Large Scale Data Analysis Using Deep Learning

Optimization for Training Deep Models

U Kang
Seoul National University

In This Lecture

- Challenges in neural network optimization
- Basic learning algorithms
- Algorithms with adaptive learning rates

How Learning Differs from Pure Optimization

- Optimization for deep models differ from traditional optimization algorithms
  - For deep model: optimize indirectly
    - We care about some performance measure P defined with respect to the test set
    - We minimize a different cost function $J(\theta)$ in the hope that doing so will improve P
  - Pure optimization: minimizing J is a goal in and of itself

How Learning Differs from Pure Optimization

- In optimization for deep models, our final goal is to minimize loss with regard to data generating distribution p_{data}
  - $E_{x,y\sim p_{data}} L(f(x,\theta),y)$
- Since we do not know p_{data}, we use empirical risk minimization
  - $E_{x,y\sim \hat{p}_{data}} L(f(x,\theta),y) = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)},\theta),y^{(i)})$
- However, optimization for deep model does not perform direct empirical risk minimization due to overfitting and often non-differentiable loss function
  - Instead, optimization for deep model uses early stopping and surrogate loss function (e.g. cross entropy)

Challenges in Neural Network Optimization

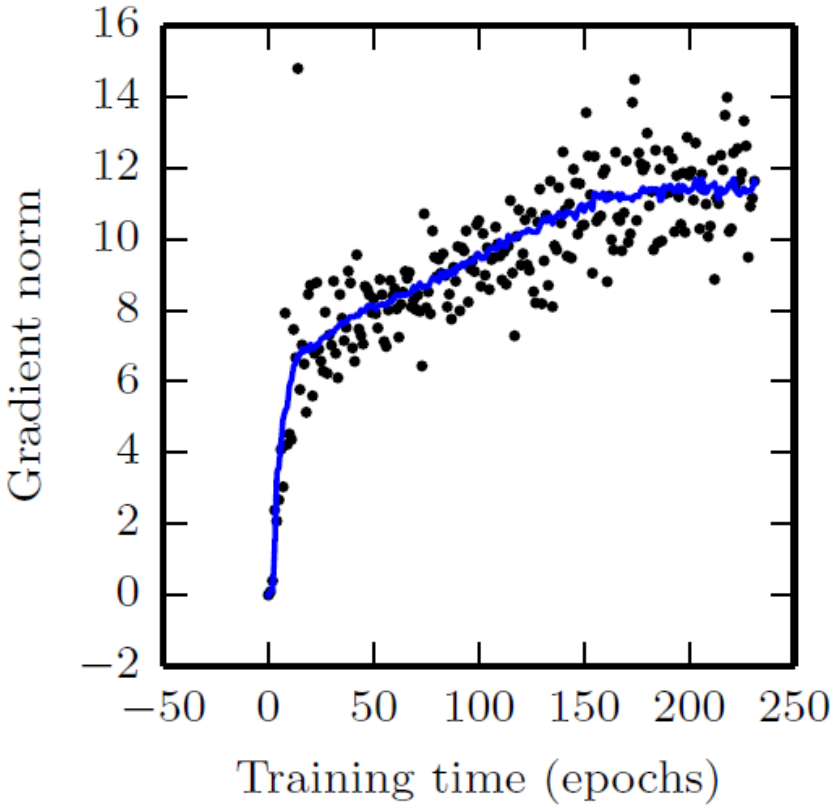
- Ill-Conditioning
- Local minima
- Cliffs and exploding gradients
- Long-term dependencies
- Poor correspondence between local and global structure

