EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATHEMATISCH-
NATURWISSENSCHAFTLICHE FAKULTÄT

Optimization for Training Deep Models

Chapter 8:

8.1 How Learning Differs from Pure Optimization

8.1 How Learning Differs from Pure Optimization

- Optimization = training a Neural Network = finding a θ that leads to good results on the training set and all evaluation sets
- The true but unknown cost function (risk) is the expected loss of the data generating distribution p_{data} :
  $$J^*(\theta) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim p_{\text{data}}} L(f(\boldsymbol{x};\theta), y)$$
- The empirical cost function (empirical risk) over the training set is given by:

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

8.1 How Learning Differs from Pure Optimization

- empirical risk minimization
  - The hard goal is to find a minimum of $J^*(\theta)$ by optimizing on $J(\theta)$
  - Finding the minimum of $J(\theta)$ is called empirical risk minimization
    - = finding the minimum on the training set
    - It is rarely used due its prone to overfitting
  - Thus, the training algorithm does not hold at a local minimum. Instead, it minimizes $J(\theta)$ but halts when a convergence criterion based on e.g. early stopping is satisfied
  - Gradient might still be high at this position

8.1 How Learning Differs from Pure Optimization

- Training in practice:
  - Training is usually done by variants of gradient descent
  - The gradient of the empirical loss is given by:
    $\nabla_\theta J(\theta) = \mathbb{E}_{x,y\sim\hat{p}_{\text{data}}}\left[\nabla_\theta L(f(\boldsymbol{x};\theta), y)\right]$
  - Computing this gradient exactly is very expensive since we have to consider the entire training set
  - In practice we sample batches

8.1 How Learning Differs from Pure Optimization

- The error of the gradient estimate:
  - The standard error of the mean estimated from **n** samples is given by $\frac{\sigma}{\sqrt{n}}$
  - **σ** is the true standard defiation of the gradients of all samples.
  - The denominator shows that there are less than linear returns to using more examples to estimate the gradient.
  - Compare two hypothetical estimates of the gradient, one based on 100 examples and another based on 10,000 examples. The latter requires 100 times more computation than the former, but reduces the standard error of the mean only by a factor of 10
    - With a standard deviation of 1 we have an error of 0.1 vs. 0.01

8.1 How Learning Differs from Pure Optimization

- Variants of gradient descent methods:
  - Optimization algorithms that use the entire training set are called batch or deterministic gradient methods. (name originates from 1 large batch)
  - Optimization algorithms that use only a single sample at a time are sometimes called stochastic or online methods
  - Most algorithms used for deep learning fall somewhere in between, using more than one but less than all of the training examples. These are called minibatch stochastic methods

8.1 How Learning Differs from Pure Optimization

- What is the minibatch size to choose?
  - 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. This motivates using some absolute minimum batch size, below which there is no reduction in the time to process a minibatch.
  - The amount of memory scales with the batch size.
  - Some kinds of hardware achieve better runtime with specific sizes of arrays. Especially when using GPUs, it is common for power of 2 batch sizes to offer better runtime.

8.1 How Learning Differs from Pure Optimization

- Regularizing effect of small batches (Wilson and Martinez, 2003),
  - Perhaps due to the noise they add to the learning process.
  - Generalization error is often best for a batch size of 1.
  - Small learning rate to maintain stability due to the high variance in the estimate of the gradient.
  - The total runtime can be very high due to the need to make more steps:
    - reduced learning rate
    - more steps to observe the entire training set.
  - Very rarely seen in practice

8.1 How Learning Differs from Pure Optimization

- Selection of minibatches:
  - Computing an unbiased estimate of the expected gradient from a set of samples, requires that those samples are independent.
  - Samples have to be selected randomly
  - In practice it is usually sufficient to shuffle the order of the dataset once and then store it in shuffled fashion (or shuffle the elements in the loading pipeline)

8.1 How Learning Differs from Pure Optimization

- The bias of gradient estimates:
  - Minibatch stochastic gradient descent follows the gradient of $J^*(\theta)$ (the true generalization error) if there is an infinite stream of data. (it gets biased by reusing data)
  - The generalization error written as a sum is:
    $$J^*(\theta) = \sum_{\boldsymbol{x}} \sum_{y} p_{\text{data}}(\boldsymbol{x}, y) L(f(\boldsymbol{x};\theta), y)$$
  - Its exact gradient is:
    $$\boldsymbol{g} = \nabla_\theta J^*(\theta) = \sum_{\boldsymbol{x}} \sum_{y} p_{\text{data}}(\boldsymbol{x}, y) \nabla_\theta L(f(\boldsymbol{x};\theta), y)$$
  - Its unbiased estimate is given by:
    $$\hat{\boldsymbol{g}} = \frac{1}{m} \nabla_\theta \sum_{i} L(f(\boldsymbol{x}^{(i)};\theta), y^{(i)})$$
  - In practice: as soon as one trains more than one epoch the gradient gets biased. But this is necessary since otherwise $J^*(\theta)$ cannot be minimzed further.

