Optimization for Training Deep Models

presented by Kan Ren

Table of Contents

How Learning Differs from Pure Optimization

Optimization for ML

- Goal and Objective Function
  - **ML** (goal not always equal to obj func)
    - Goal: evaluation measure AUC
    - Obj func: cross entropy, squared loss
  - **Pure Optimization** (goal = obj func)

Objective Function

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

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

Empirical Risk Minimization

- Risk minimization
- Empirical risk minimization
- if p*(x,y) = p(x,y)
- **ML** is based on empirical risk, **OPT** is based on true risk.

Surrogate Loss Function

- Challenges:
  - empirical risk minimization is prone to overfitting
  - 0-1 loss with no derivatives $L(\hat{y}, y) = I(\hat{y} \neq y)$
- Solution
  - negative log-likelihood of the correct class as surrogate for 0-1 loss
- ML especially for DL is usually based on surrogate loss functions.

Local Minima

- **ML** minimizes a surrogate loss and halts when a convergence criterion (e.g. early stop) is satisfied. i.e. drop into a local minima
  - converges even when gradient is still large
- **OPT** converges when gradient becomes very small.

Batch and Minibatch

- **ML** optimization algorithms typically compute update based on an expected value of cost function using only a subset of the terms of the full cost function.
- why
  - more computations, not much more effectiveness
  - redundancy within training sets $\sigma / \sqrt{n}$.
- batch/deterministic gradient methods = utilize all samples
- stochastic gradient descent = utilize 1 sample

Mini-batch

- utilize >1 and < all samples
- factors of mini-batch size
    - more accurate estimate of the gradient
    - multicore architectures underutilize extremely small batches
    - memory in parallel system scales batch size
    - specific hardware better run with specific sizes of arrays
    - small batch offers regularizing effect (Wilson 2003)

Mini-batch

- Unrepeated mini-batch learning models generalization error.

$$J^*(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \sum_{y} p_{\text{data}}(\boldsymbol{x}, y) L(f(\boldsymbol{x}; \boldsymbol{\theta}), y)$$

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J^*(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \sum_{y} p_{\text{data}}(\boldsymbol{x}, y) \nabla_{\boldsymbol{x}} L(f(\boldsymbol{x}; \boldsymbol{\theta}), y)$$

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

- Tips of mini-batch learning
  - shuffle dataset
  - parallel computing

Challenges in Neural Network Optimization

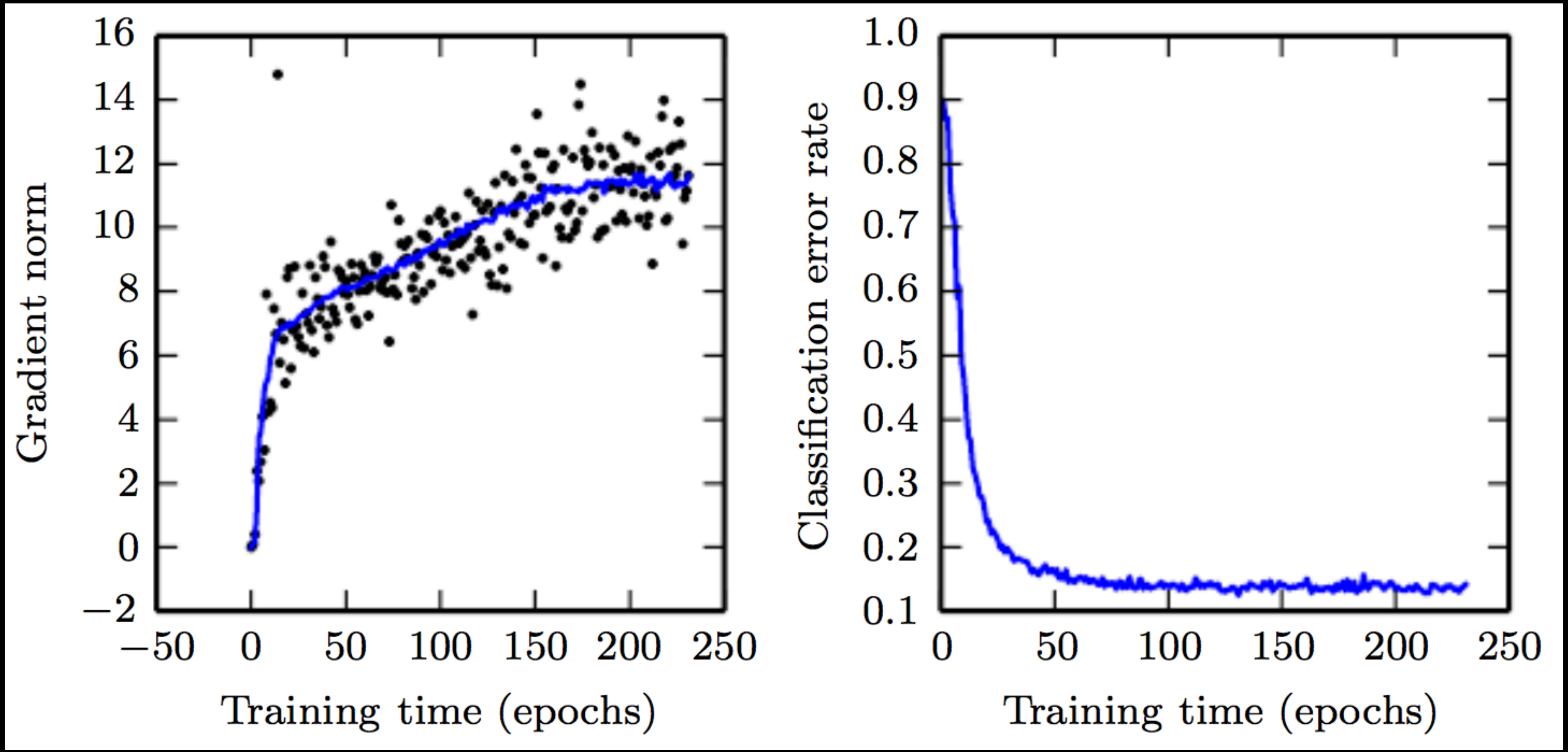
Challenges

- General non-convex case
- Ill-conditioning
  - methods to solve it needs modification for NN
- Local Minima

