Optimizers

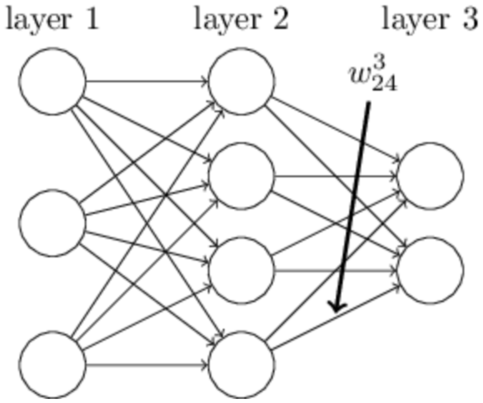
Review. Backpropagation

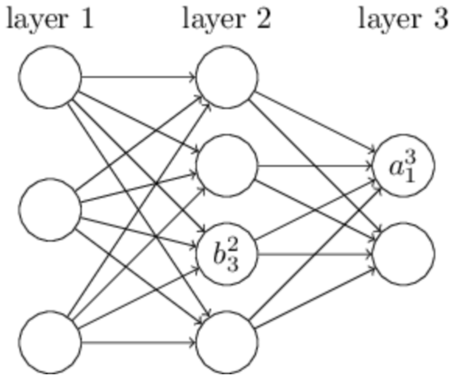
- Output activation and loss function

Task	# of output nodes	Output activation	Loss function
Binary classification	1	Sigmoid $\hat{y} = 1/(1 + \exp(-z))$	Binary cross-entropy $L(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$
Multiclass classification	k	Softmax $\hat{y}_i = \frac{\exp\{z_i\}}{\sum_{i=1}^{k} \exp\{z_i\}}$	Cross-entropy $L(y, \hat{y}) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$
Regression	k	None $\hat{y}_i = z_i$	Mean squared error $L(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^{k} (\hat{y}_i - y_i)^2$

Review. Backpropagation

- Refer to Chap. 2 in Nielsen's Neural Networks and Deep Learning
  - Notations: $w_{jk}^l, b_j^l, a_j^l, z_j^l, \delta_j^l = \frac{\delta C}{\delta z_j^l}$
  - The Hadamard product $\odot$
  - Four fundamental equations behind backpropagation
    - (BP1) $\delta^L = \nabla_a C \odot \sigma'(z^L)$
    - (BP2) $\delta^l = \left(w^{l+1}\right)^T \delta^{l+1} \odot \sigma'\left(z^l\right)$
    - (BP3) $\frac{\partial C}{\partial b_j^l} = \delta_j^l$
    - (BP4) $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$





Review. Backpropagation

- (BP1) $\delta^L = \nabla_a C \odot \sigma'(z^L)$ for multiclass classification

$$\frac{\delta C}{\delta z_j^L} = \sum_{i=1}^{k} \frac{\delta C}{\delta \hat{y}_i} \frac{\delta \hat{y}_i}{\delta z_j} = \hat{y}_j - y_j$$

$$\frac{\delta C}{\delta \hat{y}_i} = -\frac{y_i}{\hat{y}_i} \qquad \frac{\delta \hat{y}_i}{\delta z_j} = \hat{y}_i\left(I_{ij} - \hat{y}_j\right)$$

Review. Backpropagation

- Refer to [Chap. 2 in Nielsen's Neural Networks and Deep Learning]

1. **Input x:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

Basic Gradient Descent Algorithms

- Batch gradient descent

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ
1: **while** stopping criteria not met **do**
2: Compute gradient estimate over N examples:
3: $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$
4: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
5: **end while**