8.2 Challenges in Neural Network Optimization

8.2 Challenges in Neural Network Optimization

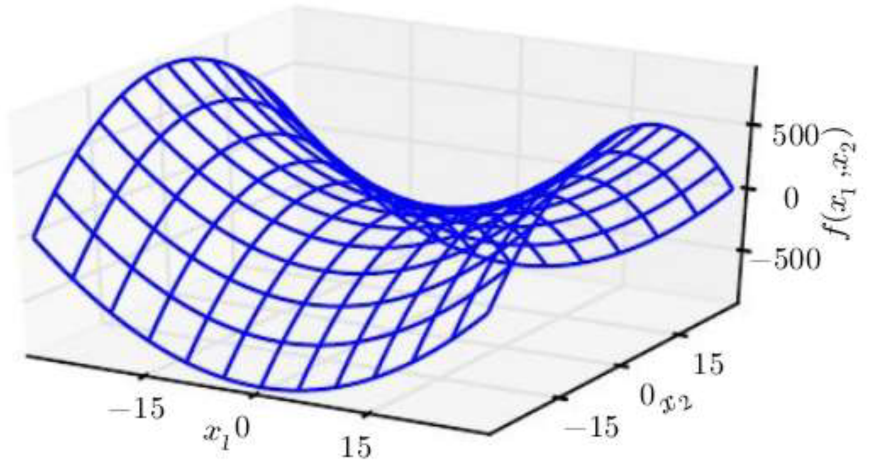
- Why optimization in deep learning is an extremely difficult task:
  - Traditionally, machine learning tried to always construct convex loss functions
  - In deep learning loss functions are in general non-convex
  - See further slides

8.2 Challenges in Neural Network Optimization

1. Local Minima:
   - Don‘t exist for convex loss functions (every local minimum is a global minimum)
   - DNNs have an infinite number of minima:
     - Weight space symmetry (explained on blackboard)
     - Scaling of ReLU networks (explained on blackboard)
     - -> For each minima we can construct an infinite amount of identical further minima
   - Usually, in practice local minima are not a problem since most local minima have a low (enough) loss value

8.2 Challenges in Neural Network Optimization

2. Plateaus, Saddle Points and other Flat Regions:
   - Saddle point <-> gradient is 0 and Hessian has positive and negative eigenvalues.



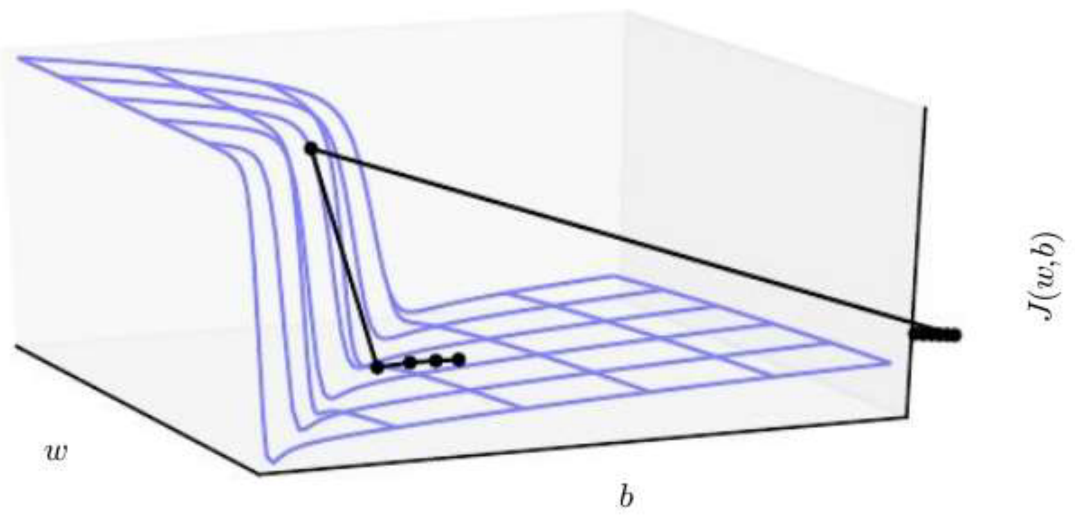
   - In higher dimensional spaces, saddle points get more common than minima.
     - For a minimum the Hessian matrix must have only positive eigenvalues
     - For a saddle point the Hessian matrix must have positive and negative values
     - Imagine the sign of each eigenvalue is generated by flipping a coin

8.2 Challenges in Neural Network Optimization

2. Plateaus, Saddle Points and Other Flat Regions:
   - The eigenvalues of the Hessian tend to become positive in regions of low loss
     - In the coin analogy this means that the probability for a positive value increases
     - This means that local minima are much more likely to have low cost than high cost
     - Fits empirical observations

8.2 Challenges in Neural Network Optimization

3. Cliffs and exploding Gradients:
   - Gradients become very large at cliffs
   - The gradient provides a good direction but not an optimal step size
   - Large step might lead to a higher loss
   - Solution: Gradient Clipping or walking in directions of low slope (Adam)
   - Cliffs are common for RNNs