ill-conditioning

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

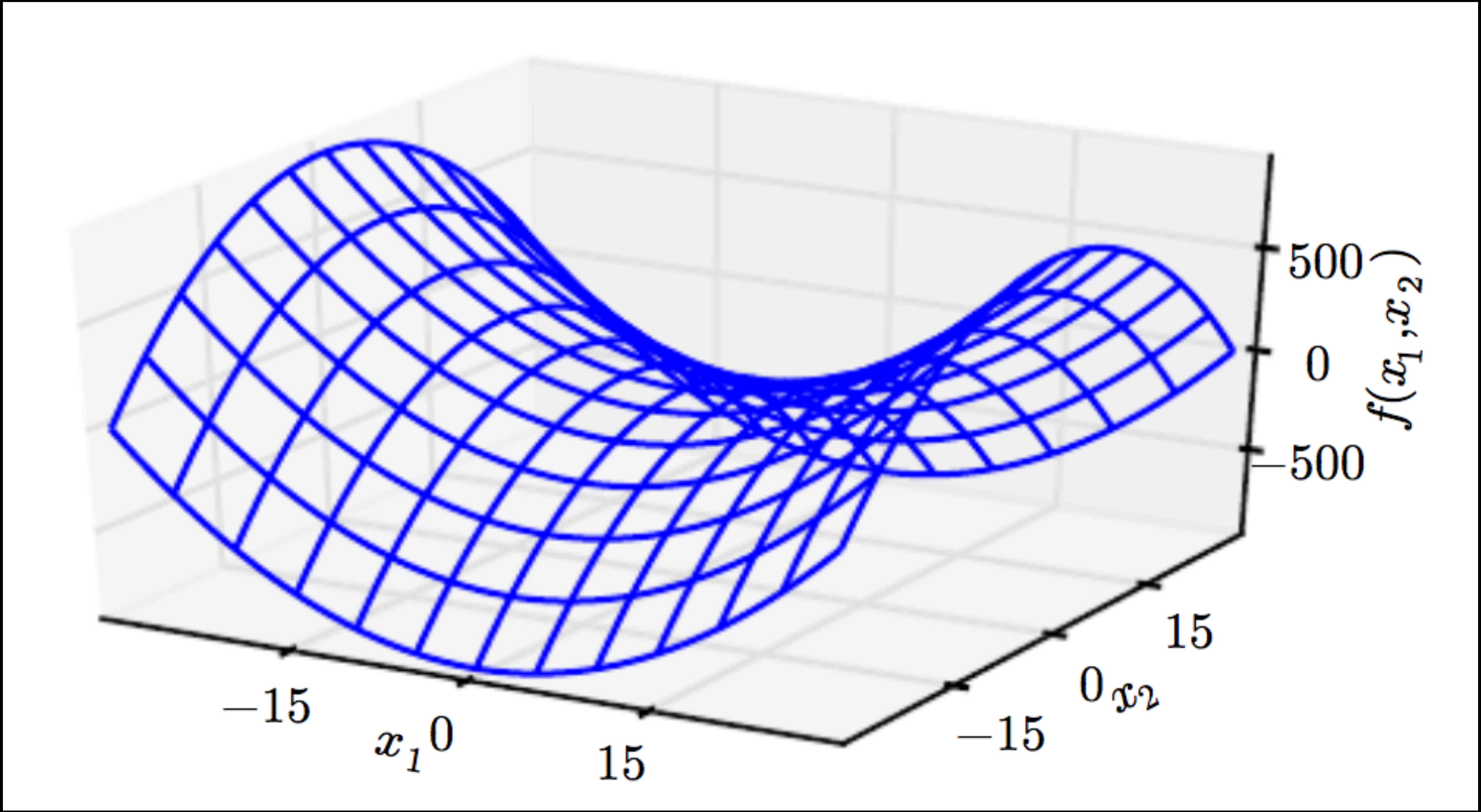
$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$$



Local minima

- Model identifiability
  - A model is said to be 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
  - m layers with n units each -> n!^m ways of arranging hidden unites (weight space symmetry)

Local minima



Saddle Points

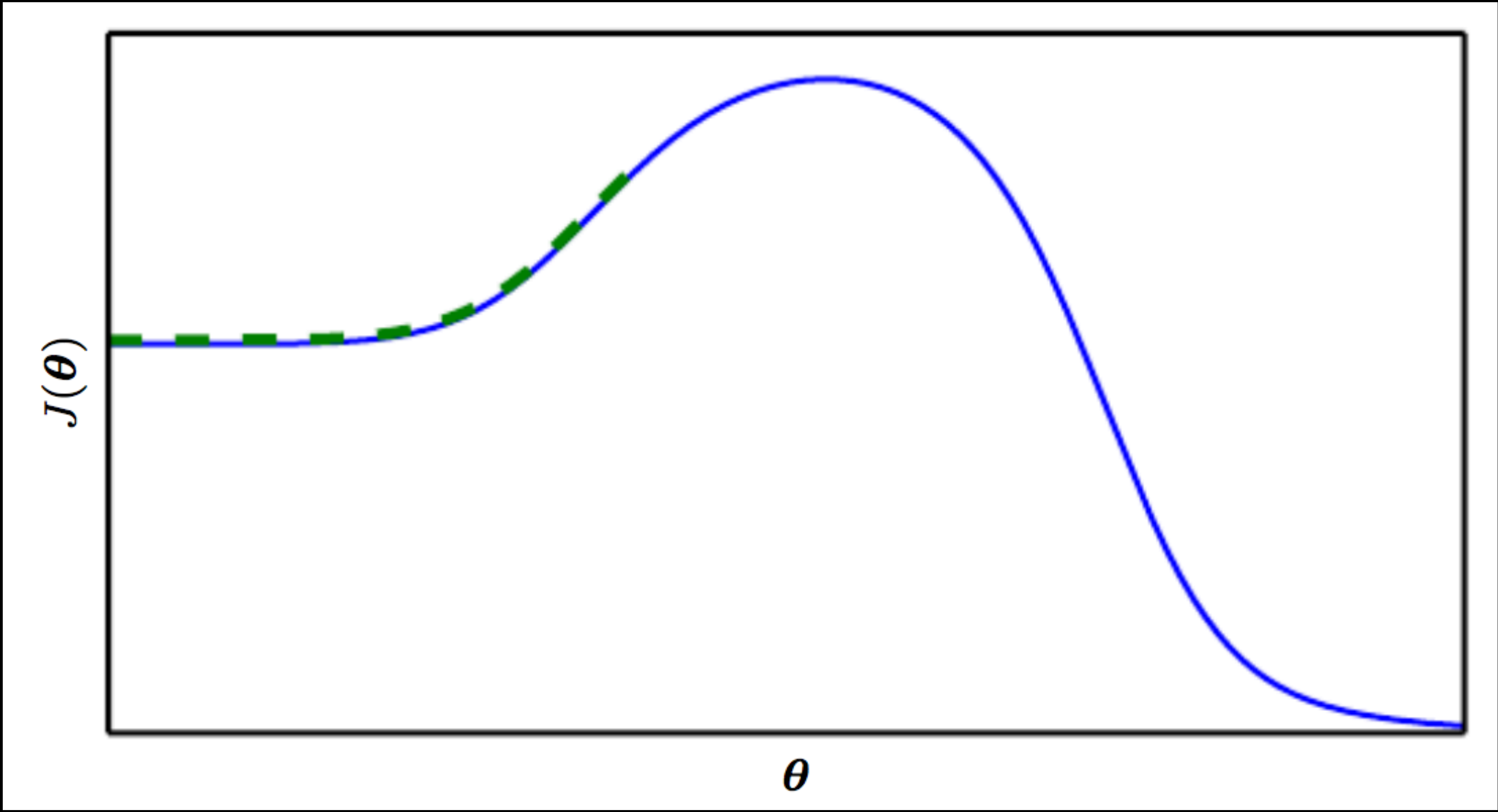
- Gradient Descent is designed to move "downhill".
- Newton's method is to solve a point where the gradient is zero.
  - Dauphin (2014): saddle free Newton method

Long-Term Dependencies

- Repeated application of the same parameters (RNN)

