Machine Learning & Pattern Recognition

Xin Xin(辛鑫)

xinxin@sdu.edu.cn

https://xinxin-me.github.io/

Iterative Optimization Technique

Optimization Methods

- Optimization: either minimize or maximize some function $f(x)$ by altering x.
- In most cases, optimizition refers to the minimization of $f(x)$.

Maximization $f(x)$ ⇔ **Minimization** $-f(x)$

- $f(x)$: objective function, cost function, loss function, error function.
- The value that minimize $f(x)$: $x^* = \arg\min f(x)$.

Optimization Methods

- **Deterministic Optimization**
  - The data for the given problem are known accurately.

- **Stochastic Optimization**
  - Refers to a collection of methods for minimizing or maximizing an objective function when randomness is present.

Deterministic Optimization

- First-order methods: methods that use only the gradient.
- Second-order methods: methods that also use the Hessian matrix.

$$\boldsymbol{H}(f)_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x})$$

$\boldsymbol{x}$: multiple input dimensions.

Newton's Methods

- Motivation: to minimize the local second-order Taylor approximation of f.

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \approx \min_{\boldsymbol{x}} f(\boldsymbol{x}_t) + \nabla f(\boldsymbol{x}_t)^T(\boldsymbol{x} - \boldsymbol{x}_t) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_t)^T \nabla^2 f(\boldsymbol{x}_t)(\boldsymbol{x} - \boldsymbol{x}_t)$$

- Take the derivative of $\boldsymbol{x}$ on both side, we have,

$$\frac{df(\boldsymbol{x})}{d\boldsymbol{x}} = \nabla f(x_t) + \nabla^2 f(\boldsymbol{x}_t)(\boldsymbol{x} - \boldsymbol{x}_t) = \mathbf{0}$$

- Update rule: suppose $\nabla^2 f(x_t)$ is positive definite,

$$\boldsymbol{x} = \boldsymbol{x}_t - [\nabla^2 f(x_t)]^{-1} \nabla f(x_t)$$

Newton’s Methods

- **Advantage:**
  - More accurate local approximation of the objective,
  - The convergence is much faster.
- **Disadvantage:**
  - Need to compute the second derivatives
  - Need to compute the inverse of Hessian (time/storage consuming)

Quasi Newton's Methods

- **Main Idea:** To approximate the inverse with a matrix B_t that is iteratively refined by low rank updates to become a better approximation of $[\nabla^2 f(x_t)]^{-1}$.

Quasi Newton’s Methods

- **BFGS (Broyden–Fletcher–Goldfarb–Shanno):**

Optimization Methods

- **Deterministic Optimization**
  - The data for the given problem are known accurately.

- **Stochastic Optimization**
  - Refers to a collection of methods for minimizing or maximizing an objective function when randomness is present.

Stochastic Gradient Descent

We now minimize the *empirical risk*, m is the number of training examples.

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

- Optimization algorithms that use the entire training set simultaneously are called *deterministic* or *batch* gradient methods.

- This terminology "batch" can be somewhat confusing.
  - We use the term "batch size" to describe the size of a minibatch in the stochastic gradient descent.
  - We use the term "batch gradient descent" to imply the use of the full training set.

Stochastic Gradient Descent

- Optimization algorithms that use only a single example at a time are sometimes called *stochastic* or sometimes *online* methods.

- E.g., consider the cost function of linear regression as

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

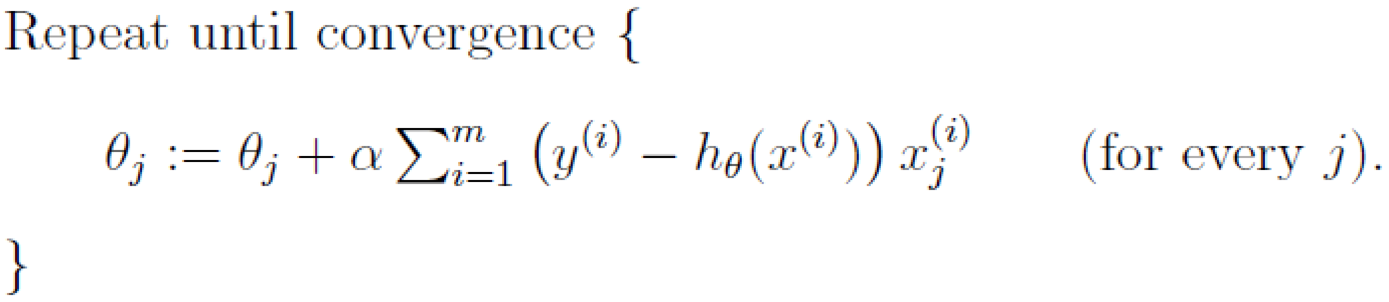
- Now, we ignore the superscript i, then for each x we have ,

$$\begin{aligned}
\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta(x) - y)^2 \\
&= 2\cdot\frac{1}{2}(h_\theta(x) - y)\cdot\frac{\partial}{\partial\theta_j}(h_\theta(x) - y) \\
&= (h_\theta(x) - y)\cdot\frac{\partial}{\partial\theta_j}\left(\sum_{i=0}^{n}\theta_i x_i - y\right) \\
&= (h_\theta(x) - y)\,x_j
\end{aligned}$$

$\boldsymbol{x} \in \mathbb{R}^n$

Batch vs Stochastic Gradient Descent

- Batch gradient descent has to scan through the entire training set before taking a single step—a costly operation if m is large.