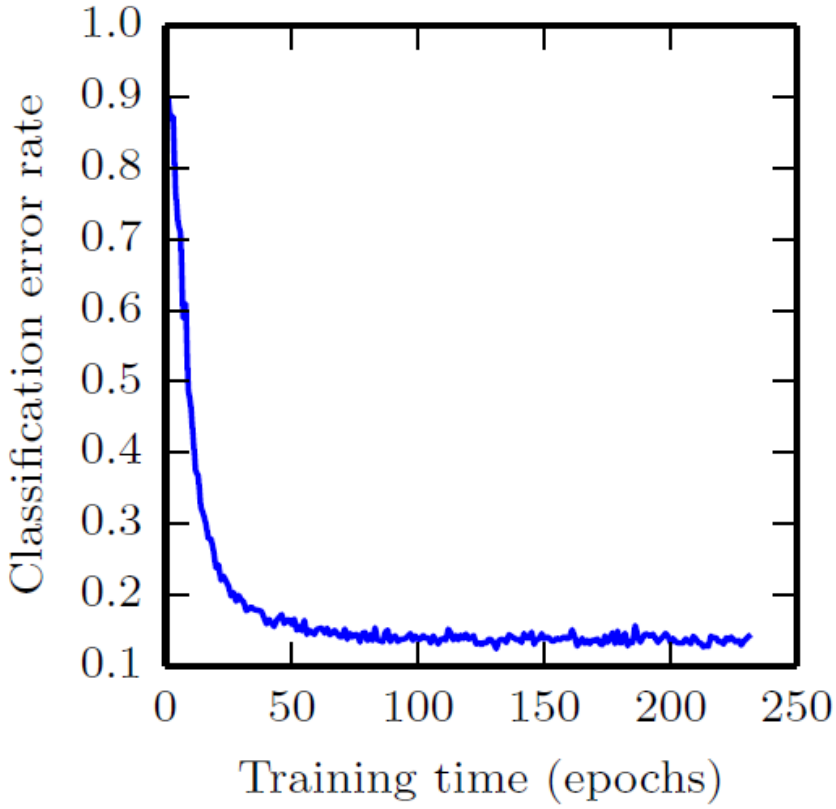
Ill-Conditioning

- Gradient descent procedure
  - $f(x) \approx f\left(x^{(0)}\right) + \left(x - x^{(0)}\right)^T g + \frac{1}{2}\left(x - x^{(0)}\right)^T H\left(x - x^{(0)}\right)$ where g is the gradient and H is the Hessian at $x^{(0)}$.
  - Using a learning rate of ϵ, then the new point x will be given by $x^{(0)} - \epsilon$g.
  - Then, $f\left(x^{(0)} - \epsilon \mathrm{g}\right) \approx f\left(x^{(0)}\right) - \epsilon g^T g + \frac{1}{2}\epsilon^2 g^T H g$
- Ill-conditioning happens when $\frac{1}{2}\epsilon^2 g^T H g$ exceeds $\epsilon g^T g$