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

Poor correspondence between local and global structure

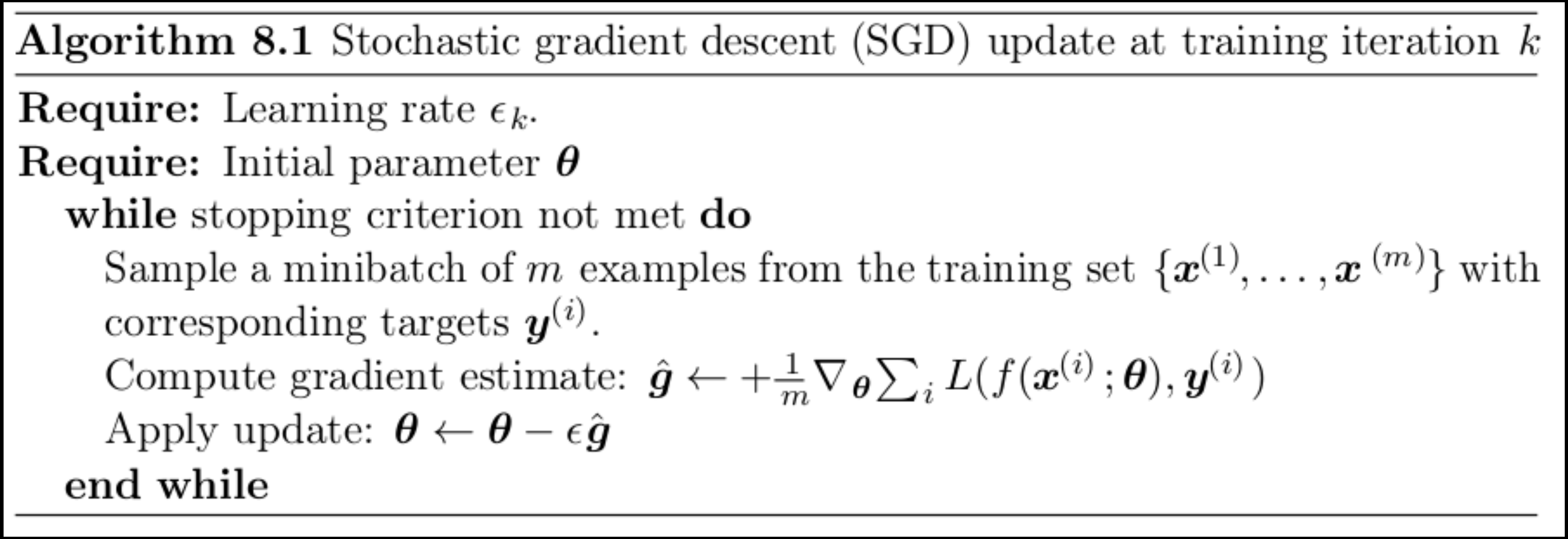


Basic Algorithms

Stochastic Gradient Descent

- sufficient condition to guarantee convergence of SGD
  - $\sum_{k=1}^{\infty} \epsilon_k = \infty$ $\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$
- a bit higher than the best performing learning rate monitored in the first 100 iterations or so.

Stochastic Gradient Descent



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

```
Require: Learning rate $\epsilon_k$.
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
```

Convergence Rate of SGD

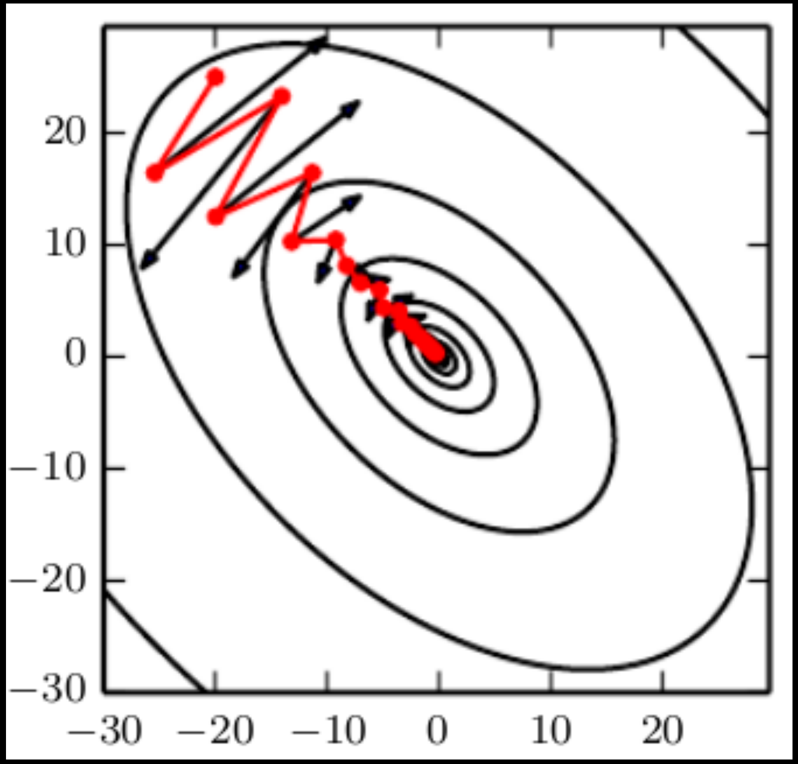
- excess error: e = J(w) - min_w J(w)
- after k iterations
    - convex problem: e = O(1/sqrt(k))
    - strong convex: e = O(1/k)
- presumably overfit when converge faster than O(1/k) of generation error, unless make some assumptions

Momentum

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

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

- v (velocity) is exponentially decaying average of negative gradient
- unit mass



Momentum

- When the same direction occurs, the maximum terminal velocity happens when terminal velocity ends in $\frac{\epsilon||\boldsymbol{g}||}{1-\alpha}$
- If alpha = 0.9/0.99/...

Physical View of Momentum

- position $\boldsymbol{\theta}(t)$
- force onto the particle $\boldsymbol{f}(t) = \frac{\partial^2}{\partial t^2}\boldsymbol{\theta}(t)$ $\boldsymbol{f}(t) = \frac{\partial}{\partial t}\boldsymbol{v}(t)$
- velocity of the particle at time t $\boldsymbol{v}(t) = \frac{\partial}{\partial t}\boldsymbol{\theta}(t)$
- two forces
  - downhill force $-\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  - viscous drag force $-\boldsymbol{v}(t)$

Nesterov Momentum

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

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

- add a correction factor to the standard method of momentum
- convex batch gradient case: O(1/k^2) convergence of excess error
- stochastic gradient descent O(1/k)

Initialization Strategies

Difficulties

- Deep learning has no such luxuries.
  - Normal Equation
  - Convergence to acceptable solution regardless of initialization
- Simple initialization strategies
  - achieve good properties after initialization
  - no idea about which property is preserved after proceeding
- Some initial points may be beneficial for optimization but detrimental for generalization

Break Symmetry

- Same inputs, same activation function, better to initialize different parameters

- Aims to capture more patterns in both feed-forward and back-propagation procedures

- Random initialization from a high-entropy distribution over a high-dimensional space is computationally cheaper and unlikely to symmetry.