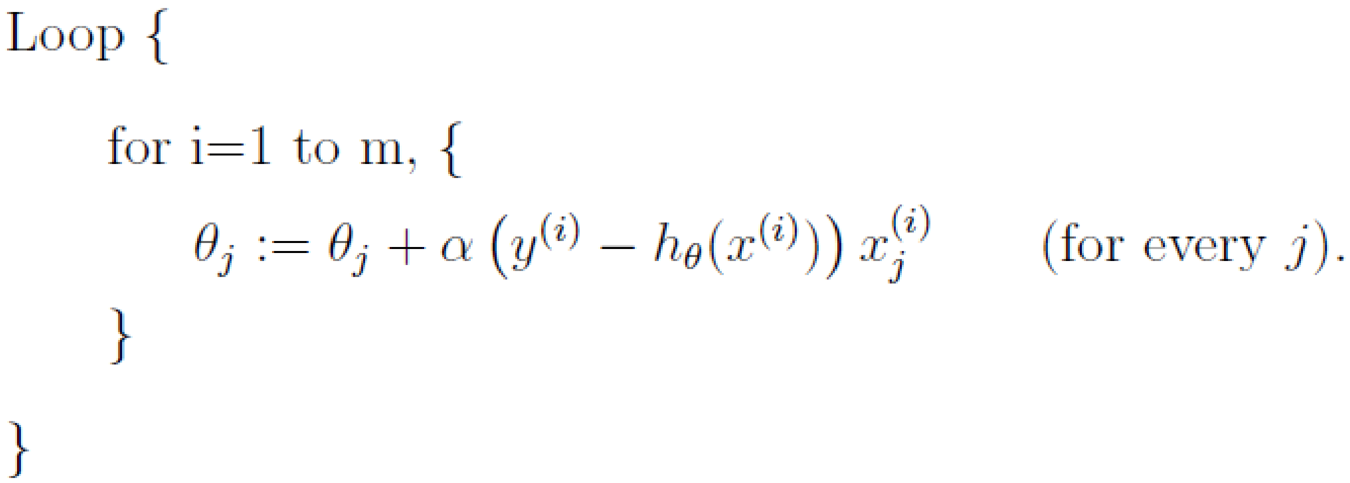


```
Repeat until convergence {
    $\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$    (for every $j$).
}
```

λw

Batch GD

- SGD can start making progress right away, and continues to make progress with each example it looks at.
- Often, SGD gets $\boldsymbol{\theta}$ "close" to the minimum much faster than batch gradient descent.

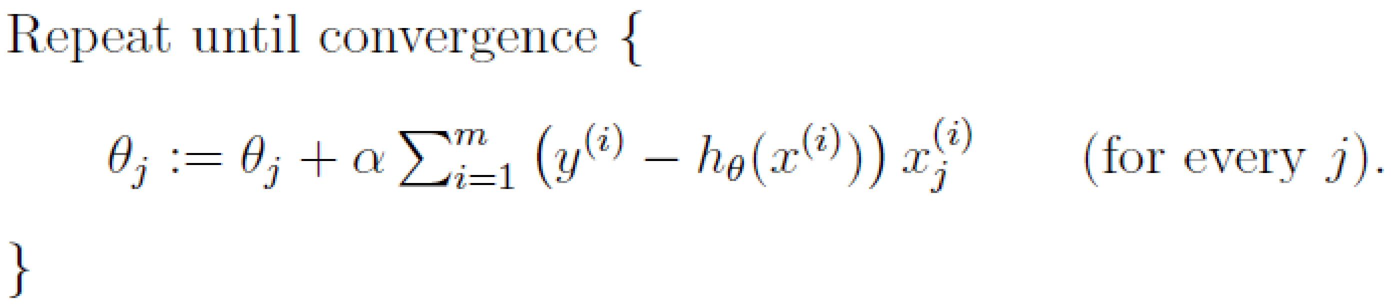


```
Loop {
    for i=1 to m, {
        $\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$    (for every $j$).
    }
}
```

SGD

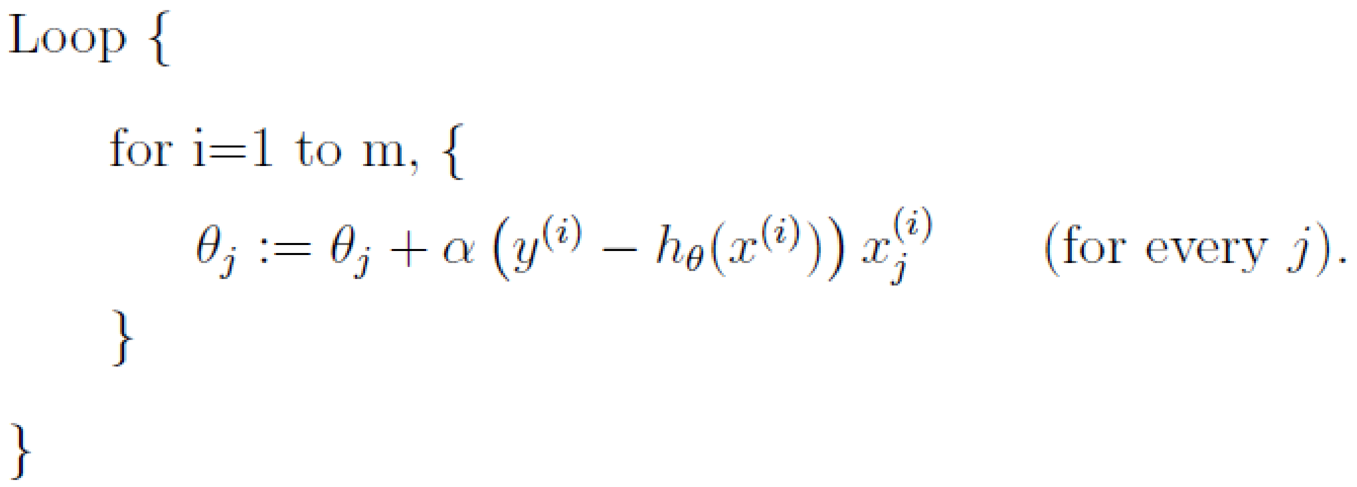
Batch vs Stochastic Gradient Descent

- SGD may never "converge" to the minimum, and the parameters $\boldsymbol{\theta}$ will keep oscillating around the minimum of $J(\boldsymbol{\theta})$;
- But in practice most of the values near the minimum will be reasonably good approximations to the true minimum.
- Therefore, when the training set is large, SGD is often preferred over batch gradient descent.



```
Repeat until convergence {
    $\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$    (for every $j$).
}
```