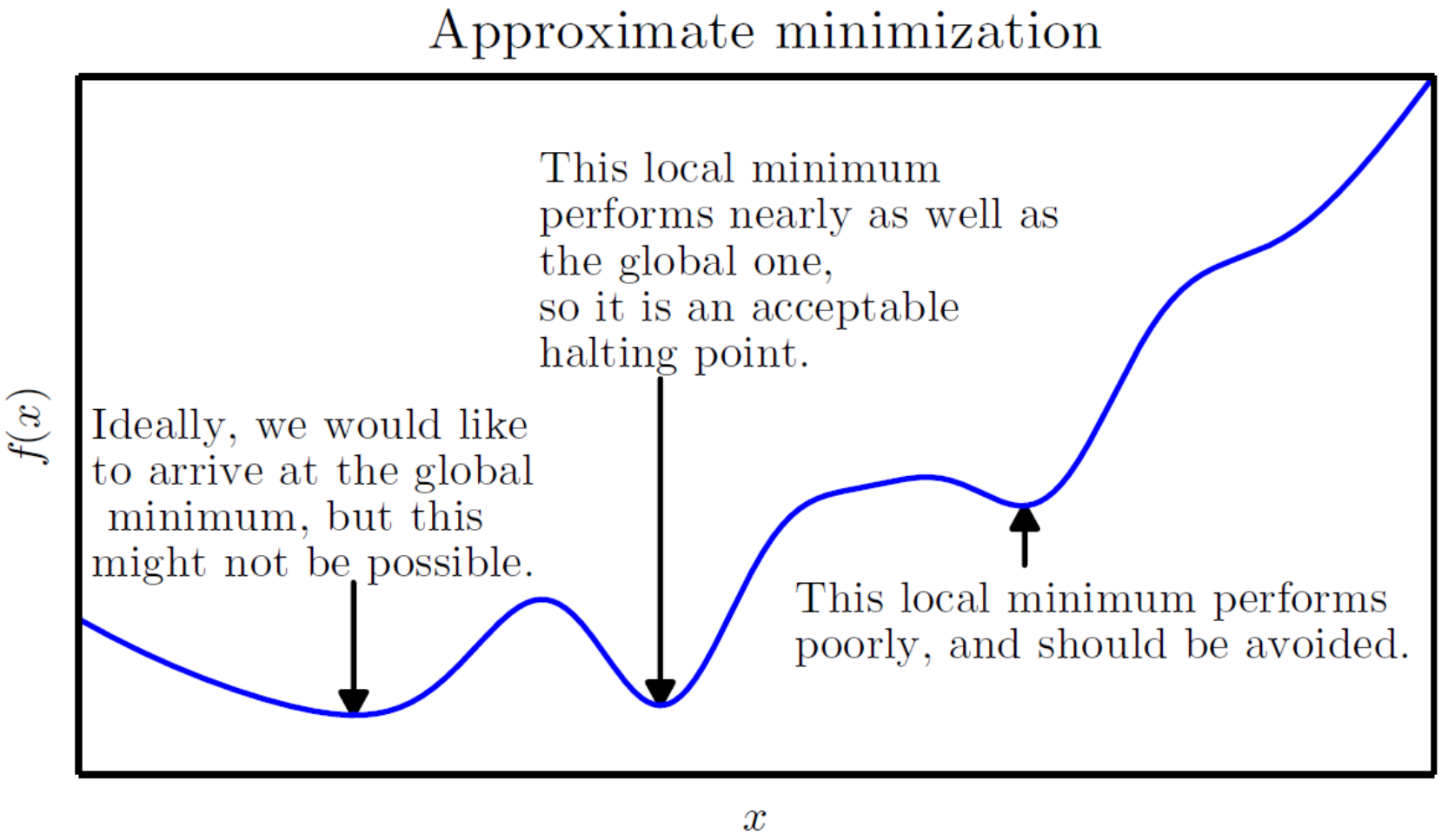
No Critical Point

- Gradient may be large even if the error converged





Local Minima

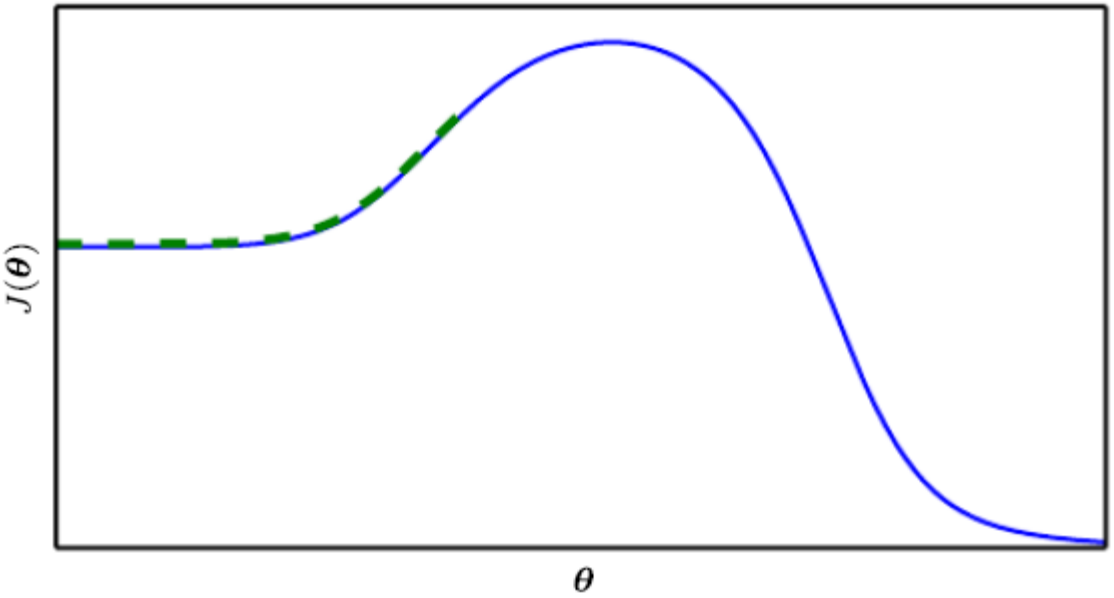


Local Minima

- Neural networks have multiple local minima, because of model identifiability problem
  - A model is *identifiable* if a sufficiently large training set can rule out all but one setting of the model's parameters
  - Models with latent variables are often not identifiable because we can obtain equivalent models by exchanging latent variables with each other
  - Neural network: one can make many equivalent models by swapping weights of node i with those of node j. This kind of non-identifiability is called *weight space symmetry*
- Optimizing in the presence of local minima is an active area of research
  - However, experts suspect that for sufficiently large neural networks, most local minima have a sufficiently low cost function value, and finding a true global minimum is not that important

Poor Correspondence between Local and Global Structure

- Optimization based on local downhill moves can fail if the local surface does not point toward the global solution



Basic Algorithms

- Stochastic Gradient Descent (SGD)
- Momentum
- Nesterov momentum

Stochastic Gradient Descent (SGD)

