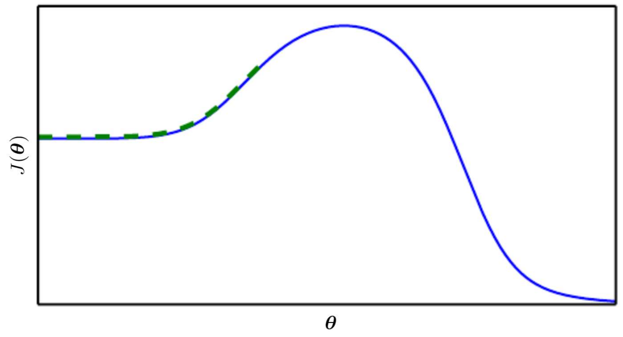
- Zero gradient at non-optimal points
  - Local minima
  - Saddle points and other flat regions
  - SGD can be a solution
- Cliffs and Exploding Gradients
  - Gradient clipping can be a solution
- Vanishing and exploding gradient problem
  - Some neural networks (e.g., RNN) contain a path consisting of repeatedly multiplying by a matrix $\boldsymbol{W}$.
  - With an eigen decomposition notation $\boldsymbol{W} = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$, after t steps, the total multiplying factor becomes:

$$\boldsymbol{W}^t = \left(\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}$$

  - If eigen values are smaller than 1 in magnitude → vanishing gradient problem
  - If eigen values are larger than 1 in magnitude → exploding gradient problem → gradient clipping.

Challenges: Others (Cont'd)

- Inexact gradients
  - Use a surrogate loss function that is easier to approximate than the true loss
- Poor initialization
  - In many cases, neural networks do not arrive at any critical points such as local minima, saddle points, etc.
  - But still, due to poor initialization, the learning result can be poor.
- Theoretical limits of optimization theory
  - e.g., discrete-valued units