Basic Gradient Descent Algorithms

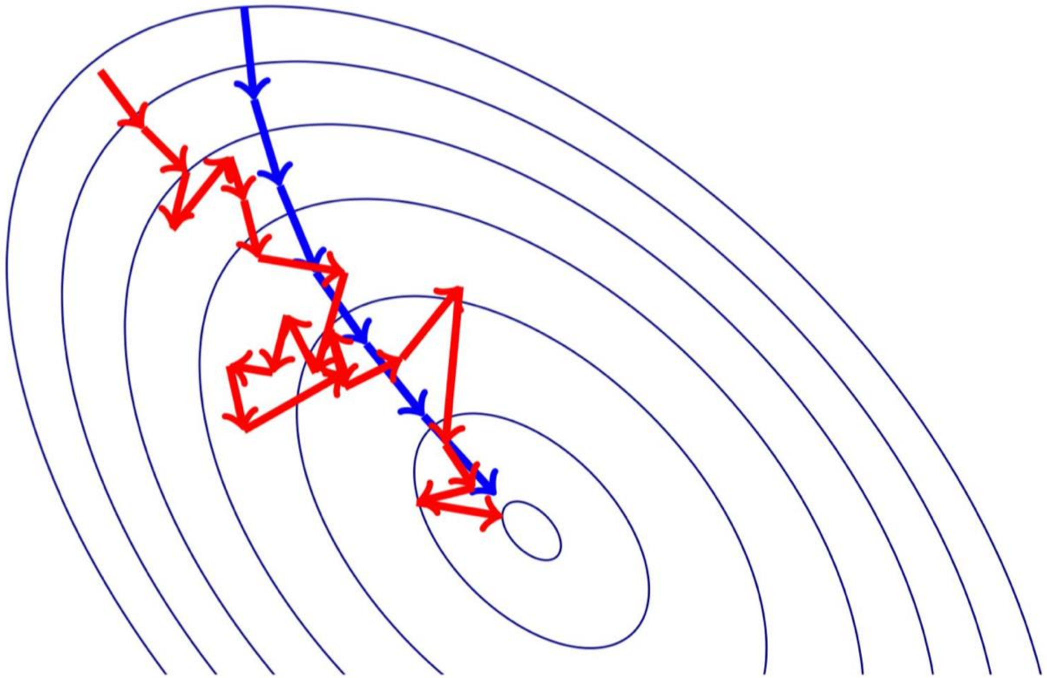
- Stochastic gradient descent

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  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: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

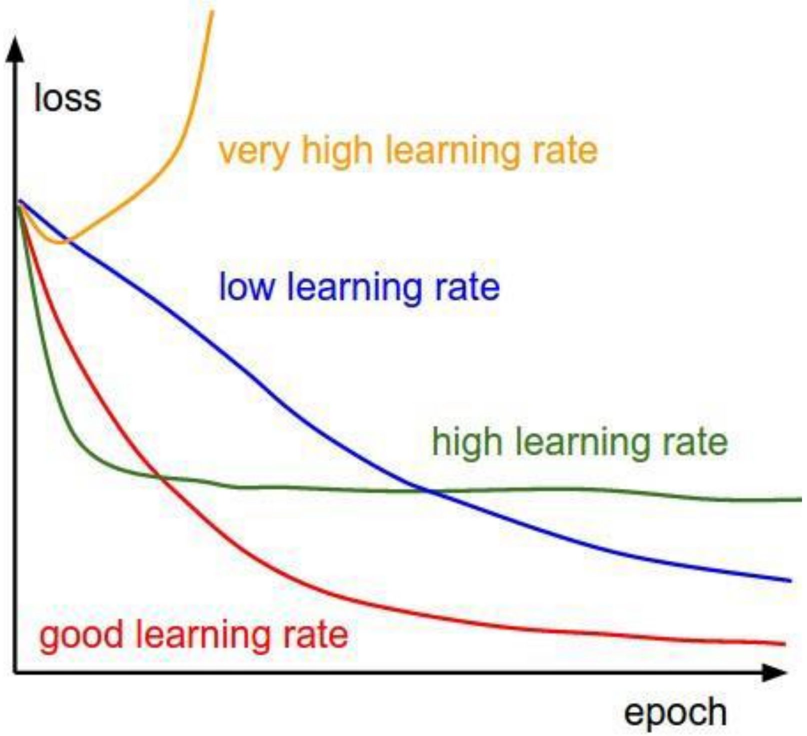
Basic Gradient Descent Algorithms

- Batch gradient descent VS Stochastic gradient descent



Basic Gradient Descent Algorithms

- How to determine the learning rate?
  - If it is too large, the objective value may explode.
  - If it is too small, a lot of iterations are needed.



