# Optimization for Training I

First-Order Methods Training algorithm

# OPTIMIZATION METHODS

**Topics:** Types of optimization methods.

- Practical optimization methods breakdown into two categories:

  1. First-order methods
  2. Second-order methods

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{a}) + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{a})(\boldsymbol{\theta} - \boldsymbol{a}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{a})^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{a})$$

- Today we will focus on first-order methods

# STOCHASTIC GRADIENT DESCENT

- Vanilla SGD is still probably the most popular method of training deep learning models.
- (+) Works on a single example or a mini-batch / ( - ) Can converge slowly.

---

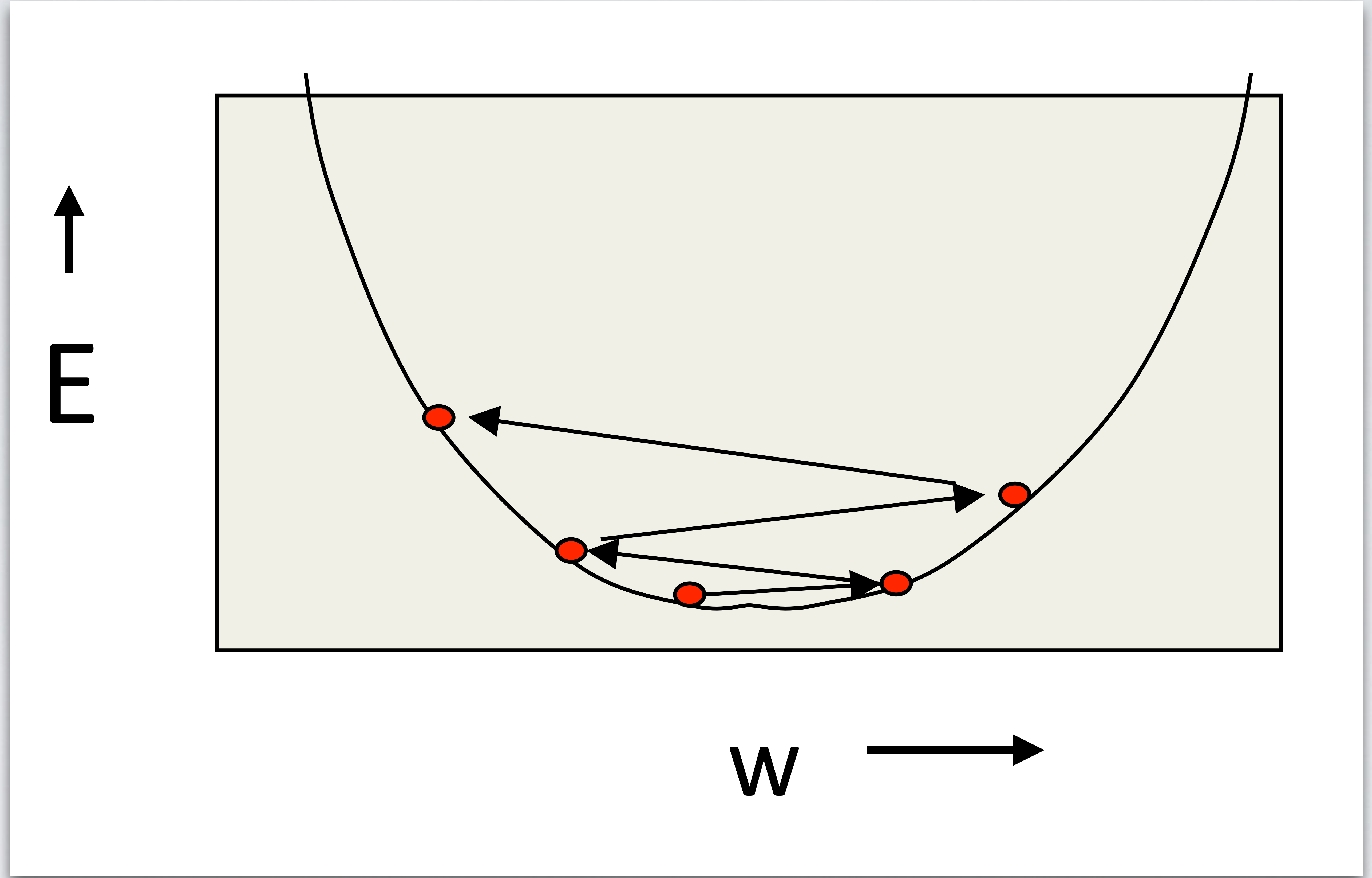
**Algorithm 1** Stochastic gradient descent (SGD) update at training iteration $k$