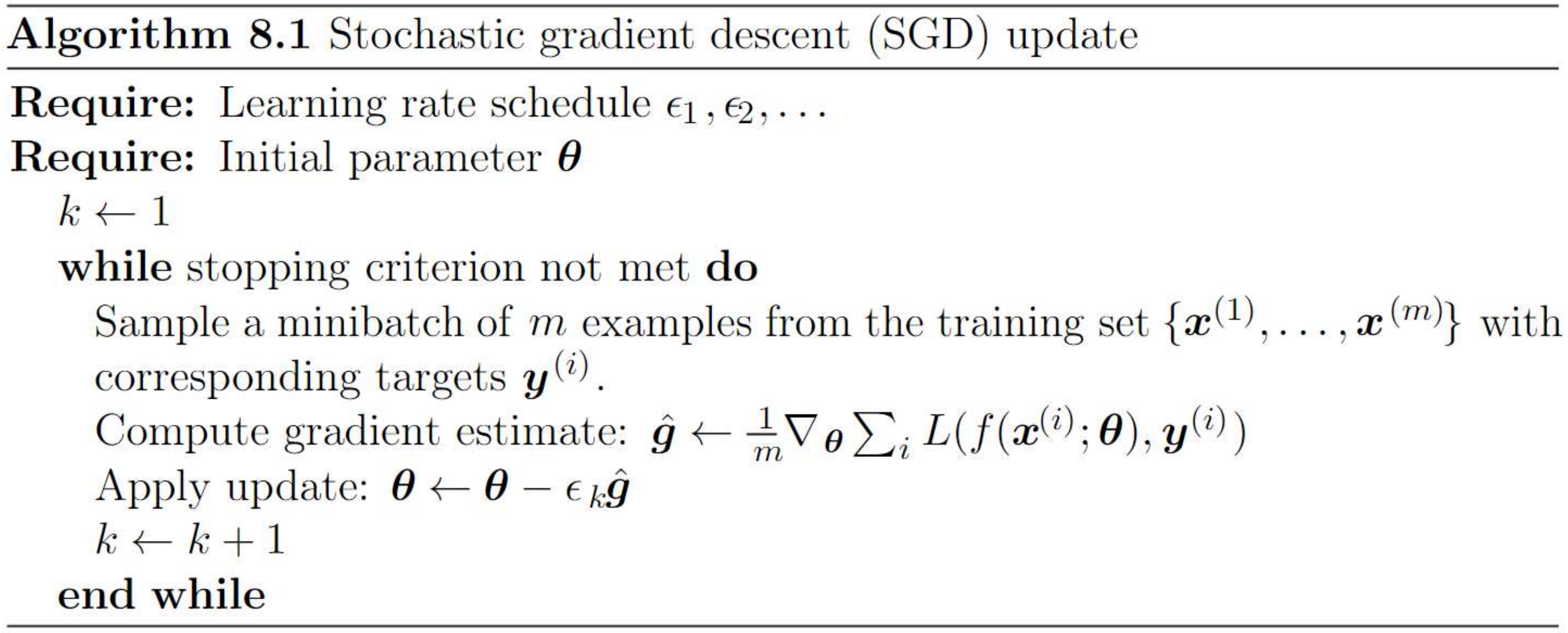


Agenda

- How learning differs from pure optimization
- Challenges in neural network optimization
- **Basic algorithms**
- Parameter initialization strategies
- Algorithms with adaptive learning rates

Stochastic Gradient Descent

- SGD and its variants are the most used optimization algorithms for deep learning.
- It's possible to obtain an unbiased estimate of the gradient by taking the average gradient on a minibatch of m examples drawn i.i.d. from the data-generating distribution.



Algorithm 8.1 Stochastic gradient descent (SGD) update

```
Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  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: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while
```

Stochastic Gradient Descent

- Scheduling the learning rate is very important.
  - We gradually decrease the learning rate over time.
  - This is because the SGD gradient estimator introduces a source of noise that does not vanish even when we arrive at a minimum.
- Sufficient condition to guarantee convergence of SGD:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \quad \text{and}$$

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

- Typical choice of ϵ_k is:

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

$\alpha = \frac{k}{\tau}$, where τ is set to the number of iterations required to make a few hundred passes through the training set.

ϵ_τ should be set to roughly 1 % the value of ϵ_0.

If it is too large, the learning curve will show violent oscillations.
If the learning rate is too low, learning proceeds slowly, and learning may become stuck with a high cost value.

Momentum

- Drawback of the SGD: learning can be slow.
- Momentum
  - Accelerates learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients.
  - Momentum in physics = mass × velocity.
  - Assuming unit mass, the momentum is the velocity here, which is defined by