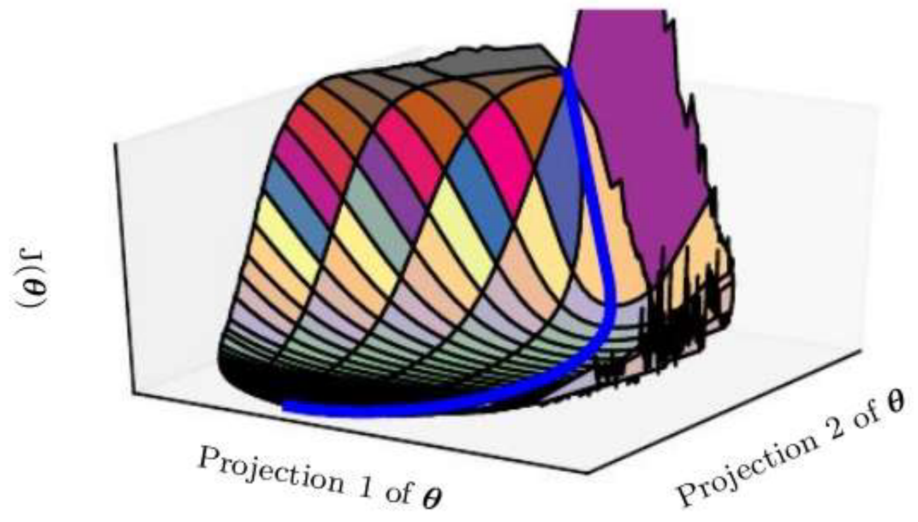


8.2 Challenges in Neural Network Optimization

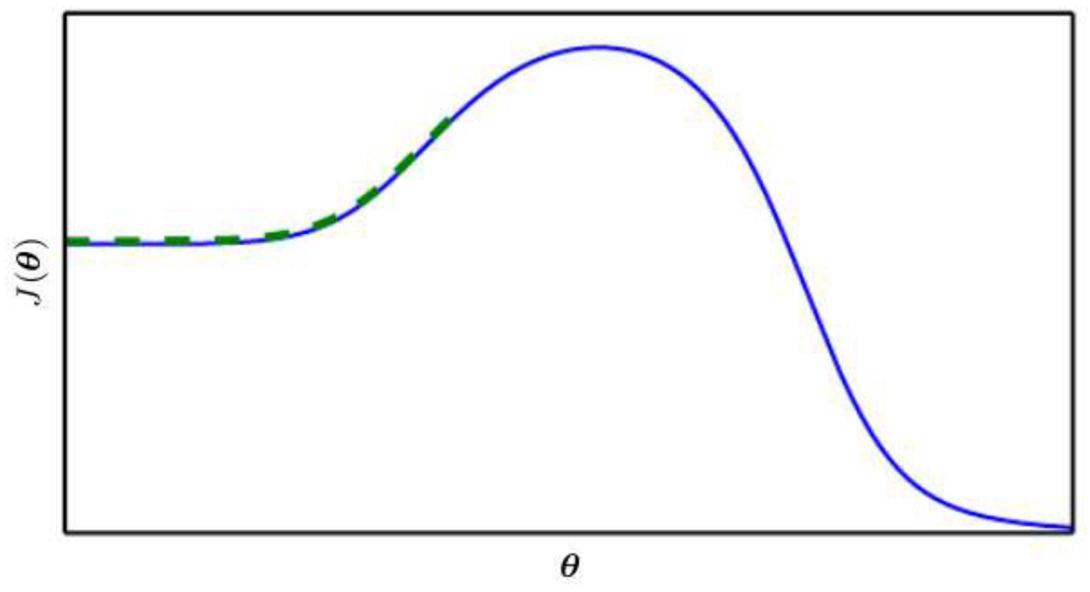
- Long term dependencies (skipped, only relevant for RNNs):
- Inexact Gradients (skipped since explained already earlier)

8.2 Challenges in Neural Network Optimization

4. Poor correspondence between local and global structure
   - The path found by gradient descent might be only locally optimal and might not lead to the global minimum
   - It is important to find good initial points (weight