Batch GD

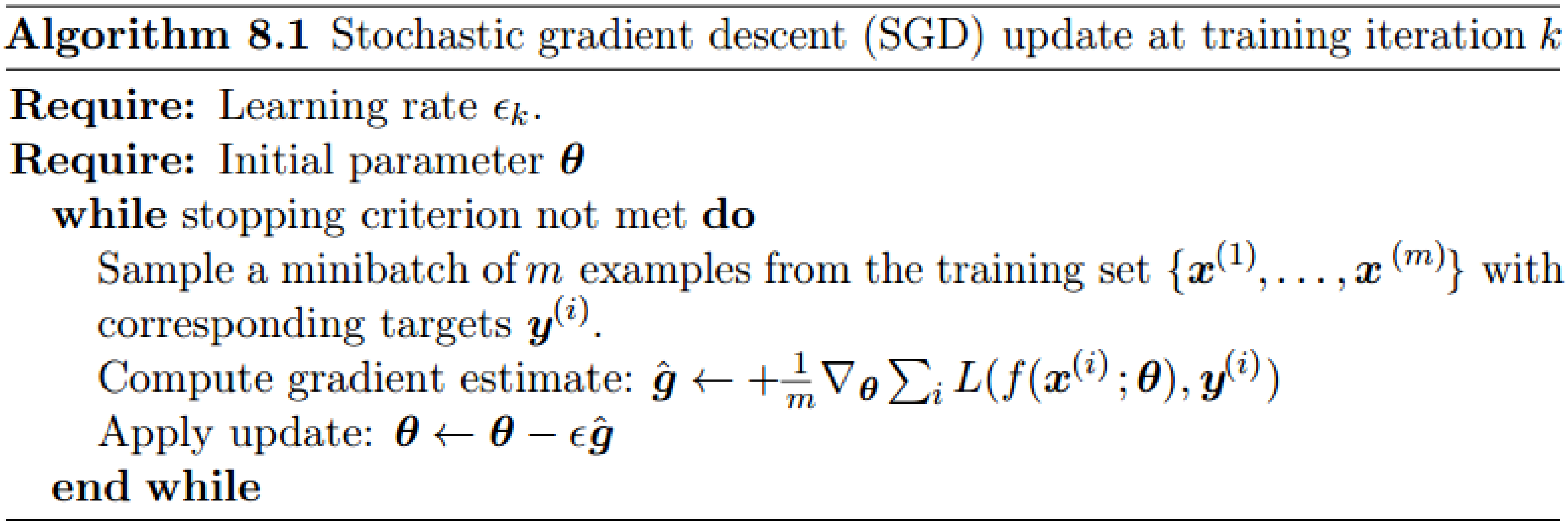


```
Loop {
    for i=1 to m, {
        $\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$    (for every $j$).
    }
}
```

SGD

Stochastic Gradient Descent

- Most algorithms (called *minibatch* or *minibatch stochastic* methods) fall somewhere in between, using more than one but less than all of the training examples.
- **Note**: Confusing again…… It is now common to simply call them *stochastic* methods.



```
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
```

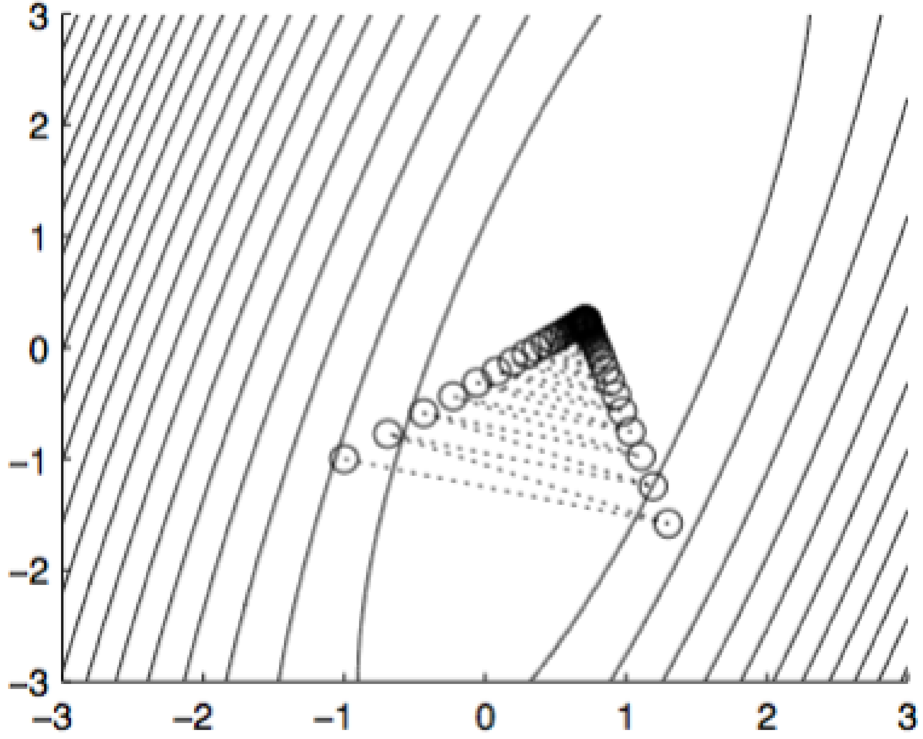
Accelerated SGD for Deep Learning

- **Polyak's Classical Momentum** [Polyak 1964]

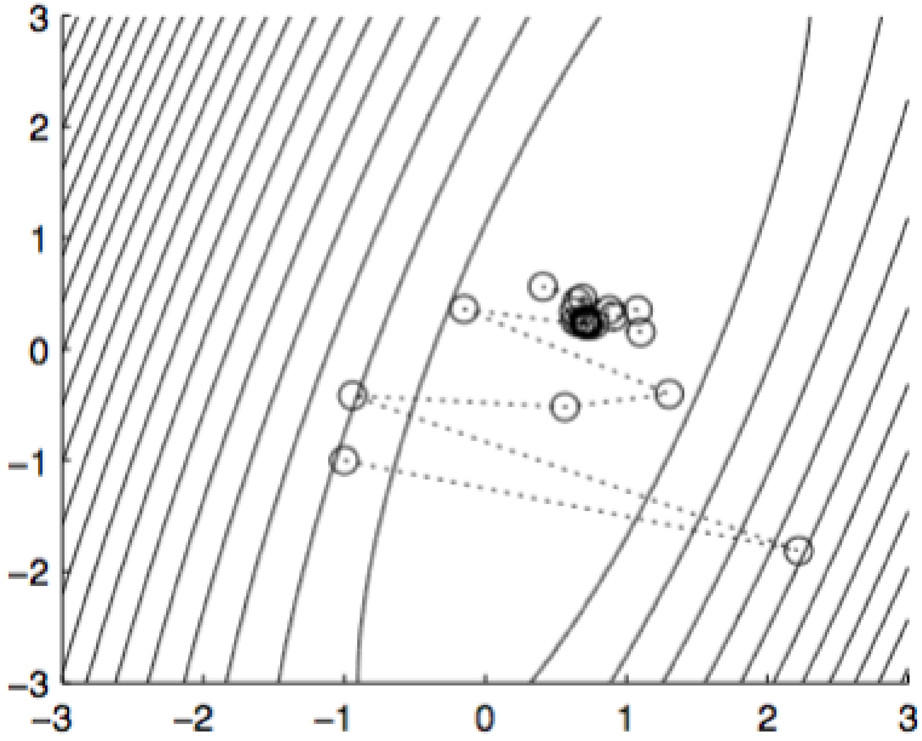
- **Nesterov's Momentum** [Nesterov 1983]