Basic Gradient Descent Algorithms

- Momentum: accumulates an exponentially decaying moving average of past gradients and continues to move in that direction

previous direction
(accumulation of past gradients)

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

current direction

current gradient

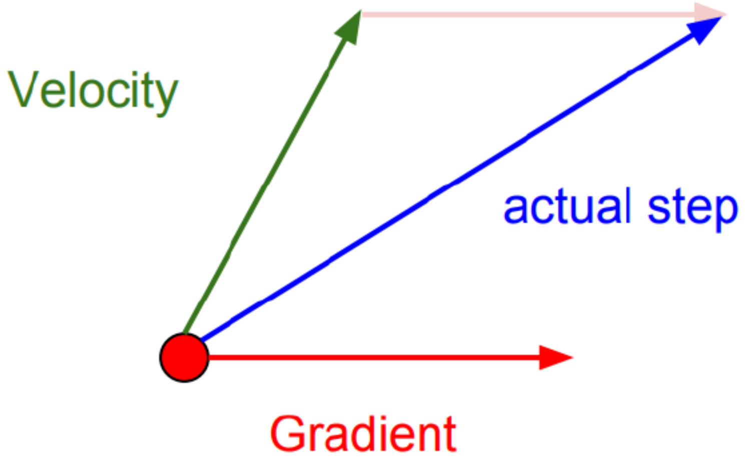


Figure Credit: Prof. Kang (SKKU)

Basic Gradient Descent Algorithms

- Stochastic gradient descent with momentum

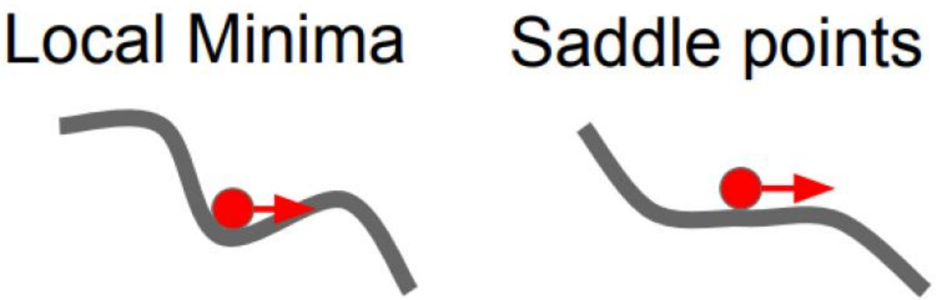
Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