Non optimal path



Path does not approach global minimum

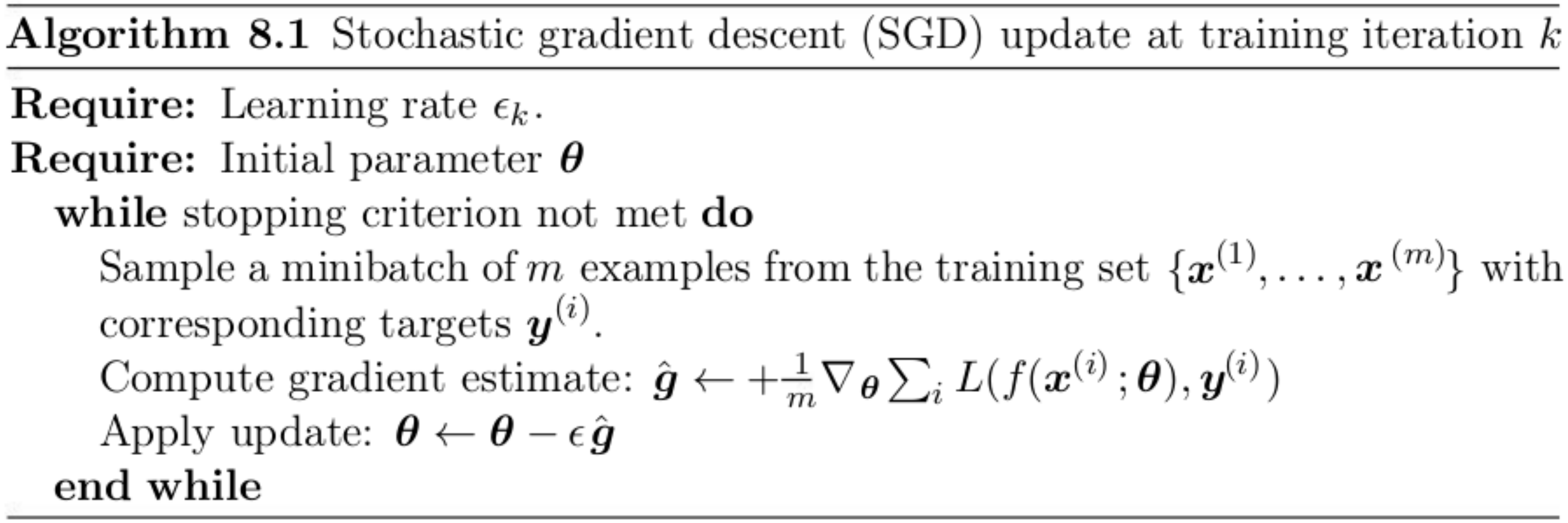
8.2 Challenges in Neural Network Optimization

5. Theoretical Limits of Optimization
   - Till now theoretical results of DNNs have little bearing on the use of neural networks in practice
   - Most results are for practically irrelevant special cases
   - The goal is to find the minimum of the generalization error by optimizing the training error. Theoretical analysis of whether an optimization algorithm can accomplish this goal is extremely difficult.