Polyak’s Classical Momentum [Polyak 1964]

- **Motivation**: Key problem of Gradient Descent is “zig-zagging”.



(a) Zig-zagging problem of GD.



(a) Trajectory of CM on same problem.

Polyak's Classical Momentum [Polyak 1964]

- The classical momentum (CM) accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.
- Letting η be the learning rate.

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1}$$

$$\boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \eta \nabla_{\boldsymbol{w}_t} f$$

- Velocity vector $\boldsymbol{v}_t$: a memory that accumulates the directions of reduction that were chosen in the previous t steps.
- The influence of $\boldsymbol{v}$ is controlled by the *momentum coefficient* $\mu \in [0,1]$. μ is usually slightly less than 1. When $\mu = 0$:

Polyak's Classical Momentum [Polyak 1964]

- The classical momentum (CM) accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.
- Letting η be the learning rate.

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1}$$

$$\boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \eta \nabla_{\boldsymbol{w}_t} f$$

- Velocity vector $\boldsymbol{v}_t$: a memory that accumulates the directions of reduction that were chosen in the previous t steps.
- The influence of $\boldsymbol{v}$ is controlled by the *momentum coefficient* $\mu \in [0,1]$. μ is usually slightly less than 1. When $\mu = 0$: it is just the Gradient Descent.

Polyak's Classical Momentum [Polyak 1964]

- **Exponential decay**

$$\boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \eta \nabla_{\boldsymbol{w}_t} f = \mu(\mu \boldsymbol{v}_{t-1} - \eta \nabla_{\boldsymbol{w}_{t-1}} f) - \eta \nabla_{\boldsymbol{w}_t} f$$

$$= \mu^{t+1} \boldsymbol{v}_0 - (\mu^t \eta \nabla_{\boldsymbol{w}_0} f + \cdots + \mu^1 \eta \nabla_{\boldsymbol{w}_{t-1}} f + \eta \nabla_{\boldsymbol{w}_t} f)$$

We know that $\lim\limits_{n \to \infty} (1 + \frac{1}{n})^n = e$ $\lim\limits_{n \to \infty} (1 - \frac{1}{n})^n = \frac{1}{e}$

Let $x = \frac{1}{n} \to 0$, we have $(1 - x)^{\frac{1}{x}} = \frac{1}{e}$

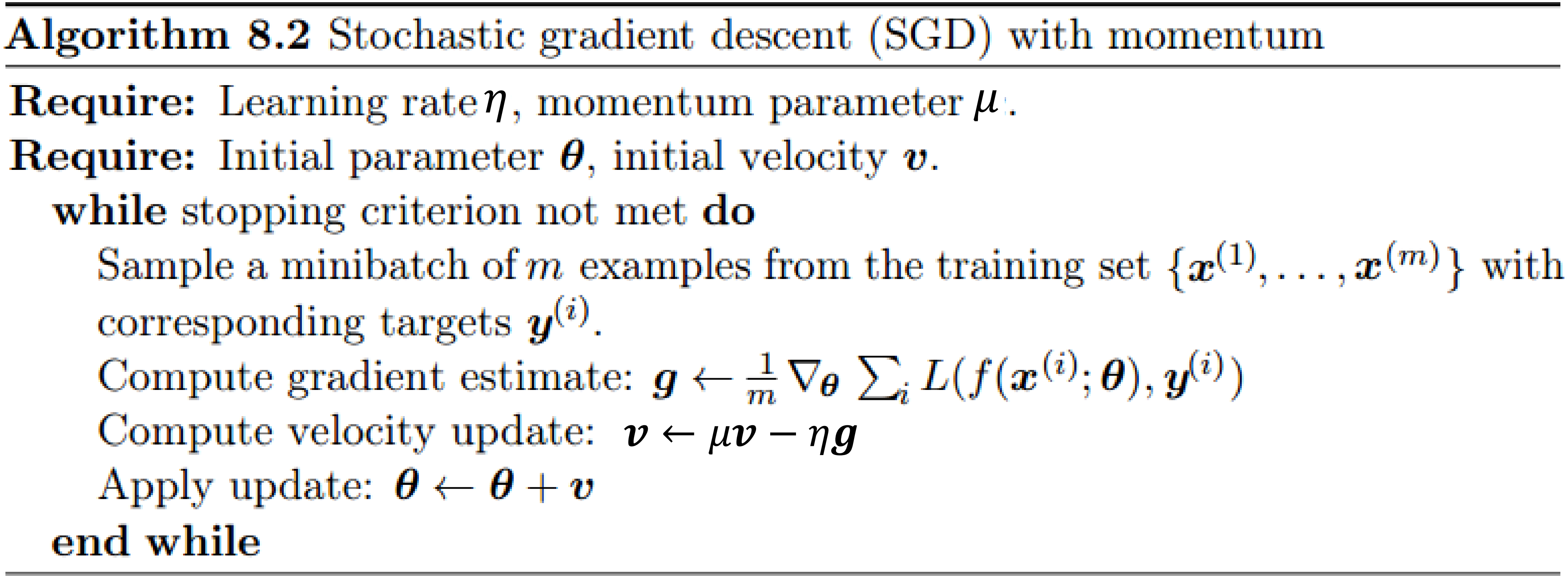
We 'ignore' the terms whose weights decay to less than $\frac{1}{e}$.