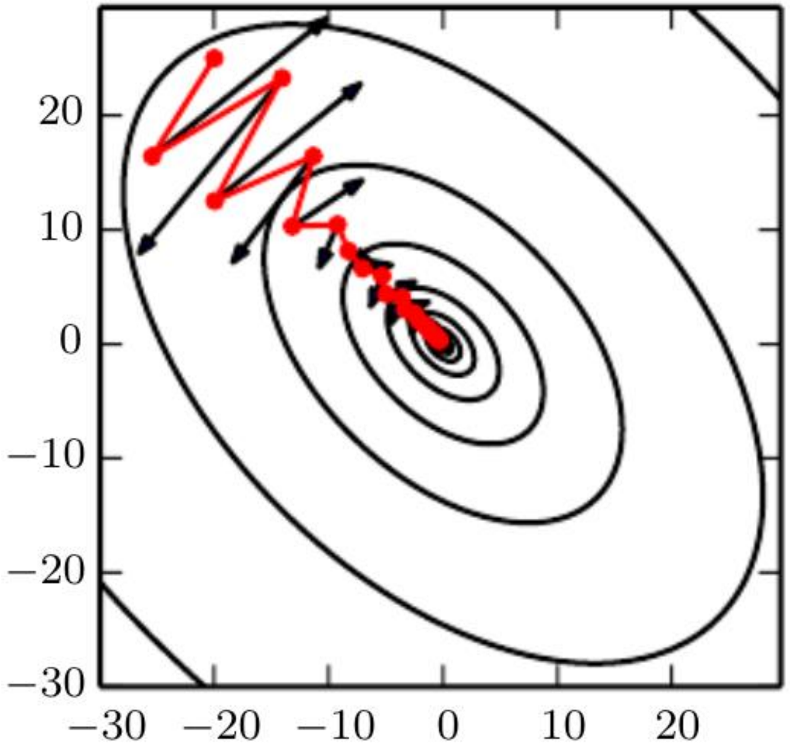
---

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

Basic Gradient Descent Algorithms

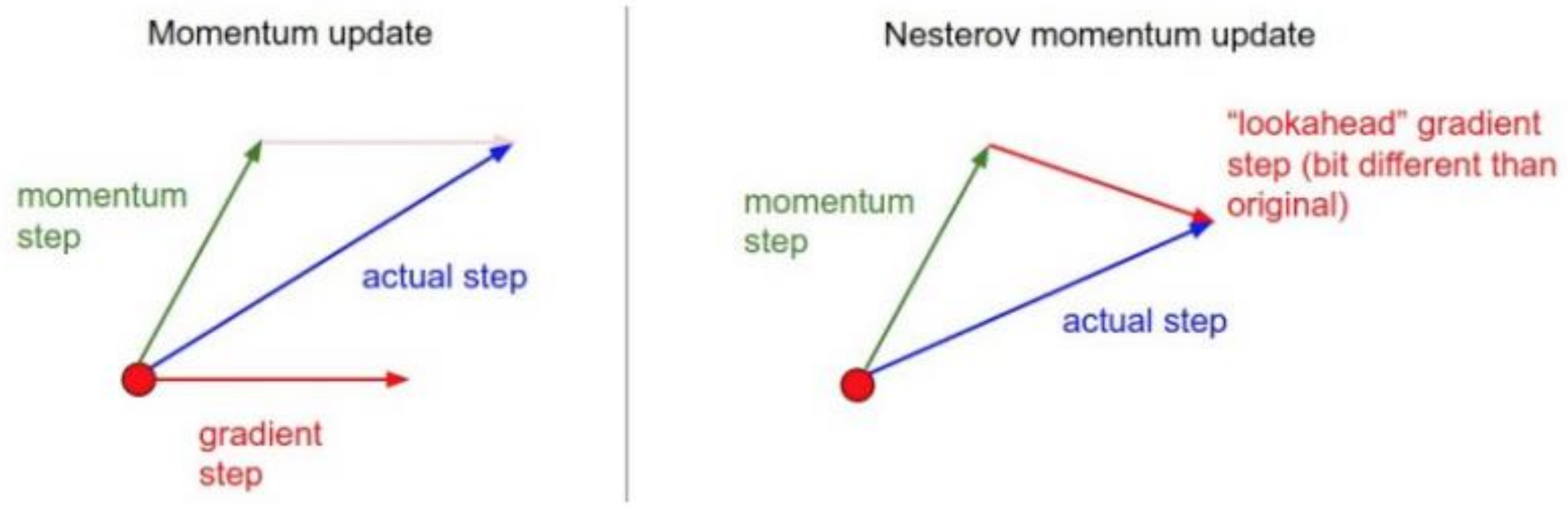
- Stochastic gradient descent with momentum





Basic Gradient Descent Algorithms

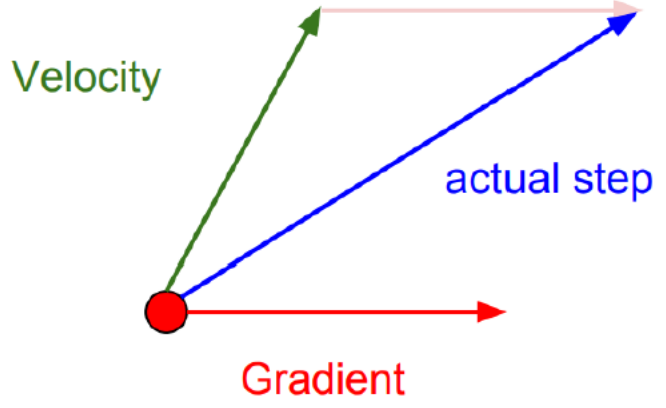
- Nesterov Momentum: the gradient is evaluated after the current velocity is applied.