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

- α: hyperparameter
- Tries to maintain the previous velocity

- Incremental change in velocity

  - Then, the parameter update is given by

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

SGD with Momentum

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$
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 $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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
```

SGD with Momentum

- SGD with momentum can mitigate abrupt change in gradients
  - Fast and reliable learning in the face of high curvature.
- SGD with momentum can also accelerate the update when the gradients are all lying in the same direction.
  - If the gradient's direction is always in the same direction, the accumulation of the velocity gives us the converged velocity $\boldsymbol{v}$ where

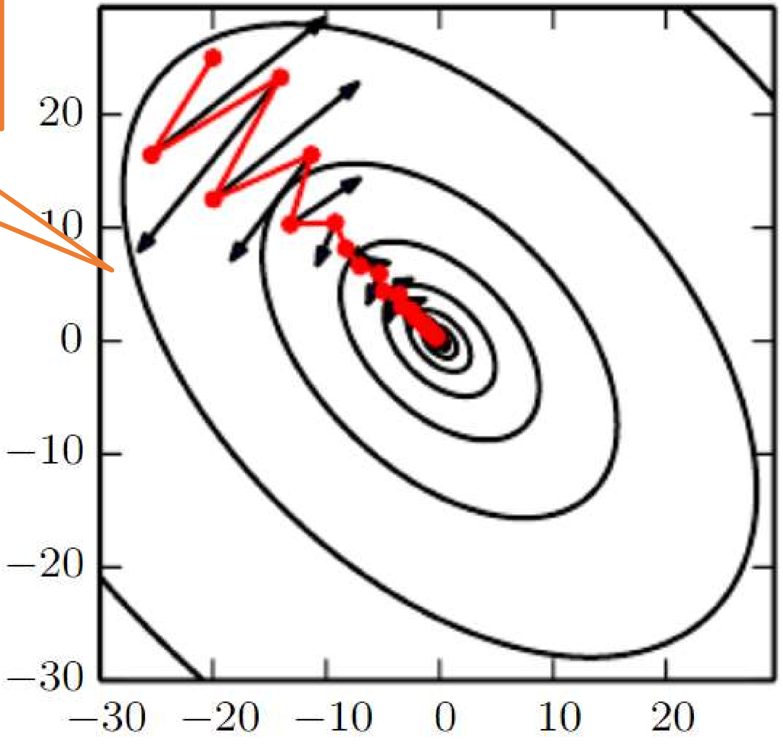
$$\boldsymbol{v} = \alpha \boldsymbol{v} - \epsilon \boldsymbol{g} \rightarrow \boldsymbol{v} = -\frac{\epsilon \boldsymbol{g}}{1-\alpha}, \quad \boldsymbol{g}\left(= \nabla_{\boldsymbol{\theta}}\left(\frac{1}{m}\sum_{i=1}^{m} L\left(\boldsymbol{f}\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right),\boldsymbol{y}^{(i)}\right)\right)\right)$$

  - So, the magnitude of $\boldsymbol{v}$ becomes $\frac{\epsilon\|\boldsymbol{g}\|}{1-\alpha}$.
  - Typical choice is $\alpha = \{0.5, 0.9, 0.99\}$.
  - With $\alpha = 0.9$, we get $\frac{1}{1-\alpha} = 10$, so 10 times faster update is possible compared to the original SGD.

SGD with Momentum

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

The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix



Agenda

- How learning differs from pure optimization
- Challenges in neural network optimization
- Basic algorithms
- **Parameter initialization strategies**
- Algorithms with adaptive learning rates

Parameter Initialization Strategies

- Deep learning training algorithms usually are iterative and thus require the user to specify some initial point.
- Training deep models is strongly affected by the choice of initializations.
  - Initial points determine whether the algorithm converges at all.
  - Initial points determine how quickly learning converges, and whether it converges to a point with high or low cost.
  - Initial points affect the generalization error.
- Modern initialization strategies are simple and heuristic.
  - Designing improved initialization strategies is a difficult task, because neural network optimization is not yet well understood.
  - Although some initial points are good in the optimization perspectives, but detrimental from the viewpoint of generalization error.

Parameter Initialization Strategies

- Break symmetry