For example, if $\mu = 0.9$ ($x = 0.1$), to calculate $\boldsymbol{v}_{100}$, we only consider the last 10 steps (i.e., $\boldsymbol{v}_{99} \ldots \boldsymbol{v}_{90}$) as the valid memory.

Polyak's Classical Momentum [Polyak 1964]

- The SGD algorithm with momentum is given as follows.

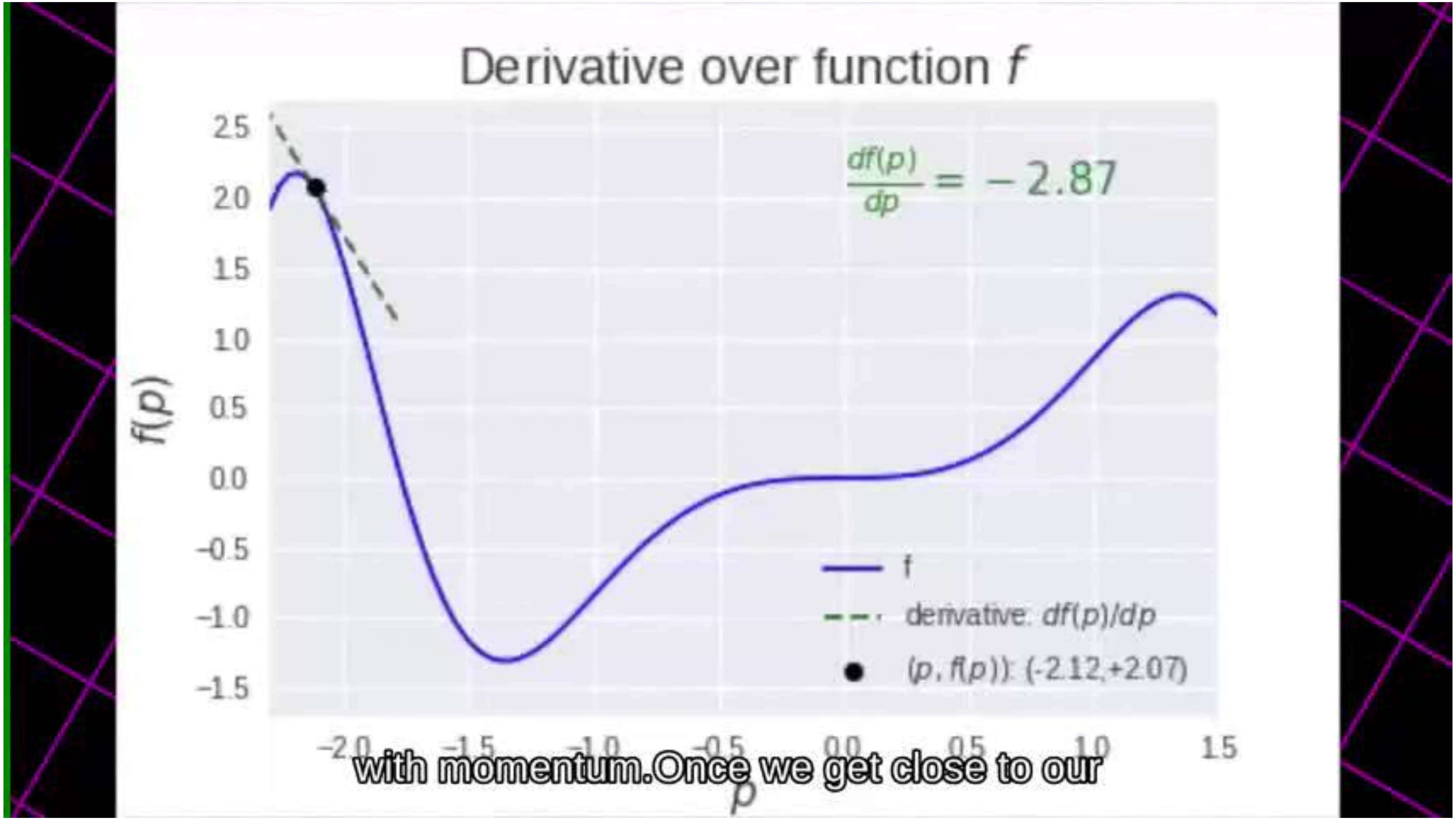
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1} \qquad \boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \eta \nabla_{\boldsymbol{w}_t} f$$



Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

```
Require: Learning rate $\eta$, momentum parameter $\mu$.
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: $\boldsymbol{v} \leftarrow \mu \boldsymbol{v} - \eta \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

Polyak's Classical Momentum [Polyak 1964]


Derivative over function f
$\frac{df(p)}{dp} = -2.87$
f(p)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
p
f
derivative: df(p)/dp
● (p, f(p)): (-2.12,+2.07)
with momentum.Once we get close to our

Nesterov's Accelerated Gradient [Nesterov 1983]

- The update equations of NAG are:

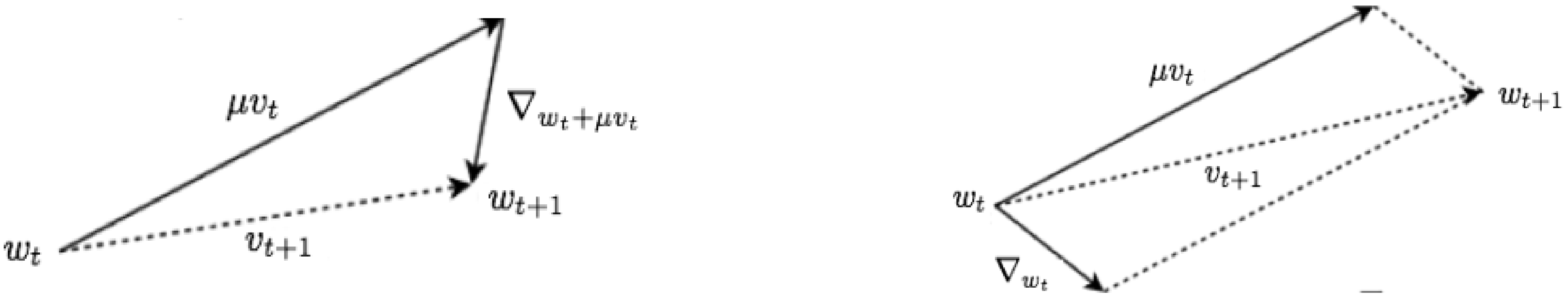
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1}$$

$$\boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \eta \nabla_{\boxed{\boldsymbol{w}_t + \mu \boldsymbol{v}_t}} f$$

CM

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1}$$

$$\boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \eta \nabla_{\boxed{\boldsymbol{w}_t}} f$$



NAG

CM

Illustration of the comparison between CM and NAG.

Nesterov’s Accelerated Gradient [Nesterov 1983]

- 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.

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1}$$

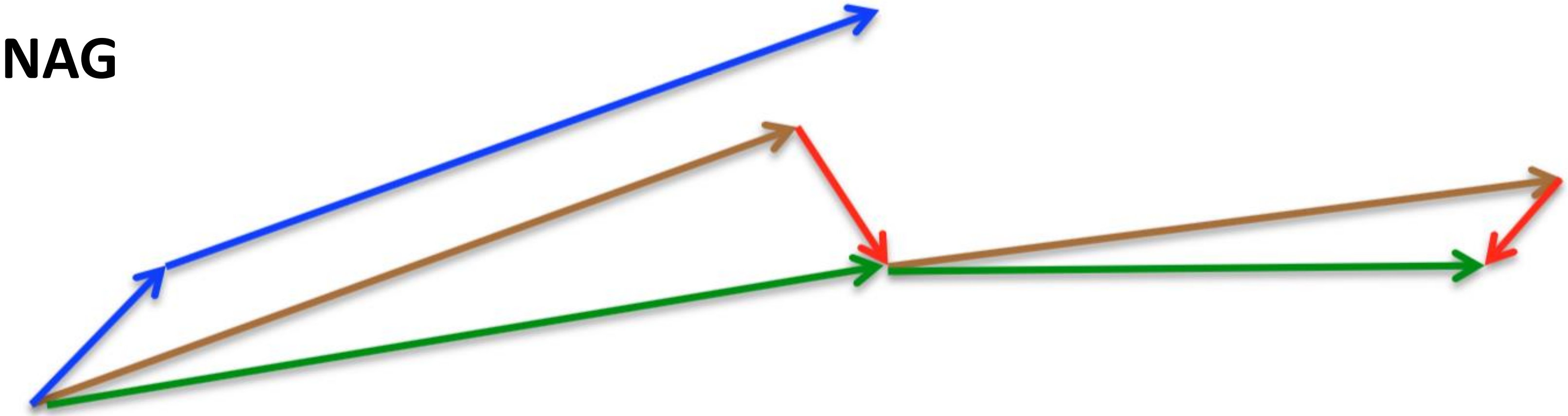
$$\boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \eta \nabla_{\boxed{\boldsymbol{w}_t + \mu \boldsymbol{v}_t}} f$$

NAG

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1}$$

$$\boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \eta \nabla_{\boxed{\boldsymbol{w}_t}} f$$

CM



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

blue vectors = standard momentum

Nesterov’s Accelerated Gradient [Nesterov 1983]

- The update equations of NAG are:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1}$$

$$\boldsymbol{v}_{t+1} = \mu\boldsymbol{v}_t - \eta\nabla_{\boxed{\boldsymbol{w}_t+\mu\boldsymbol{v}_t}} f$$

CM

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1}$$

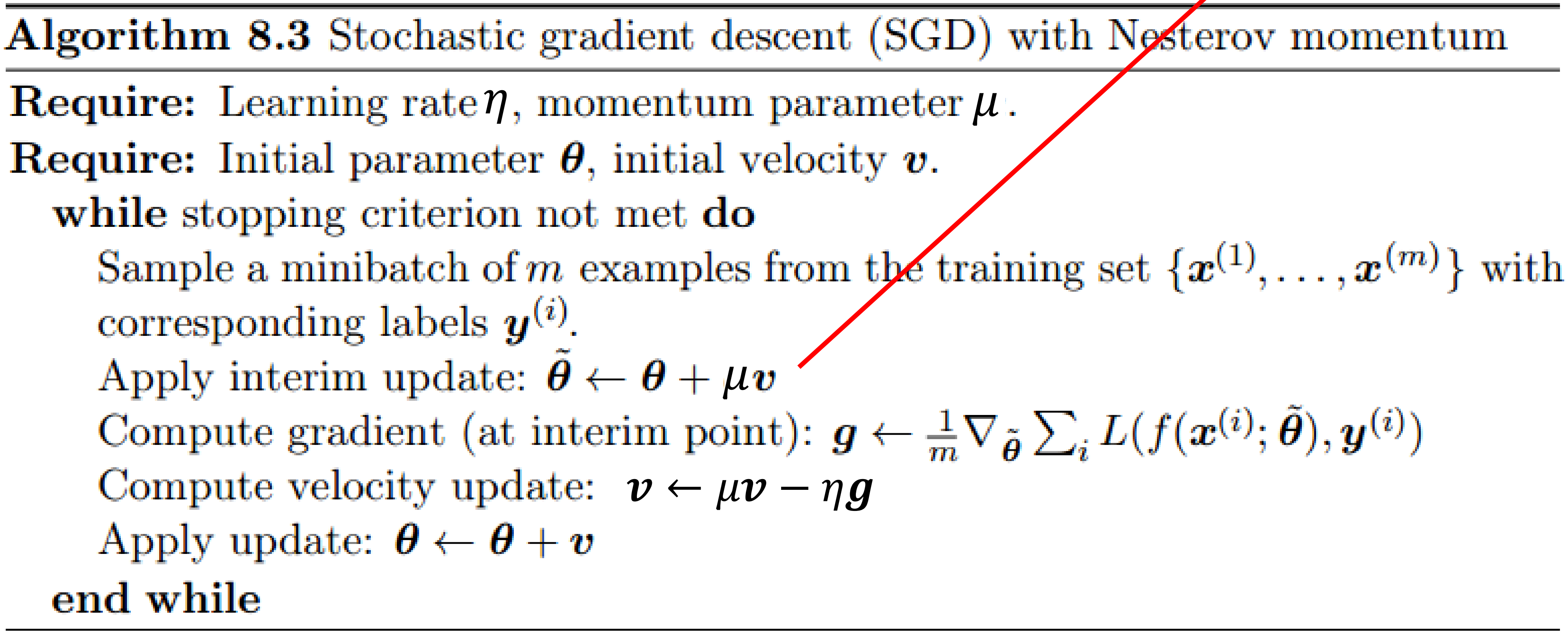
$$\boldsymbol{v}_{t+1} = \mu\boldsymbol{v}_t - \eta\nabla_{\boxed{\boldsymbol{w}_t}} f$$

- CM → inspecting the gradient at the current iterate of $\boldsymbol{w}_t$;
  - Faithfully trusts the current iterate;
- NAG → inspecting the gradient at $\boldsymbol{w}_t + \mu\boldsymbol{v}_t$.
  - Puts less faith into the current iterate and looks ahead in the direction suggested by the velocity vector.
- The small difference allows NAG to adapt faster and in a more stable way.

Nesterov’s Accelerated Gradient [Nesterov 1983]

- The complete Nesterov momentum algorithm is presented as follows.

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{v}_{t+1} \qquad \boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \eta \nabla_{\boxed{\boldsymbol{w}_t + \mu \boldsymbol{v}_t}} f$$



Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