Basic Gradient Descent Algorithms

Momentum

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

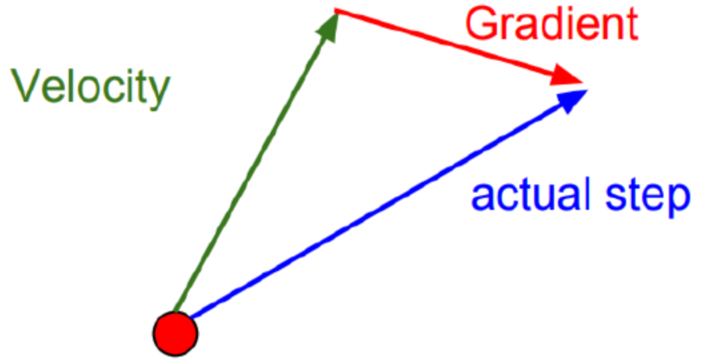


current direction

previous direction (accumulation of past gradients)

current gradient

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



Nesterov Momentum

Figure Credit: Prof. Kang (SKKU)

Basic Gradient Descent Algorithms

- Stochastic gradient descent with Nesterov's momentum

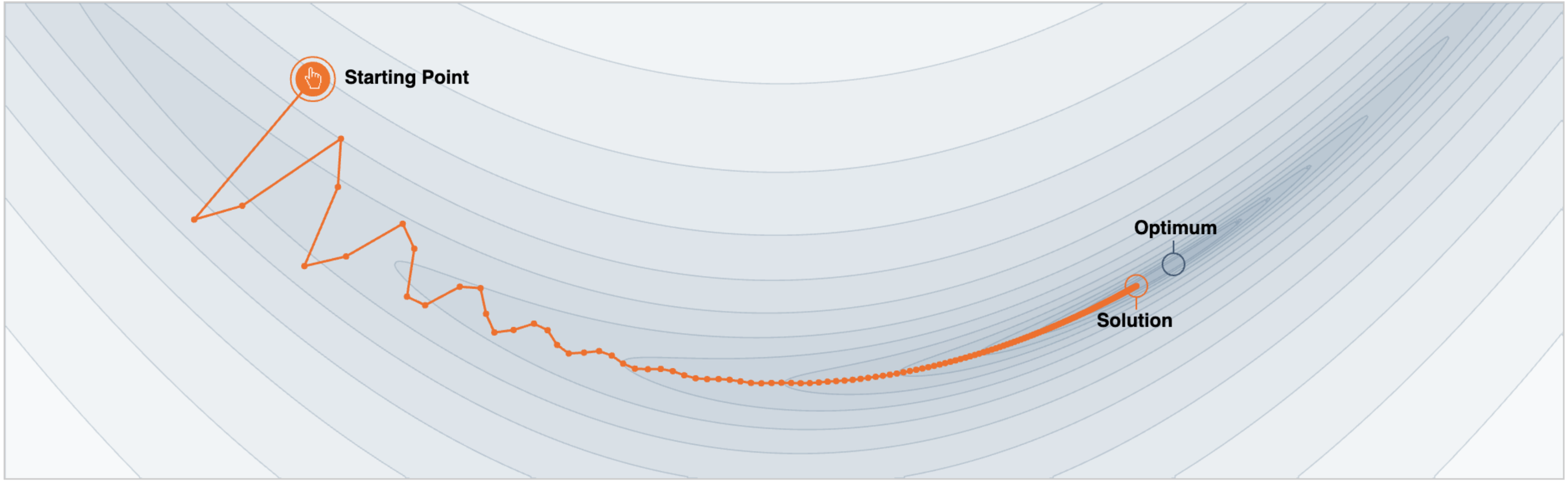
Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

---

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

Basic Gradient Descent Algorithms

Why Momentum Really Works



Step-size α = 0.02

0 0.003 0.006

Momentum β = 0.99

0.00 0.500 0.990

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

https://distill.pub/2017/momentum/

Algorithms with Adaptive Learning Rates

- The learning rate significantly affects the performance.
- How to determine the learning rate?