8.3 Basic Algorithms

8.3 Basic Algorithms

1. 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 $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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
```

8.3 Basic Algorithms

- Decay:
  - The step size has to be decayed over time.
  - Sufficient conditions for convergence are:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \text{ and } \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

  - The book states that one should use a linear decay:

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

  - However, in practice a piecewise constant, exponential or cosine decay is used mostly.
  - The initial step size has usually to be adapted for each problem.

8.3 Basic Algorithms

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

2. Momentum (SGD with Momentum):
   - accelerates learning, especially in the case of high curvature, small but consistent gradients, or noisy gradients
   - accumulates an exponentially decaying moving average of past gradients and continues to move in this direction
   - The update rule is given by:

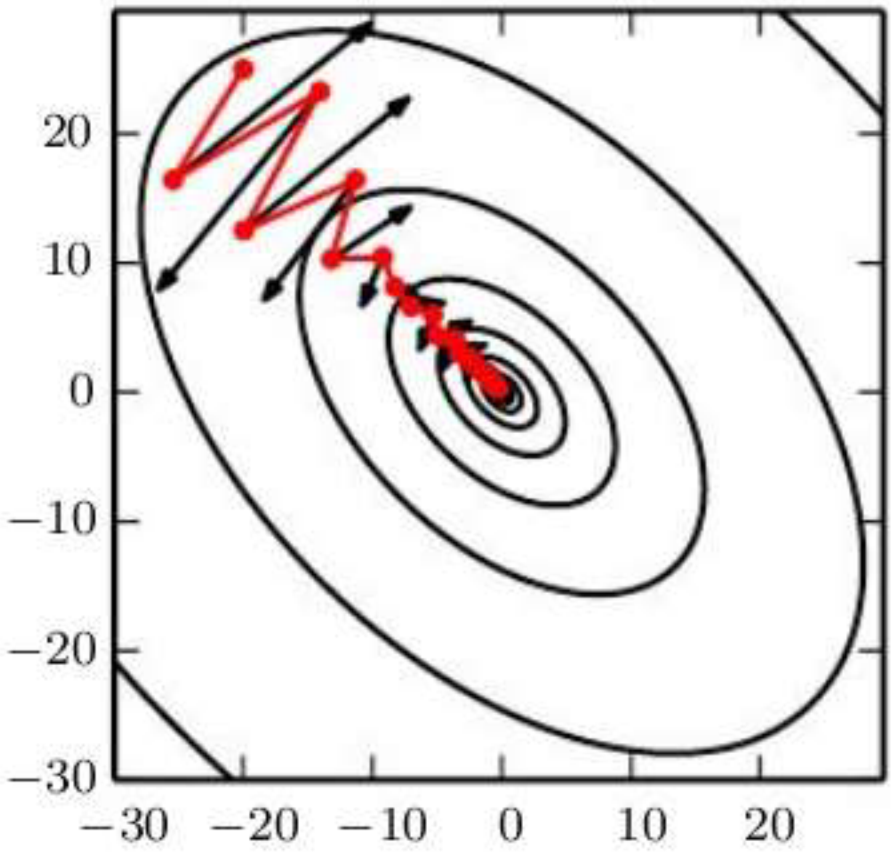
$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)};\theta), \boldsymbol{y}^{(i)}) \right)$$

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

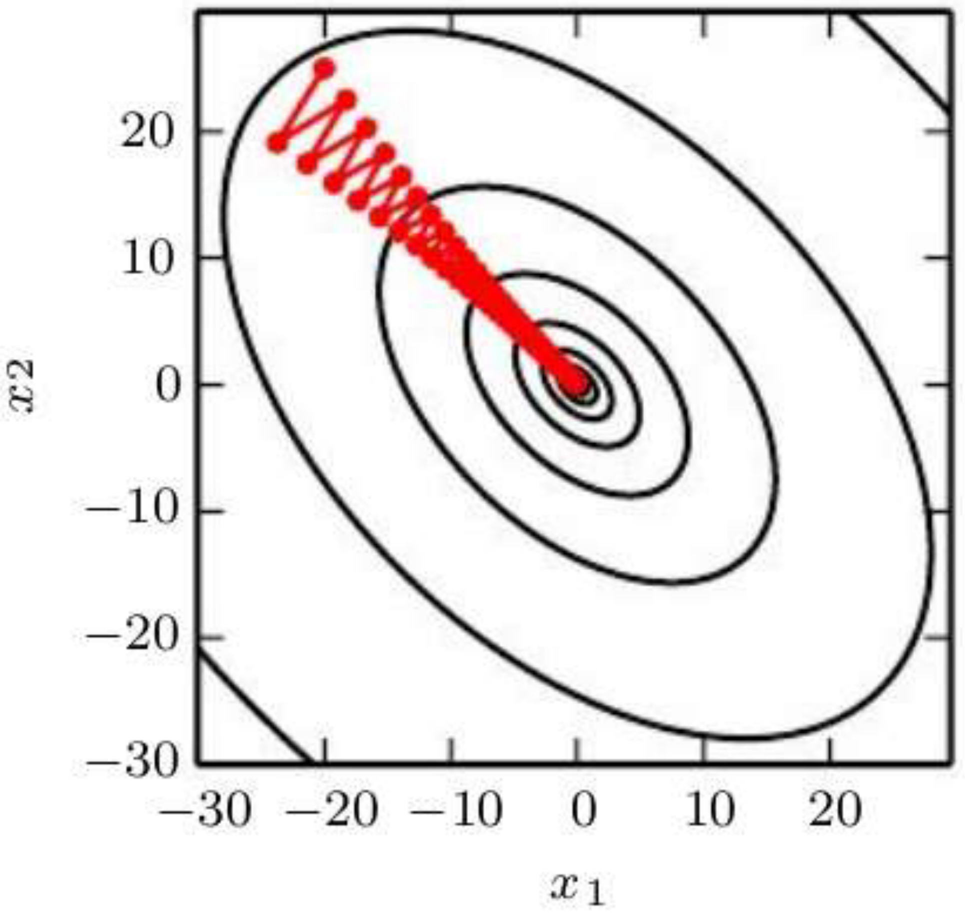
   - The velocity $\mathbf{v}$ accumulates the gradient elements
   - The larger alpha is relative to epsilon the more previous gradients affect the current direction

8.3 Basic Algorithms

2. Momentum (valley with high curvature):



Gradient descent with monemtum



Gradient descent

8.3 Basic Algorithms

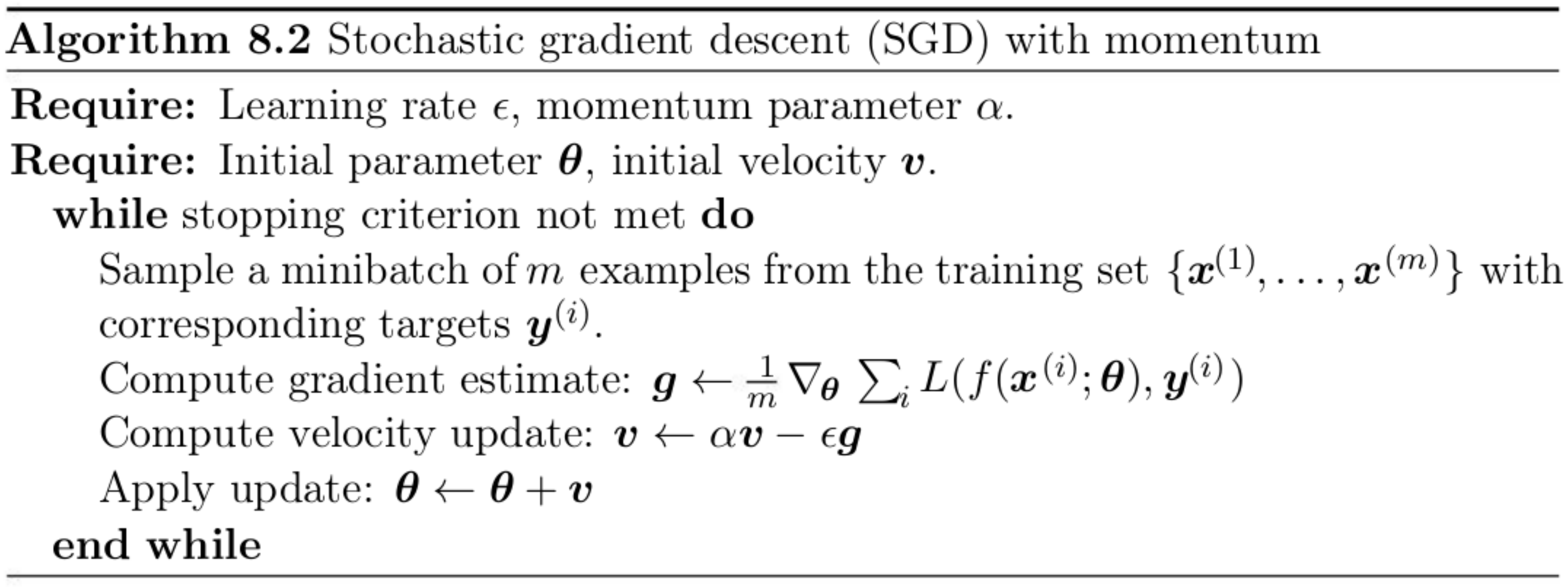
2. Momentum:
   - If the momentum algorithm always observes gradient **g**, then it will accelerate in the direction of −**g** , until reaching a terminal velocity where the size of each step is

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

   - Thus we get a speed up compared to normal gradient descent of $\frac{1}{1-\alpha}$
   - E.g. with an alpha of 0.9 we get a speed up of 10 compared to usual gradient descent (this does not mean better optimization!)
   - It is more important to shrink the step size over time than alpha