```
Require: Learning rate $\eta$, momentum parameter $\mu$.
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: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \mu \boldsymbol{v}$
    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: $\boldsymbol{v} \leftarrow \mu \boldsymbol{v} - \eta \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

Algorithms with Adaptive Learning Rates

- Researchers have long realized that the learning rate is the most difficult hyperparameter to set because it has a significant impact on model performance.

- Recently, a number of methods have been introduced that adapt the learning rates of model parameters.
  - Adagrad
  - RMSProp
  - Adam

Adagrad

Adagrad (Duchi et al, COLT 2010) uses a different learning rate for every parameter θ_i at every time step t. It 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

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i})$$

The SGD update

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t.

Weakness: Since every added term is positive, the accumulated sum keeps growing which in turn causes the learning rate to shrink and eventually become infinitely small.

Adadelta & RMSprop

Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size *w*.

Via a decaying mechanism, $E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$

RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in his Coursera Class. RMSprop and Adadelta have both been developed independently around the same time to resolve Adagrad’s radically diminishing learning rates.

衰减速率：可调超参

$$E[g^2]_t = \boxed{0.9} E[g^2]_{t-1} + 0.1 g_t^2$$

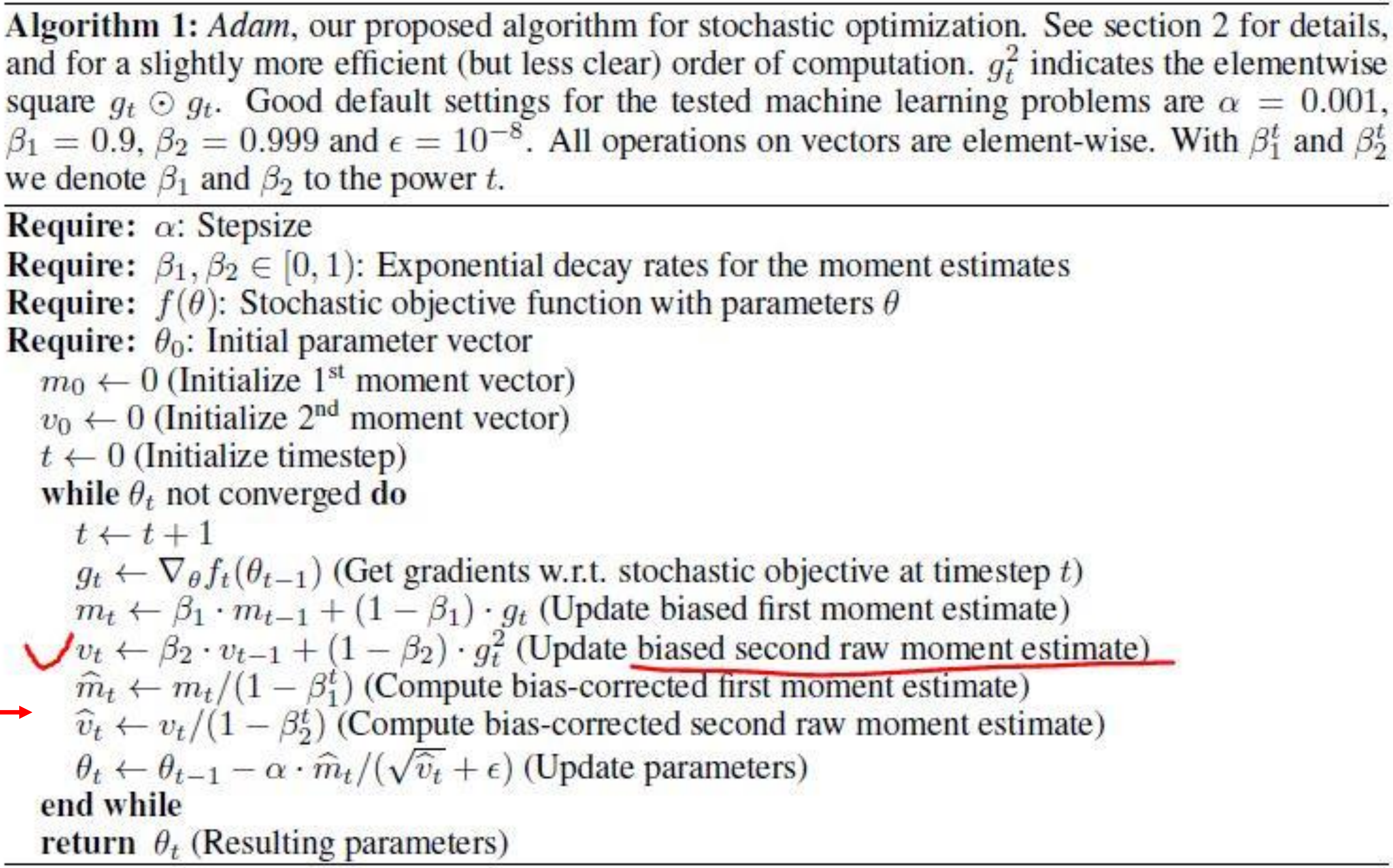
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

采用指数衰减平均，以丢弃遥远过去的历史。

Adam

- Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter.
- Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled “Adam: A Method for Stochastic Optimization“.
  1. Adam stores an exponentially decaying average of past squared gradients $\boldsymbol{v}_t$ (variance) like Adadelta and RMSprop.
  2. Adam also keeps an exponentially decaying average of past gradients $\boldsymbol{m}_t$ (mean), similar to momentum.