- Adaptive learning rate
  - To use a separate learning rate for each parameter and automatically adapt these learning rates throughout the course of learning

Algorithms with Adaptive Learning Rates

- AdaGrad (2011)
  - Large gradients → small update, small gradients → large update
  - Code snippet

```
# Assume the gradient dx and parameter vector x
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

  - Cache keeps growing during training → learning rate shrinks too quickly

Algorithms with Adaptive Learning Rates

- AdaGrad (2011)

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 $\boldsymbol{r} = \boldsymbol{0}$
  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: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    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

Algorithms with Adaptive Learning Rates

- RMSProp (2012)
  - Extension of AdaGrad which solves AdaGrad's aggressive, monotonically decreasing learning rate problem
  - The gradient accumulation → exponentially weighted moving average

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g} \quad \blacktriangleright \quad \boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$$

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

Algorithms with Adaptive Learning Rates

- RMSProp (2012)

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

Algorithms with Adaptive Learning Rates

- Adam (2015)
  - Adaptive moment estimation
  - It uses the first moment and the second moment.
  - A variant on the combination of RMSProp and momentum

```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

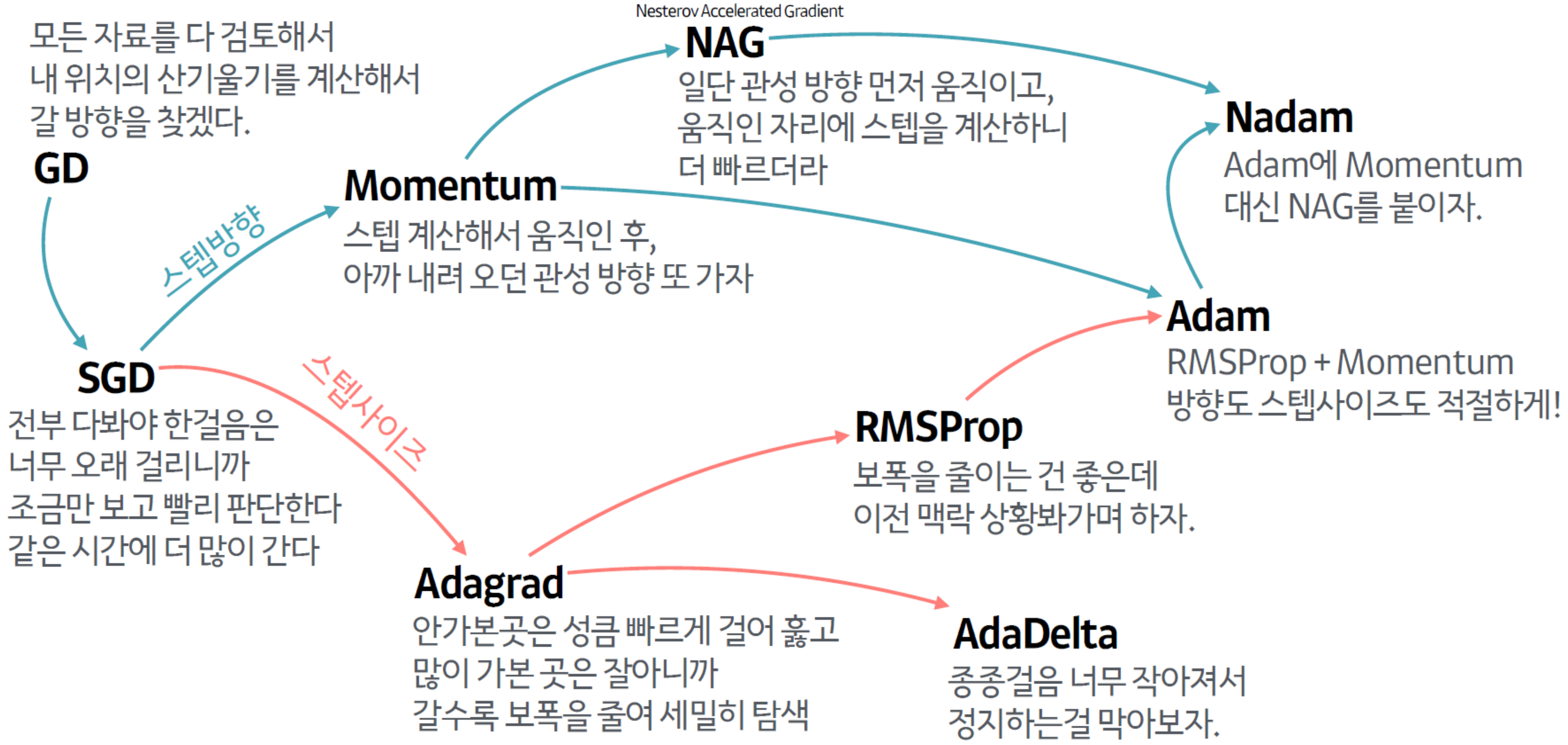
Algorithms with Adaptive Learning Rates

- Adam (2015)

Algorithm 8.7 The Adam algorithm

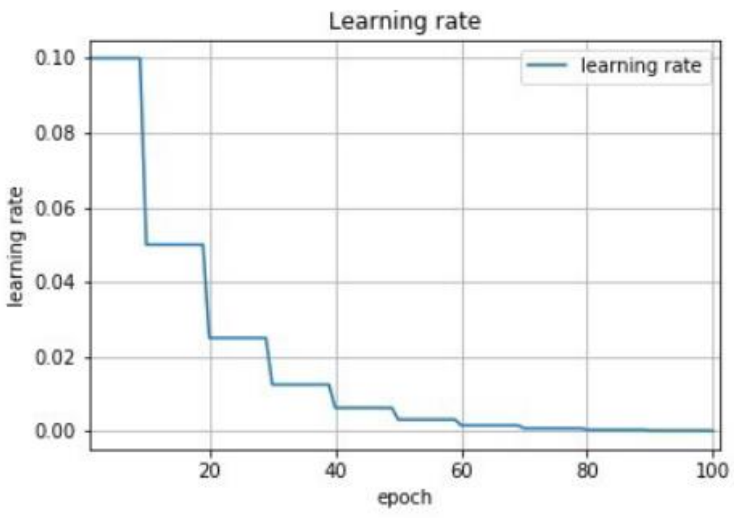
Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
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