8.3 Basic Algorithms

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

8.3 Basic Algorithms

2. Momentum physical interpretation:
   - We consider a particle witch position $\theta(t)$ is given at any point in time
   - We have a force **f** that is changing the parameters:

     $$\boldsymbol{f}(t) = \frac{\partial^2}{\partial t^2}\theta(t)$$

   - We can rewrite this as two first-order differential equation representing a velocity **v** and a force (acceleration) **f**

     $$\boldsymbol{v}(t) = \frac{\partial}{\partial t}\theta(t) \quad \boldsymbol{f}(t) = \frac{\partial}{\partial t}\boldsymbol{v}(t)$$

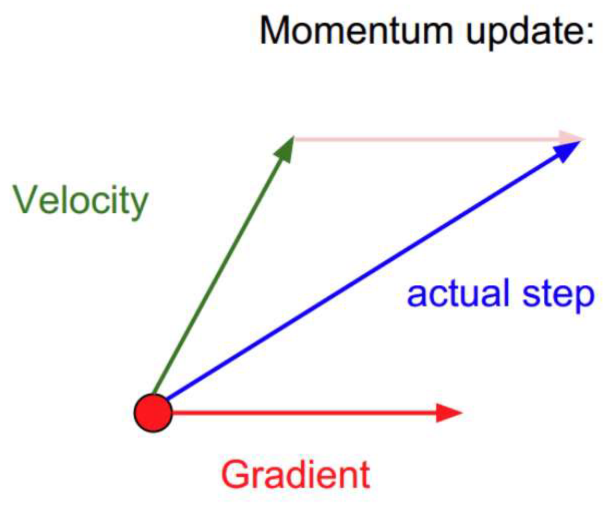
   - One forces is proportional to the negative gradient **–g** which pushes the particle downhill
   - The second force is proportional to **v(t).** It is the viscous drag (or air resistance).

8.3 Basic Algorithms

3. Nesterov Momentum
   - the gradient is evaluated after the current velocity is applied

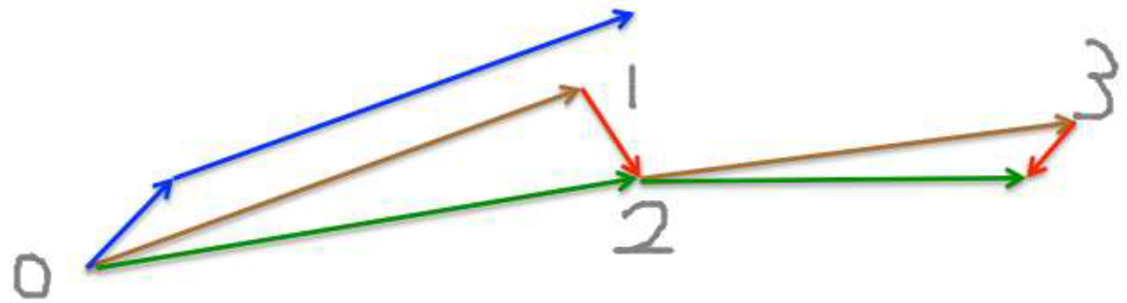
$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_\theta \left[\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \theta + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)}) \right]$$

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



A picture of the Nesterov method

- First make a big jump in the direction of the previous accumulated gradient.
- Then measure the gradient where you end up and make a correction.



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

8.3 Basic Algorithms

4. Parameter Initialization Strategies:
   - Most initialization strategies achieve some nice properties when the network is initialized.
     - no good understanding of which of these properties are preserved after learning begins to proceed
   - The only property known with complete certainty is that the initial parameters need to “break symmetry” between different units (neurons).
   - If two hidden units with the same activation function are connected to the same inputs, then these units must have different initial parameters.
     - If they have the same initial parameters, gradient descent will constantly update both of these units in the same way. (show on blackboard)