Random Initialization

- Drawn from Gaussian Distribution or uniform distribution
- not very small, large weights may help more to break symmetry
- not very large, may activation function saturation or hard to optimize

Heuristic: Uniform Distribution

- initialize the weights of a fully connected layer with m inputs and n outputs by sampling from U(-1/sqrt(m), 1/sqrt(n))
  - Glorot 2010: normalized initialization
    - assumes a chain of matrix multiplication without non linearities
    - $\mathrm{W}_{i,j} \sim U(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}})$

Heuristic: Orthogonal Matrix

- Saxe 2013: orthogonal matrix initialization
  - chosen scaling or **gain factor** for the nonlinearity applied at each layer
  - They derive specific values of the scaling factor for different types of nonlinear activation functions
- Sussillo 2014: correct gain factor
  - sufficient to train as deep as 1000 layers
  - without orthogonal initializations

Heuristic: Sparse Initialization

- Martens 2010
    - each unit is initialized to have k non-zero weights
- impose sparsity
- cost more to coordinate for Maxout unites with several filters

Method: hyper-searching

- Hyperparameters for
  - choice of dense or sparse initialization
  - initial scale of the weights
- what to look at
  - standard deviation of activations or gradients
  - on a single mini-batch of data

Initialization for bias

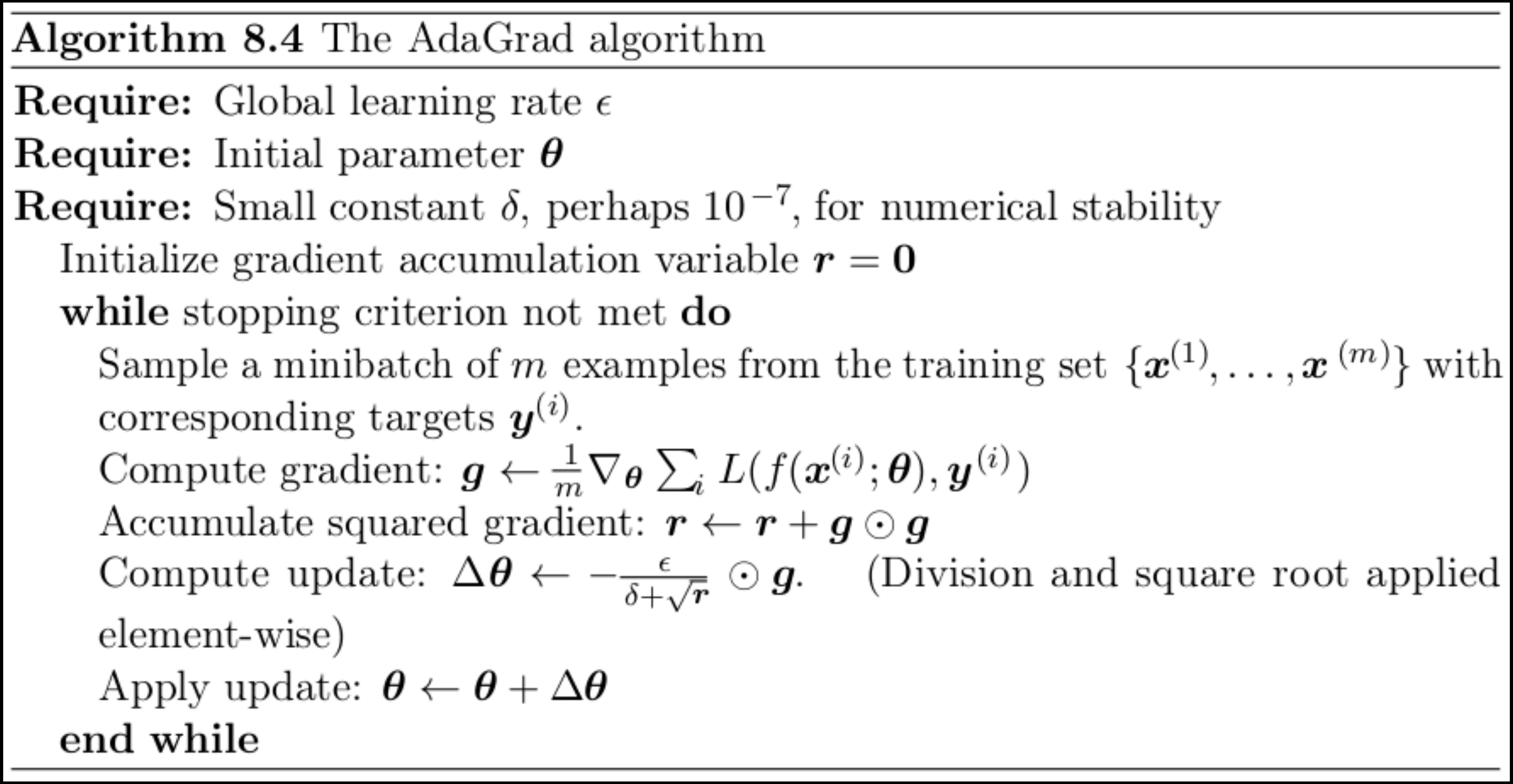
- if bias is for an output unit
  - softmax(b) = c
- to avoid saturation at initialization
  - set bias 0.1 in ReLU hidden unit rather than 0
- for controller whether other units to participate
  - u*h ≈ 0/1, initially set h ≈ 1
- variance or precision parameter
  - $p(y \mid \boldsymbol{x}) = \mathcal{N}(y \mid \boldsymbol{w}^T \boldsymbol{x} + b, 1/\beta)$

Algorithms with Adaptive Learning Rates

Learning Rate

- A hyper-parameter the most difficult to set
- Jacobs 1988: delta-bar-delta method
    - partial derivatives remain the same sign, then increase the learning rate