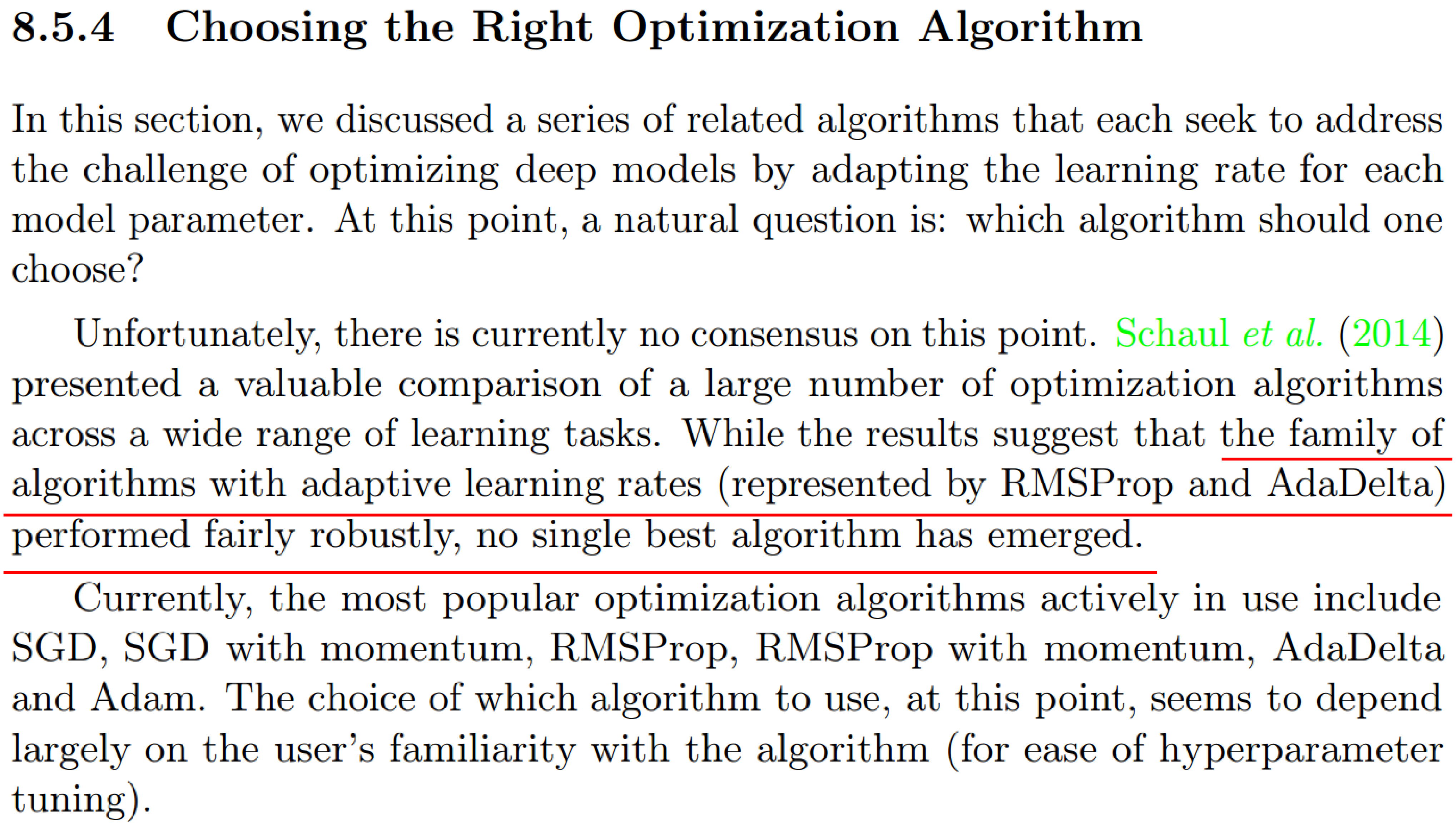
Adam



Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  ✓ $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

As $\boldsymbol{m}_t, \boldsymbol{v}_t$ are initialized as 0, they are biased towards 0, especially when $\beta_1 \rightarrow 1, \beta_2 \rightarrow 1$. →



8.5.4 Choosing the Right Optimization Algorithm

In this section, we discussed a series of related algorithms that each seek to address the challenge of optimizing deep models by adapting the learning rate for each model parameter. At this point, a natural question is: which algorithm should one choose?

Unfortunately, there is currently no consensus on this point. Schaul *et al.* (2014) presented a valuable comparison of a large number of optimization algorithms across a wide range of learning tasks. While the results suggest that the family of algorithms with adaptive learning rates (represented by RMSProp and AdaDelta) performed fairly robustly, no single best algorithm has emerged.

Currently, the most popular optimization algorithms actively in use include SGD, SGD with momentum, RMSProp, RMSProp with momentum, AdaDelta and Adam. The choice of which algorithm to use, at this point, seems to depend largely on the user's familiarity with the algorithm (for ease of hyperparameter tuning).

Summary

- **Deterministic Optimization**
  - The data for the given problem are known accurately.
  - First order method: e.g., Gradient Descent.
  - Second order method: e.g., Newton’s method, Quasi Newton’s Methods (BFGS).
- **Stochastic Optimization**
  - Using several samples of the training examples (minibatch).
  - SGD
  - SGD+Accelaration
    - Polyak’s Classical Momentum
    - Nesterov’s Momentum
    - Agrad/Adadelta/RMSprop/Adam