   - Thus, each unit should implement a different function -> random initialization of the parameters

8.4 Parameter Initialization Strategies

4. Parameter Initialization Strategies:
   - We can think of initialization of imposing a Gaussian prior p(θ) with mean θ_0.
   - From this point of view, it makes sense to choose θ_0 to be near 0.
   - Since, it is more likely that units do not interact with each other than that they do interact. Units interact only if the objective function expresses a strong preference for them to interact.
   - On the other hand, if we initialize θ_0 to large values, then our prior specifies which units should interact with each other, and how they should interact. In this case learning has to suppress man already existing interaction paths.

8.4 Parameter Initialization Strategies

- Usually *normalized initialization* by Glorot and Bengio (2010) is used for dense layers with **m** inputs and **n** outputs:

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

- It compromises between:
  - the goal of initializing all layers to have the same activation variance
  - and the goal of initializing all layers to have the same gradient variance
- In practice the biases are intitialized with 0

8.5 Algorithms with Adaptive Learning Rates

8.5 Algorithms with Adaptive Learning Rates

- The learning rate is the hyperparameter that is the most difficult to set because it has a significant impact on model performance
- If we believe that the directions of sensitivity are somewhat axis-aligned:
  - use a separate learning rate for each parameter
  - automatically adapt these learning rates throughout the course of learning

8.5 Algorithms with Adaptive Learning Rates

1. 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
   - Parameters with large partial derivatives have rapid decrease of their learning rate
   - Parameters with small partial derivatives have a small decrease in their learning rate
   - The net effect is greater progress in the more gently sloped directions of parameter space
   - Problem: the learning rates are constantly decreasing
   - AdaGrad performs well for some but not all deep learning problems

8.5 Algorithms with Adaptive Learning Rates

1. 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
```