---

**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: $\hat{\boldsymbol{h}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{h}}$
  **end while**

---

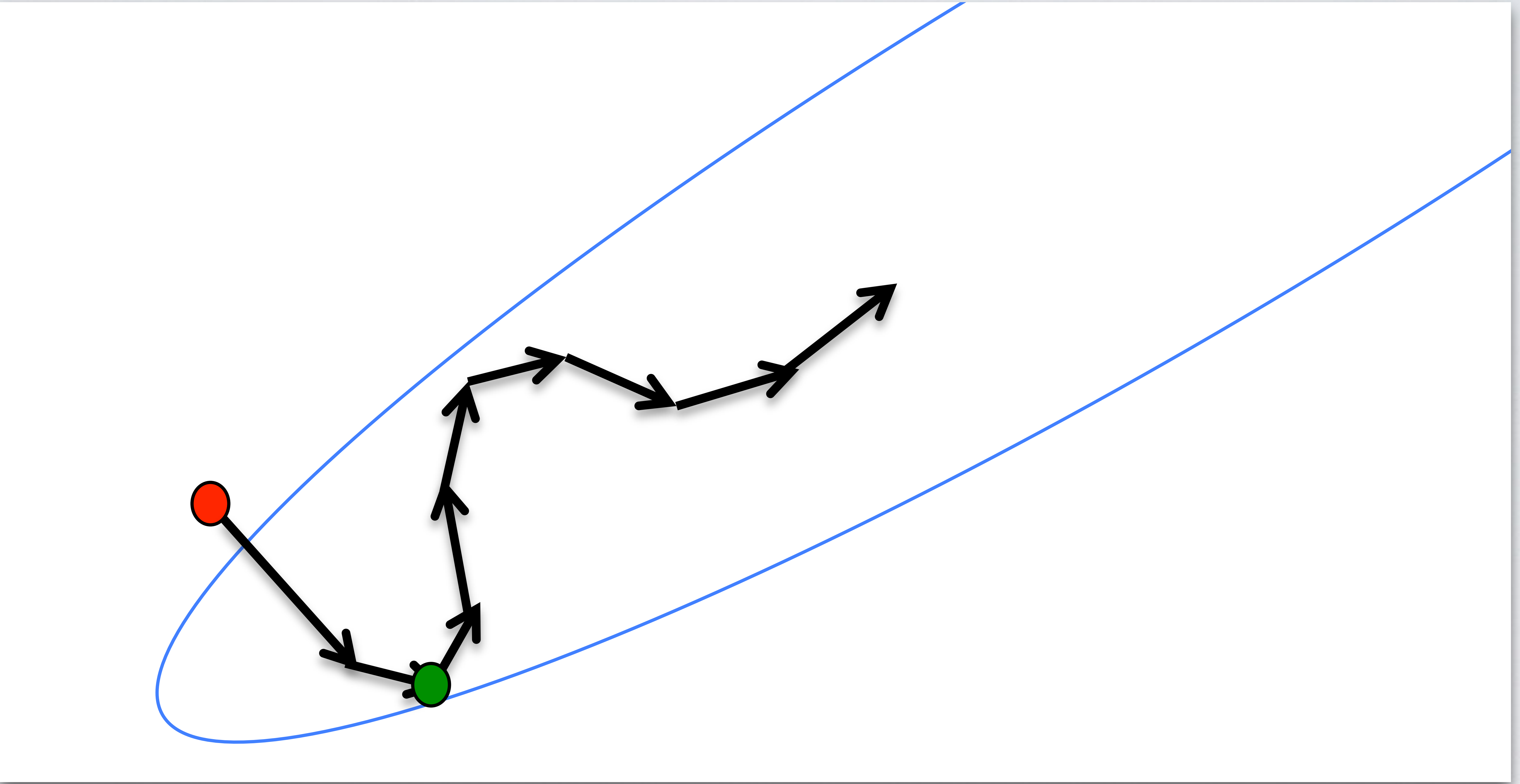
# STOCHASTIC GRADIENT DESCENT



# MOMENTUM METHOD

- Designed to accelerate learning, especially with small consistent gradients.
- Inspired from physical interpretation of the optimization process: Imagine you have a small ball rolling on a surface defined by the loss function.