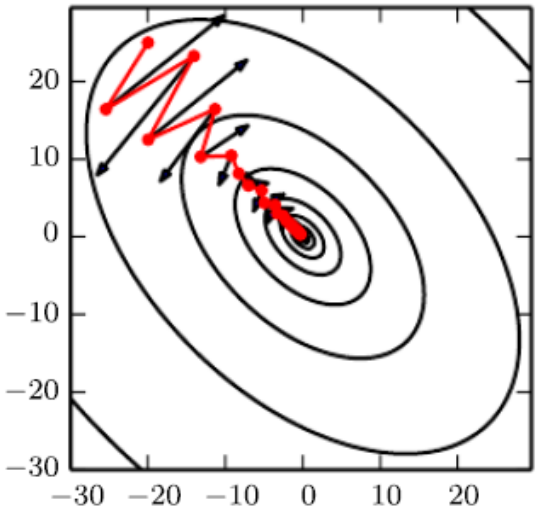
- Obtain an unbiased estimate of the gradient by taking the average gradient on a minibatch of m examples drawn i.i.d from the data generating distribution

Algorithm 8.1 Stochastic gradient descent (SGD) update 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

Momentum

- Learning with SGD can sometimes be slow
- The method of momentum is designed to solve poor conditioning of the Hessian matrix and variance in SGD
- The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction



Red line: from momentum
Black arrow: from SGD

Momentum

- Update rule in momentum
  - $v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m}\sum_{i=1}^{m} L\left(f\left(x^{(i)}, \theta\right), y^{(i)}\right)\right)$
  - $\theta \leftarrow \theta + v$
- Common values of α: 0.5, 0.9, and 0.99

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

Nesterov Momentum

- Update rule is the same as that of momentum
  - $v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m}\sum_{i=1}^{m} L\left(f\left(x^{(i)}, \theta\right), y^{(i)}\right)\right)$
  - $\theta \leftarrow \theta + v$
- Difference is that with Nesterov momentum the gradient is evaluated after the current velocity is applied

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

Algorithms with Adaptive Learning Rates

- Learning rate is one of the hyperparameters that is the most difficult to set since it has a significant impact on model performance
- The momentum algorithm can mitigate the issue, but at the expense of introducing another hyperparameter
- Methods with adaptive learning rates for model parameters
  - AdaGrad
  - RMSProp
  - Adam (variant of RMSProp + momentum)

AdaGrad

- 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

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

AdaGrad

- 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
- Puts more weights on rare features
  - E.g., in document classification, rare terms would become more important features than common terms do
    - Consider a linear model (e.g., logistic regression) for document classification
    - Assume a rare term y does not appear in a document D
    - Gradient vector's element for w_y(weight for term y) would be 0 for loss wrt D (i.e., changing w_y does not change classification of D) since y is not in D
    - Thus, AdaGrad would accumulate small values for rare terms (since their gradients would be 0 if they do not appear in documents), but large values for frequent terms
    - AdaGrad gives larger weights for gradients of rare terms, and smaller weights for those of common terms

AdaGrad

- For convex optimization, AdaGrad works well with desirable theoretical properties
- For training deep neural network models, however, accumulation of squared gradients from the beginning of training can result in a premature and excessive decrease in the effective learning rate
  - Correction: RMSProp
- AdaGrad performs well for some but not all deep learning models

RMSProp

- RMSProp modifies AdaGrad to perform better in the non-convex setting by changing the gradient computation into an exponentially weighted moving average

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

RMSProp

- RMSProp modifies AdaGrad to perform better in the non-convex setting by changing the gradient computation into an exponentially weighted moving average
- 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
- RMSProp is an effective and practical optimization algorithm for deep neural networks. It is one of the go-to optimization methods by deep learning practitioners

RMSProp with Nesterov Momentum

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

Choosing the Right Opt. Algorithm

- There is no consensus on this point
- Algorithms with adaptive learning rates perform robustly, but no single best algorithm has emerged
- The most popular optimization algorithms actively in use
  - SGD
  - SGD with momentum
  - RMSProp
  - RMSProp with momentum
  - AdaDelta
  - Adam

What you need to know

- Challenges in neural network optimization
  - Ill-conditioning, local minima, cliffs and exploding gradients, long-term dependencies, and poor correspondence between local and global structure
- Basic learning algorithms
  - SGD, momentum, and Nesterov momentum
- Algorithms with adaptive learning rates
  - AdaGrad and RMSProp

Questions?