8.5 Algorithms with Adaptive Learning Rates

2. RMSProp
   - AdaGrad, but instead of the squared gradient accumulation a moving average over the squared gradient is used
   - For a non-convex function the learning trajectory may pass through many different structures and eventually arrive at a region that is a locally convex bowl.
   - RMSProp uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex bowl.
   - Performs well on most deep learning problems (especially RNNs)

8.5 Algorithms with Adaptive Learning Rates

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

2. RMSProp

Algorithm 8.5 The RMSProp algorithm

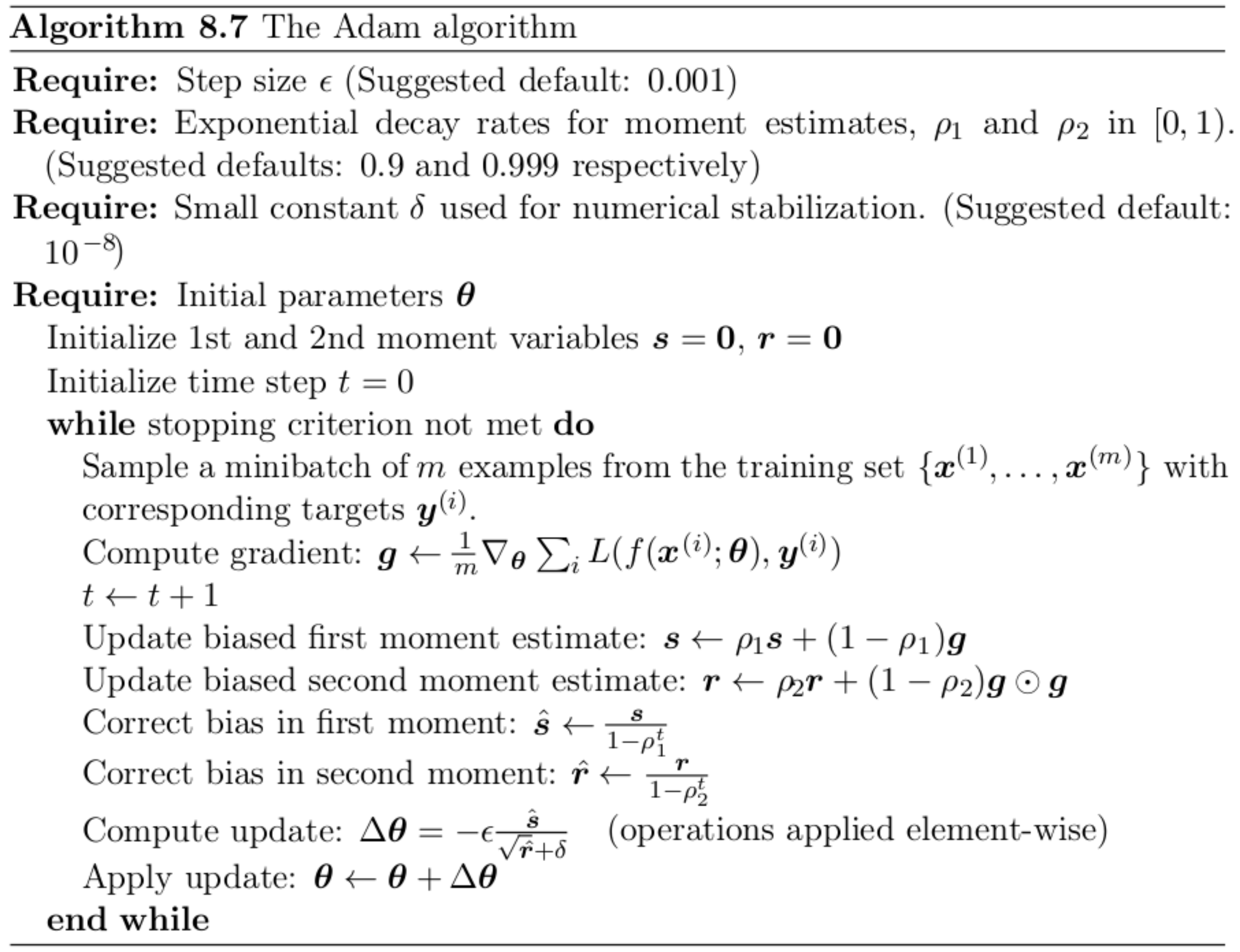
```
Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small
  numbers.
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not 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
```

8.5 Algorithms with Adaptive Learning Rates

3. Adam
   - **Ada**ptive **M**omentum
   - Uses the first order momentum and second order momentum estimates (uncentered Variance). (show on black board)
   - Uses bias corrections for the momentum estimated which accounts for their 0 initialization.
   - Adam is generally regarded as being fairly robust to the choice of hyperparameters, though the learning rate sometimes needs to be changed from the suggested default.
   - Has the expected step size of the 1 times the learning rate.
   - Moves into the direction of less noise (constantly sine changing gradient values).
   - Performs well on most deep learning problems.

8.5 Algorithms with Adaptive Learning Rates

3. 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
```

8.6 Approximate Second-Order Methods

Skipped since usually not time-efficiently applicable for DNNs

8.7 Optimization Strategies and Meta-Algorithms

8.7 Optimization Strategies and Meta-Algorithms

- Batch Normalization
  - Not an optimization algorithm but simplifies the loss function.
  - When we make a parameter update, unexpected results can happen because many functions composed together are changed simultaneously, using updates that were computed under the assumption that the other functions remain constant.
  - Weights at the front of the network might have a large influence on the output.
  - Example of loss functions with n-order Taylor approximations on the black board (book page 319).

8.7 Optimization Strategies and Meta-Algorithms

2. Batch Normalization
   - Method to reduce the sensitivity of the network output to parameter changes.
   - **H =** minibatch of activations of the layer to normalize. One column holds the outputs of one neuron over the minibatch.

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \mu}{\sigma}$$

   - **μ** and **σ** contain the mean and standard deviation of each neuron. They are row vectors, and applied on each row of **H.**

$$\mu = \frac{1}{m} \sum_i \boldsymbol{H}_{i,:}$$

$$\sigma = \sqrt{\delta + \frac{1}{m} \sum_i (\boldsymbol{H} - \mu)_i^2}$$

8.7 Optimization Strategies and Meta-Algorithms

2. Batch Normalization
   - In addition, **H’** is usually shifted and rescaled:

     $$\boldsymbol{H}'' = \gamma \boldsymbol{H}' + \beta$$
   - γ and β are trainable parameters
   - Why did we set he mean to 0, and then introduce a parameter that allows it to be set back to any arbitrary value β?
     - Before the mean of H was determined by complicated interactions between the parameters in the layers below H
     - Now it is only determined by β -> parameters become more independent

8.7 Optimization Strategies and Meta-Algorithms

2. Batch Normalization
    - Is usually applied before the activation function
    - At test time, μ and σ may be replaced by running averages that are collected during training time, since for testing the inference should be deterministic and when applying we don’t have the information of a minibatch and.

8.7 Optimization Strategies and Meta-Algorithms

2. Designing Models to Aid Optimization
   - In general it is more important to choose a model family that is easy to optimize than to use a powerful optimization algorithm
   - ReLu, BatchNorm, L2 norm, skip connections…
   - Stochastic gradient descent is state of the art since 1980