AdaGrad

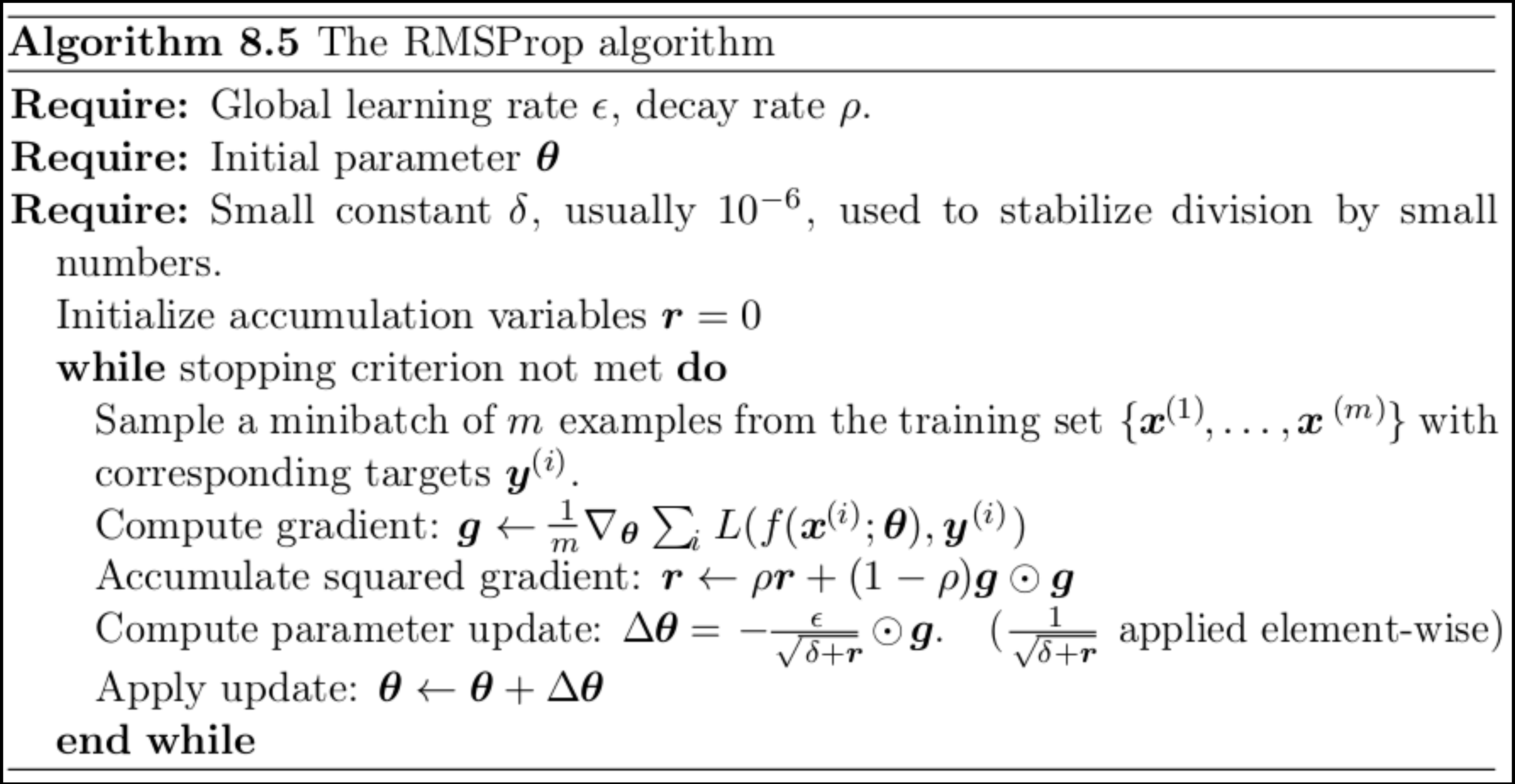


Algorithm 8.4 The AdaGrad algorithm

```
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, perhaps $10^{-7}$, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
  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
```