# MOMENTUM METHOD



# MOMENTUM METHOD

**Algorithm 1** 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: $\boldsymbol{h} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{h}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  **end while**

# NESTEROV MOMENTUM

- Sutskever et al (ICML 2013) presented a modified version of momentum they called Nesterov momentum.
- **Basic idea**: apply the gradient "correction" after the velocity term is applied.

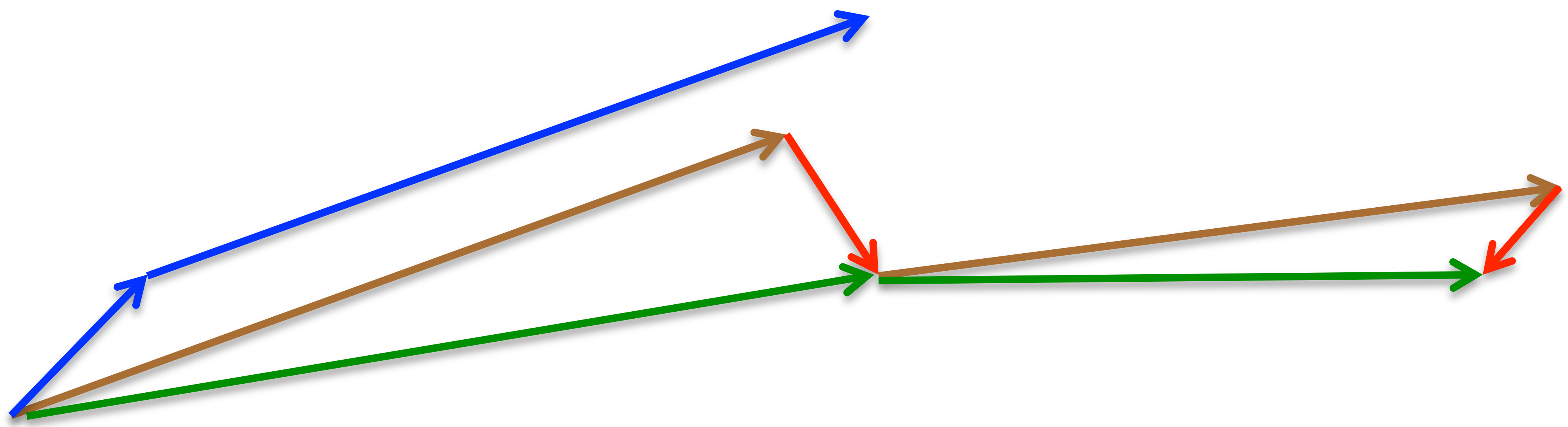
# NESTEROV MOMENTUM

**Algorithm 1** Stochastic gradient descent (SGD) with Nesterov 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 labels $\boldsymbol{y}^{(i)}$.
  Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
  Compute gradient (at interim point): $\boldsymbol{h} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{h}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
**end while**

# NESTEROV MOMENTUM

- 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

Slide from Hinton's Coursera course.

# ADAGRAD

- Adagrad (Duchi et al, COLT 2010) is a method of adapting the learning rate.

- (+) Can adapt independent learning rates for all parameters
- ( - ) Accumulating gradients from the start makes later learning very slow.

# ADAGRAD

**Algorithm 1** 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: $\boldsymbol{h} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{h} \odot \boldsymbol{h}$
Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{h}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
**end while**

# RMSPROP

- Modifies AdaGrad to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average.
- Compared to AdaGrad, the use of the moving average introduces a new hyperparameter that controls the length scale of the moving average.
- Empirically, RMSProp has been shown to be an effective and practical optimization algorithm for deep neural networks.

# RMSPROP

**Algorithm 1** 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: $\boldsymbol{h} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{h} \odot \boldsymbol{h}$
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{h}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied elem-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
**end while**

# RMSPROP+MOMENTUM

**Algorithm 1** 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: $\boldsymbol{h} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{h} \odot \boldsymbol{h}$

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{h}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

**end while**

# ADAM

- ``Adam'' derives from the phrase ``adaptive moments.''
- Variant of RMSProp + momentum with a few important distinctions:
    1. Momentum is incorporated directly as an estimate of the first order moment (with exponential weighting) of the gradient.
    2. Includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin.
- To date, Adam has largely become the default optimization algorithm for training deep learning systems.

# ADAM:

**Algorithm 1** 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. (Suggestion: $10^{-8}$)
**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 $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient: $\boldsymbol{h} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  $t \leftarrow t + 1$
  Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{h}$
  Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{h} \odot \boldsymbol{h}$
  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**