  - Initialize each unit to compute a different function from all the other units.
  - Gram-Schmidt orthogonalization can be used to find a symmetry-breaking initial weight matrix.
  - But random initialization works well even with low computational complexity.
- Choosing the magnitudes of initial points
  - Large initial weights are good because
    - ✓ We can expect a stronger symmetry-breaking effect
    - ✓ We can avoid losing signal during forward or back-propagation.
  - Too large initial weights are not good because
    - ✓ There can be exploding gradients problems.
    - ✓ They result in saturation after the activation function.
- Popular choice: sampling each unit from

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right) \quad \textbf{OR} \quad U\left(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right)$$

  - where m: # of input units, n: # of output units

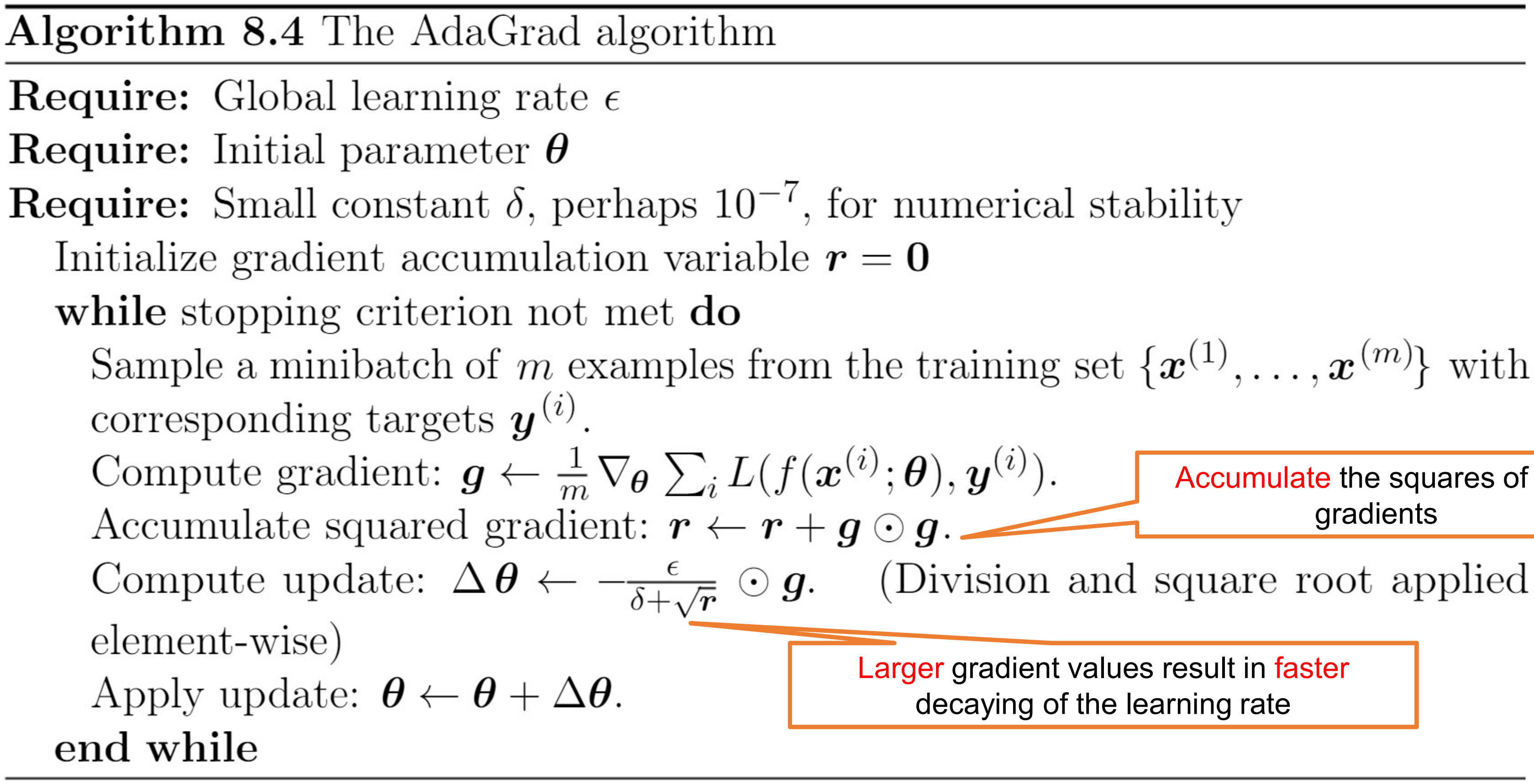
Agenda

- How learning differs from pure optimization
- Challenges in neural network optimization
- Basic algorithms
- Parameter initialization strategies
- **Algorithms with adaptive learning rates**

Algorithms with Adaptive Learning Rates

- Learning rate is very important to train deep models robustly and reliably.
- The cost is often highly sensitive to some directions in parameter space and insensitive to others.
  - Having the same learning rate for all parameters is not good.
- The momentum algorithm can mitigate these issues to some extent.
- Are there any other approaches?
  - Idea is to separate learning rate for each parameter and automatically adapt these learning rates throughout the course of learning.

AdaGrad



```
Algorithm 8.4 The AdaGrad algorithm
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, 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
```

Accumulate the squares of the gradients

Larger gradient values result in faster decaying of the learning rate

RMSProp

Algorithm 8.5 The RMSProp algorithm

```
Require: Global learning rate $\epsilon$, decay rate $\rho$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, 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
```

Moving average instead of accumulation, which can avoid too fast decaying of the learning rate.

Adam

Algorithm 8.7 The Adam algorithm

```
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$.
  (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ 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
```

Momentum method

Moving average of the squared gradients

Choosing the Right Optimization Algorithm

- There is currently no consensus on "which algorithm should be the best for given problem and dataset".
- The most popular optimization algorithms actively in use include SGD, SGD with momentum, RMSProp, RMSProp with momentum, AdaDelta, and Adam.