may cause premature/excessive decrease for learning rate

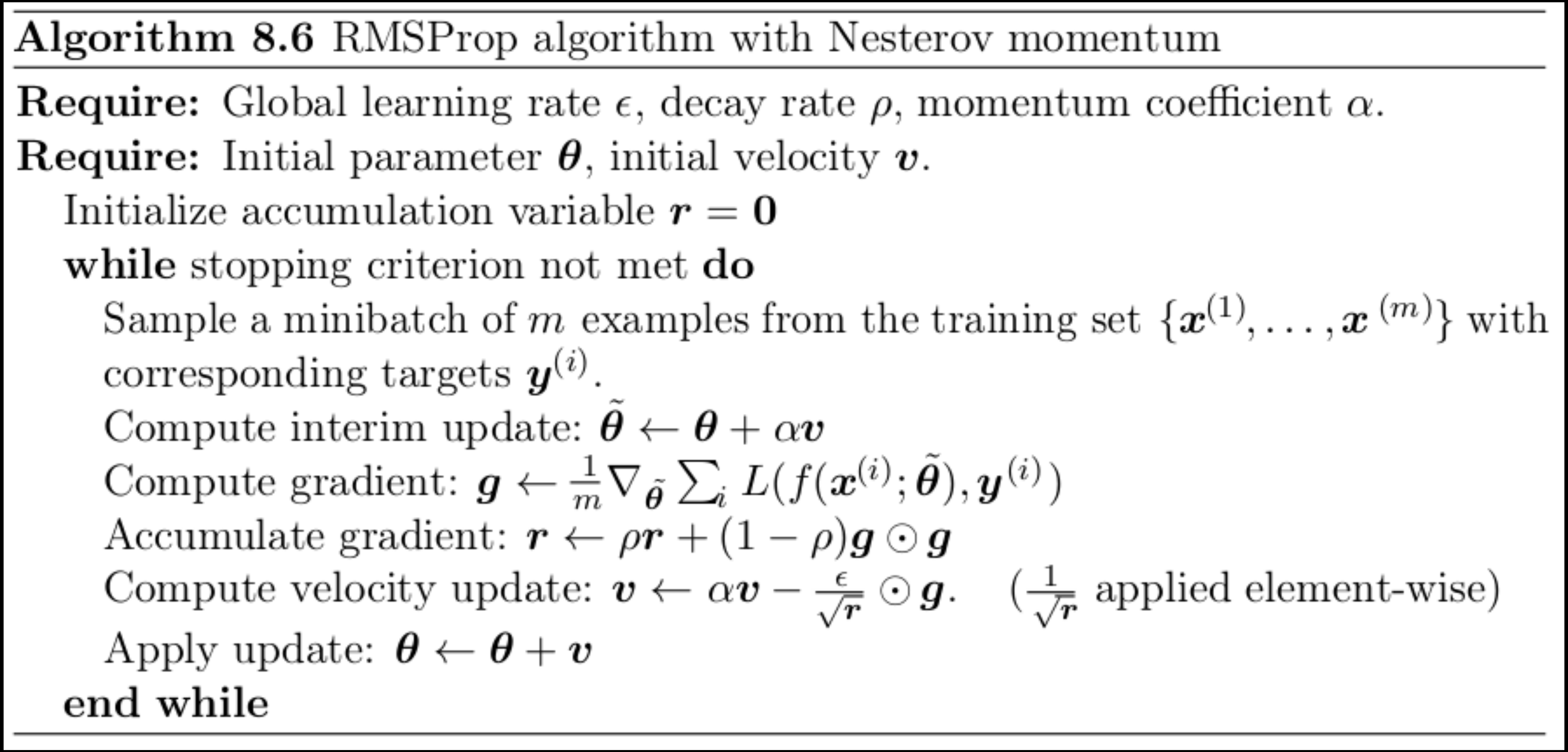
RMSProp



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

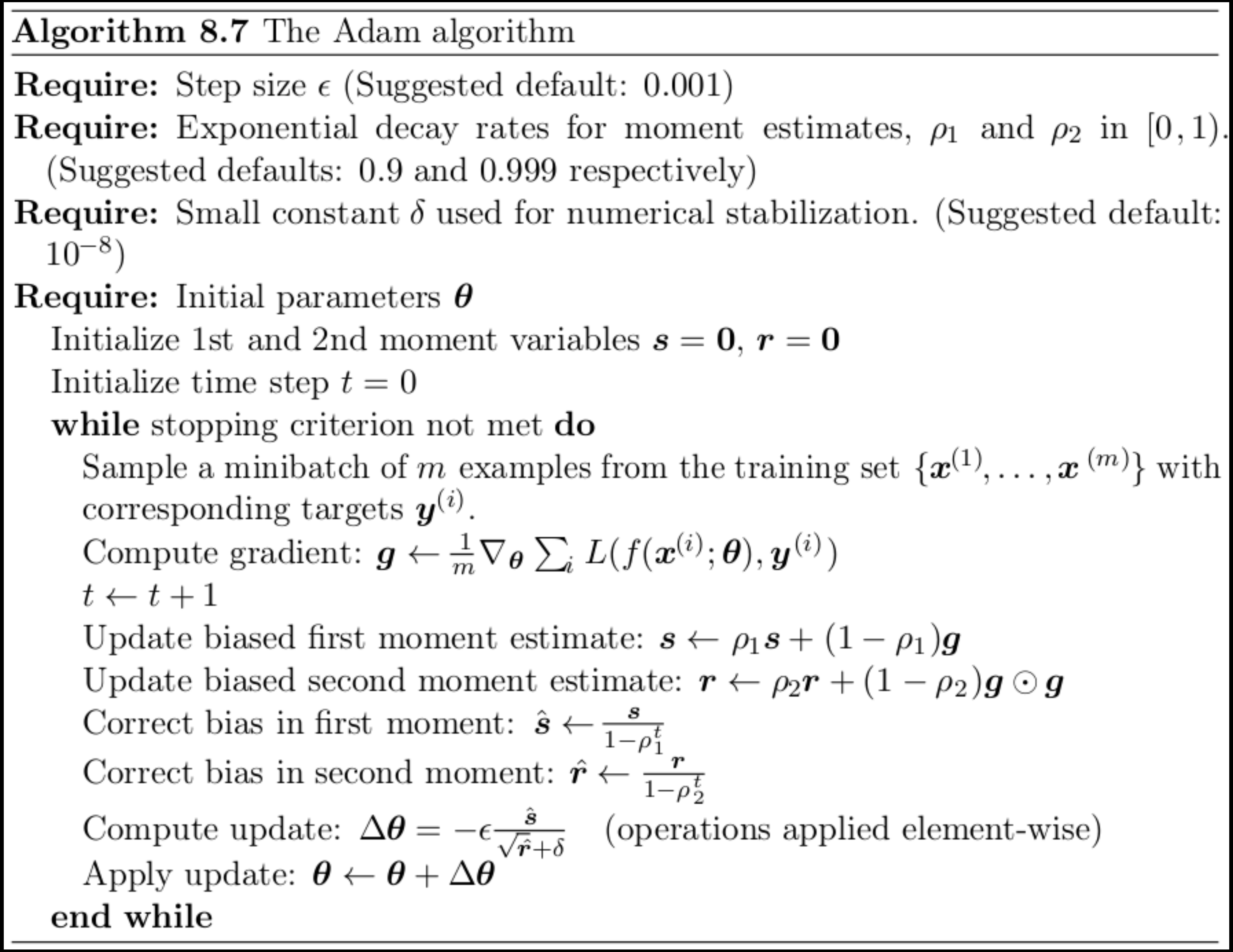
RMSProp with Nesterov momentum



Algorithm 8.6 RMSProp algorithm with Nesterov momentum

```
Require: Global learning rate $\epsilon$, decay rate $\rho$, momentum coefficient $\alpha$.
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
```

Adam



Algorithm 8.7 The Adam algorithm

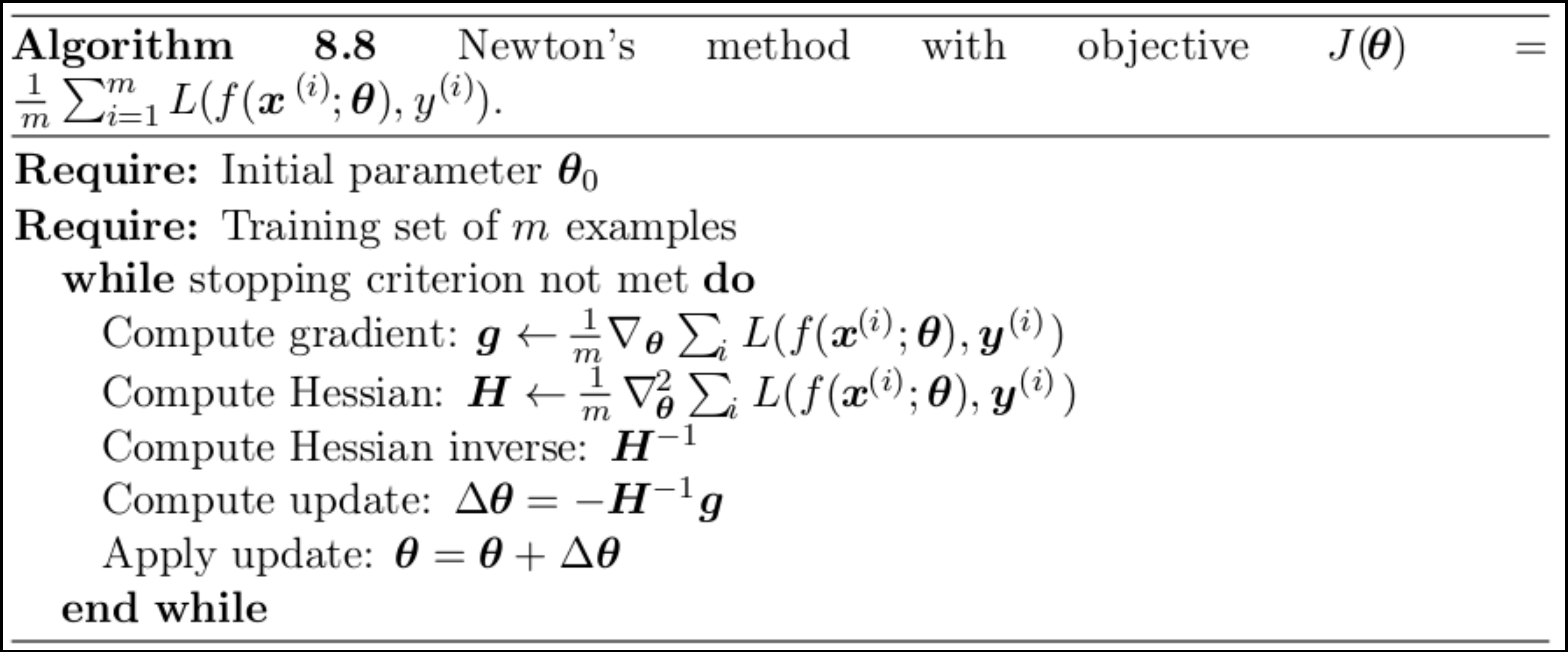
```
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$.
  (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ used for numerical stabilization. (Suggested default:
  $10^{-8}$)
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
```

Visualization

- http://sebastianruder.com/optimizing-gradient-descent/

Approximate 2nd-order Methods

Newton's Method

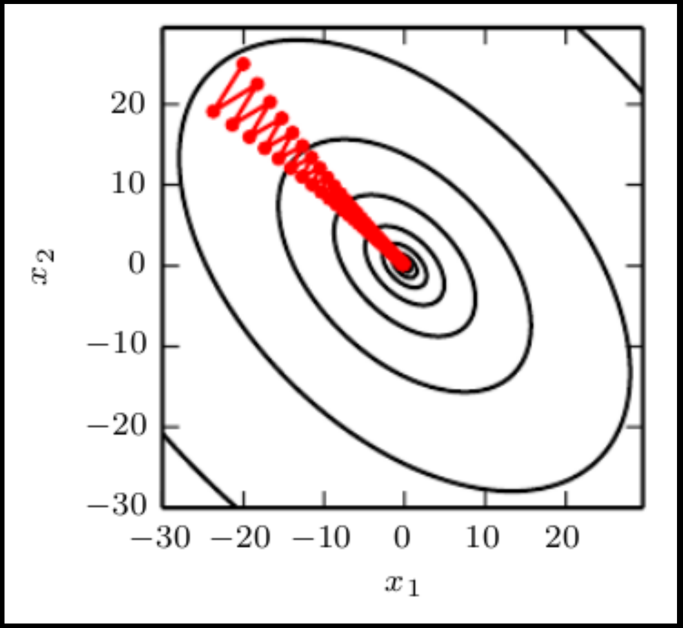


Algorithm 8.8 Newton's method with objective $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$.

```
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
```

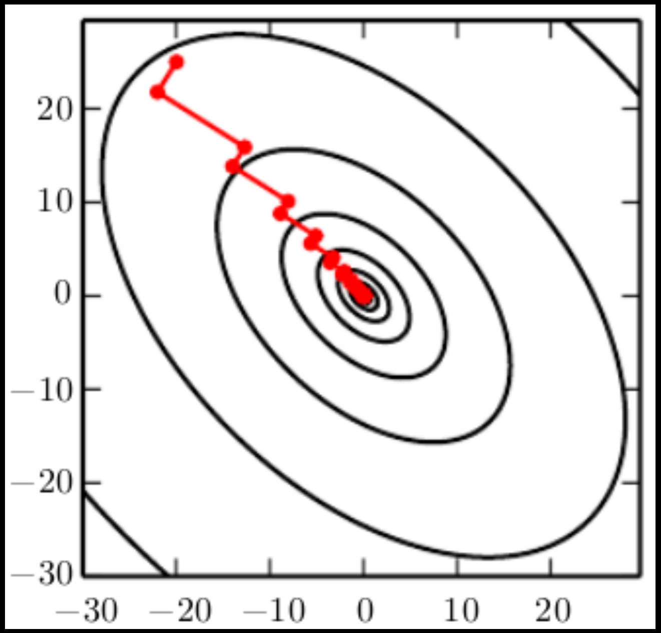
$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \left[H\left(f(\boldsymbol{\theta}_0)\right) + \alpha\boldsymbol{I}\right]^{-1}\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_0).$$

Conjugate Gradients



$$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \beta_t \boldsymbol{d}_{t-1}$$

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



1. Fletcher-Reeves:

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

2. Polak-Ribière:

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

BFGS

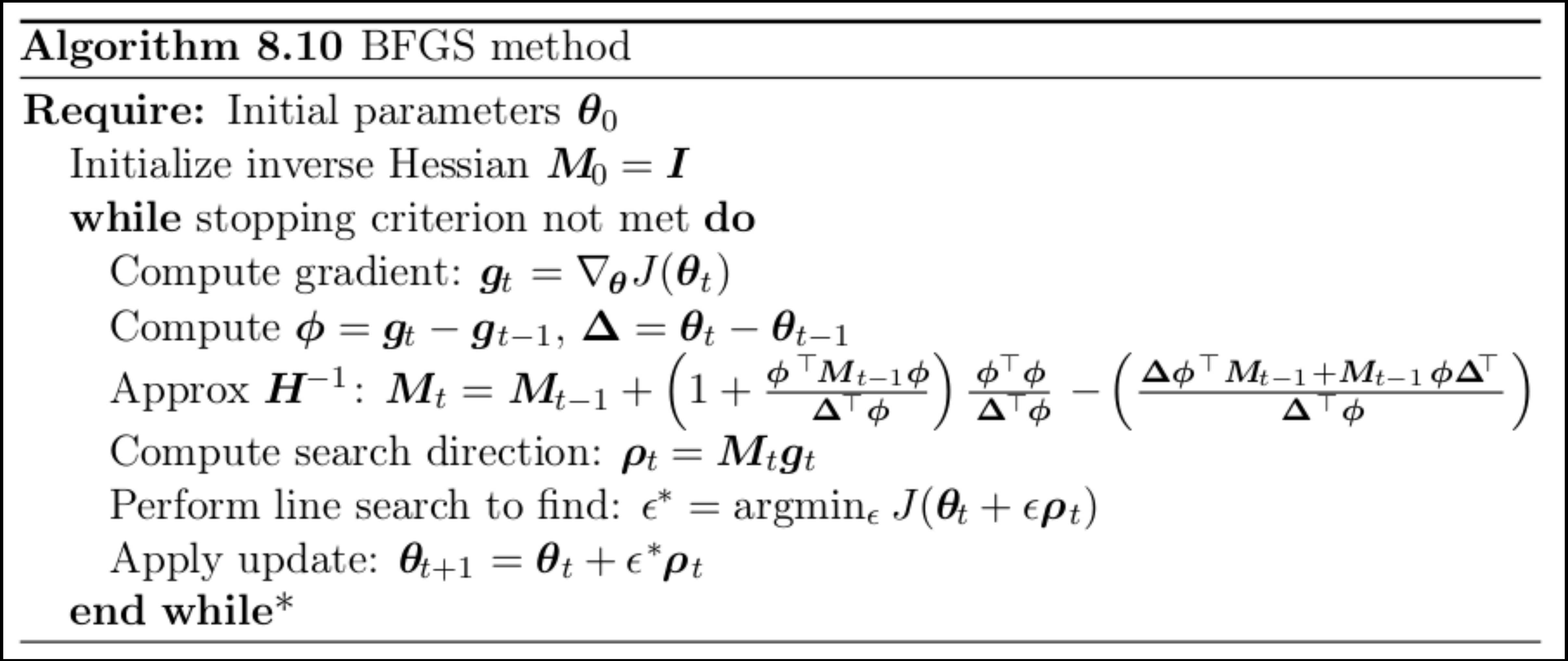
- Newton's method: $\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$
- secant condition (quasi-Newton condition):

$$\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t = -\boldsymbol{H}^{-1}\left(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t+1}) - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)\right)$$

- Approximation of inverse of the Hessian inverse
- $$\boldsymbol{M}_t = \boldsymbol{M}_{t-1} + \left(1 + \frac{\boldsymbol{\phi}^\top \boldsymbol{M}_{t-1}\boldsymbol{\phi}}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}}\right)\frac{\boldsymbol{\phi}^\top \boldsymbol{\phi}}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}} - \left(\frac{\boldsymbol{\Delta}\boldsymbol{\phi}^\top \boldsymbol{M}_{t-1} + \boldsymbol{M}_{t-1}\boldsymbol{\phi}\boldsymbol{\Delta}^\top}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}}\right)$$

where $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$, $\boldsymbol{\phi} = \boldsymbol{g}_t - \boldsymbol{g}_{t-1}$ and $\boldsymbol{\Delta} = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$

BFGS



Algorithm 8.10 BFGS method

```
Require: Initial parameters $\boldsymbol{\theta}_0$
  Initialize inverse Hessian $\boldsymbol{M}_0 = \boldsymbol{I}$
  while stopping criterion not met do
    Compute gradient: $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$
    Compute $\boldsymbol{\phi} = \boldsymbol{g}_t - \boldsymbol{g}_{t-1}$, $\boldsymbol{\Delta} = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$
    Approx $\boldsymbol{H}^{-1}$: $\boldsymbol{M}_t = \boldsymbol{M}_{t-1} + \left(1 + \frac{\boldsymbol{\phi}^\top \boldsymbol{M}_{t-1} \boldsymbol{\phi}}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}}\right) \frac{\boldsymbol{\phi}^\top \boldsymbol{\phi}}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}} - \left(\frac{\boldsymbol{\Delta} \boldsymbol{\phi}^\top \boldsymbol{M}_{t-1} + \boldsymbol{M}_{t-1} \boldsymbol{\phi} \boldsymbol{\Delta}^\top}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}}\right)$
    Compute search direction: $\boldsymbol{\rho}_t = \boldsymbol{M}_t \boldsymbol{g}_t$
    Perform line search to find: $\epsilon^* = \text{argmin}_\epsilon J(\boldsymbol{\theta}_t + \epsilon \boldsymbol{\rho}_t)$
    Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$
  end while*
```

L-BFGS

- Limited Memory BFGS

$$\boldsymbol{\rho}_t = -\boldsymbol{g}_t + b\boldsymbol{\Delta} + a\boldsymbol{\phi}.$$

$$a = -\left(1 + \frac{\boldsymbol{\phi}^\top \boldsymbol{\phi}}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}}\right) \frac{\boldsymbol{\Delta}^\top g_t}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}} + \frac{\boldsymbol{\phi}^\top g_t}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}}$$

$$b = \frac{\boldsymbol{\Delta}^\top g_t}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}}$$

-

Optimization Strategies and Meta-Algorithms

Batch Normalization

$$\hat{y} = x w_1 w_2 w_3 \dots w_l.$$

$$x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2) \dots (w_l - \epsilon g_l)$$

- effect of the update of parameters has $\epsilon^2 g_1 g_2 \prod_{i=3}^{l} w_i$ for second-order term of Taylor series approximation of y(hat).

- perhaps solution

  - second-order / n-th order optimization, hopeless

Batch Normalization

- H' = (H - mu) / sigma
    - mu: mean of each unit $$\boldsymbol{\mu} = \frac{1}{m}\sum_i \boldsymbol{H}_{i,:}$$
    - sigma: standard deviation $$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_i (\boldsymbol{H} - \boldsymbol{\mu})_i^2}$$
- we back-propagate through these operations for computing the mean and the standard deviation, and for applying them to normalize H
- not changes a lot if lower layer changes
    - except for lower layer weights to 0 or changing the sign

Batch Normalization

- expressions of NN has been reduced
  - replace H' with $\gamma \boldsymbol{H}' + \boldsymbol{\beta}$
  - gamma and beta are learned

Coordinate Descent

- repeatedly cycling learning through all variables

$$J(\boldsymbol{H}, \boldsymbol{W}) = \sum_{i,j} |H_{i,j}| + \sum_{i,j} \left(\boldsymbol{X} - \boldsymbol{W}^\top \boldsymbol{H} \right)^2_{i,j}$$

- may has problem in some cost functions, e.g.

$$(x_1 - x_2)^2 + \alpha \left(x_1^2 + x_2^2 \right)$$

Polyak Averaging

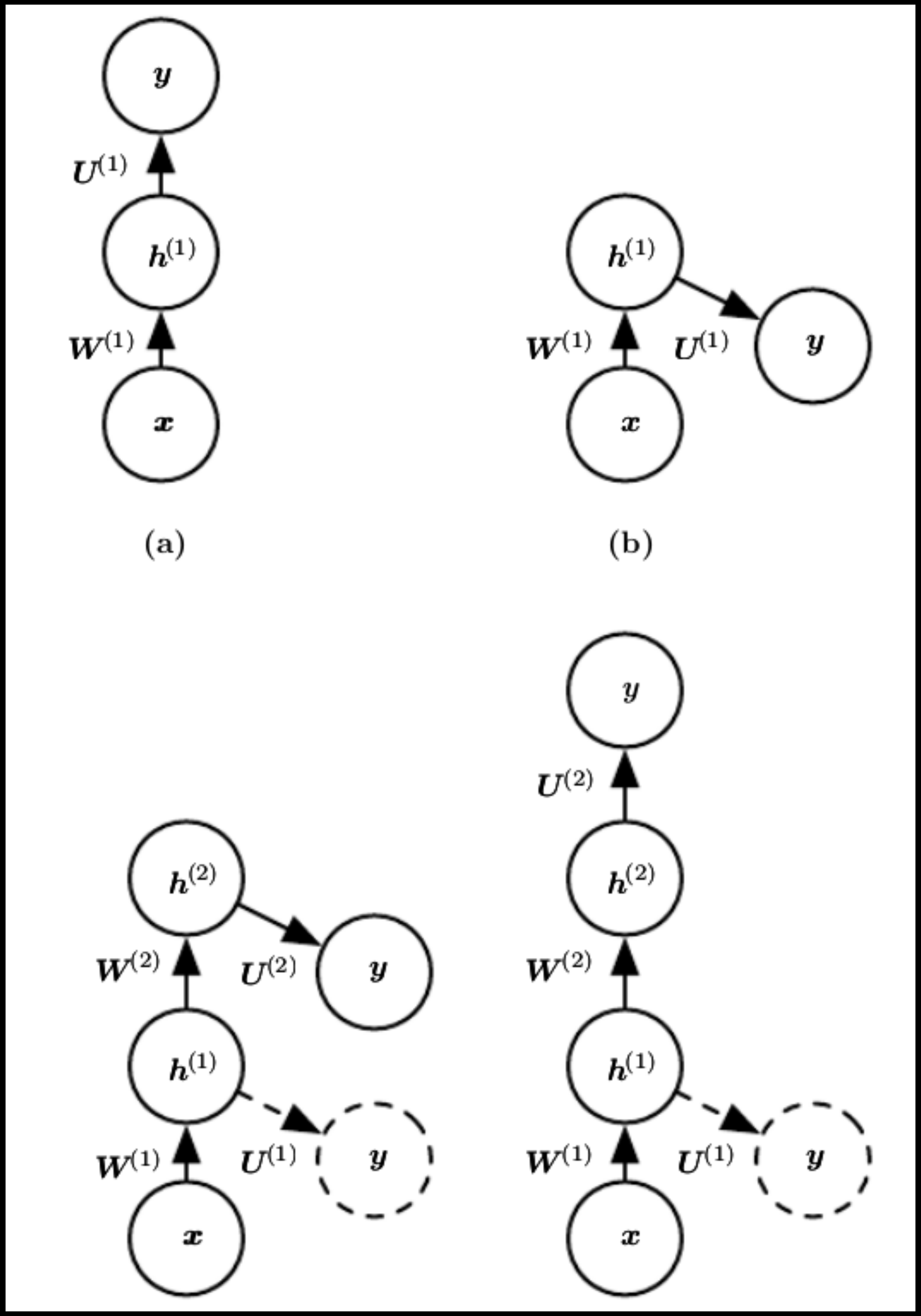
$$\hat{\boldsymbol{\theta}}^{(t)} = \frac{1}{t}\sum_i \boldsymbol{\theta}^{(i)}$$

$$\hat{\boldsymbol{\theta}}^{(t)} = \alpha\hat{\boldsymbol{\theta}}^{(t-1)} + (1-\alpha)\boldsymbol{\theta}^{(t)}$$

Supervised Pretraining

- Pretraining: learn for a difficult task from a simple model
- Greedy: break a problem into comopnents

Greedy Supervised Pretraining



Related Work: Yosinski 2014

- Pretrain a CNN with 8 layers on a set of tasks
- Initialize a same-size net with first k layers of the first net

Related Work: FitNets

- train a low & fat teacher net
- then train a deep & thin student net to
  - predict the output for the original task
  - predict the value of the middle layer of the teacher network

Designing Models to Aid Optimization

- In practice, **it is more important to choose a model family that is easy to optimize than to use a powerful optimization algorithm.**
- skip connections (Srivastava 2015)
- adding extra copies to the output (GoogLeNet, Szegedy 2014, Lee 2014)

Continuation Methods

- The series of cost functions are designed so that a solution to one is a good initial point of the next.
    - $\{J^{(0)}, \ldots, J^{(n)}\}$
- aim to overcome the challenge of local minima
    - reach a global minimum despite the presence of many local minima
- "blurring" the original cost function (non-convex to convex)
    - $J^{(i)}(\boldsymbol{\theta}) = \mathbb{E}_{\theta' \sim \mathcal{N}(\boldsymbol{\theta}'; \boldsymbol{\theta}, \sigma^{(i)2})} J(\boldsymbol{\theta}')$

Table of Contents