And Many More

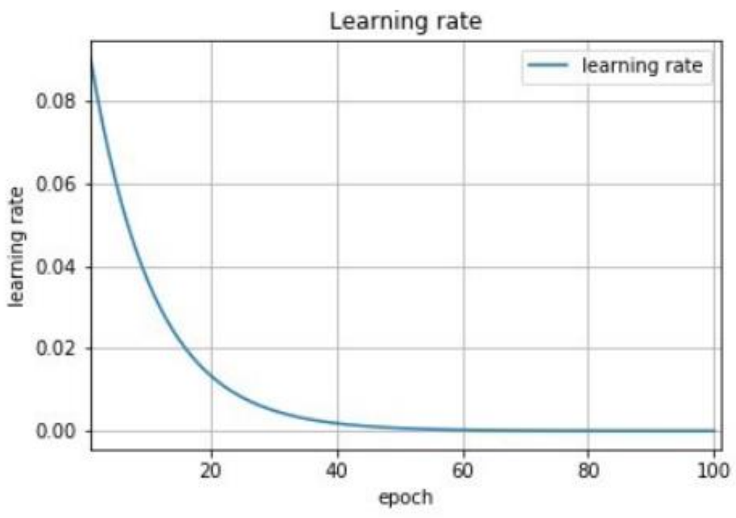


Learning Rate Scheduling

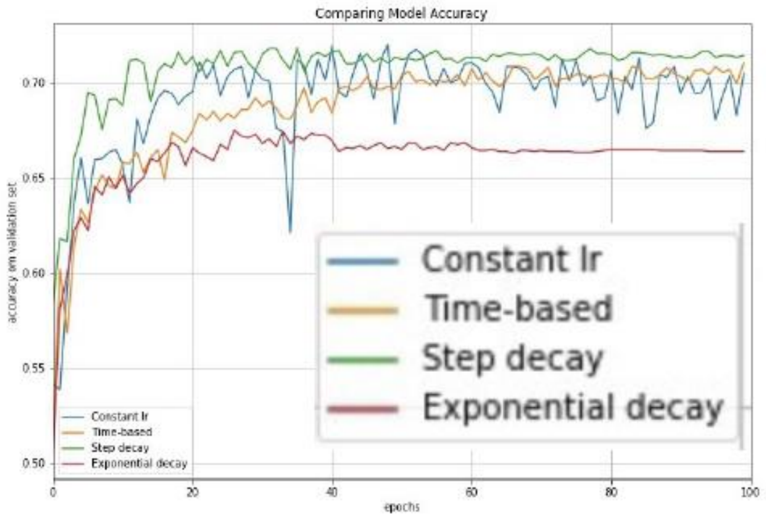
- Learning rate decaying
  - Constant learning rate often prevents convergence.



Step decay

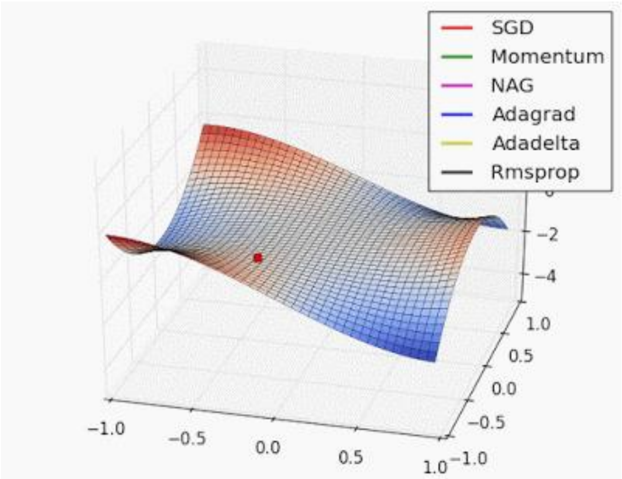


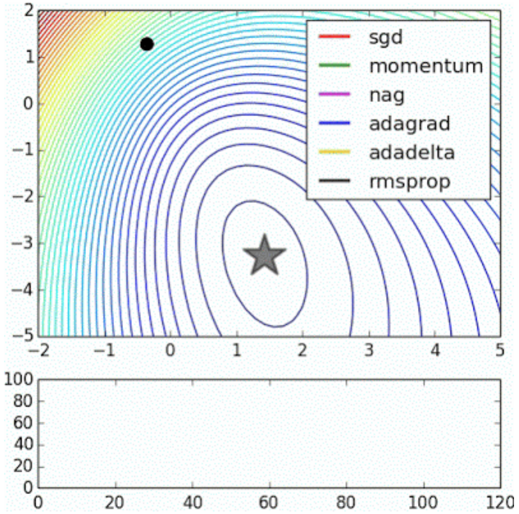
Exponential decay

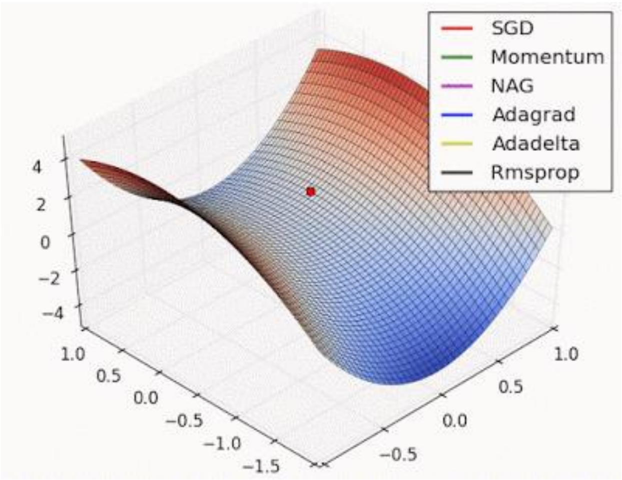


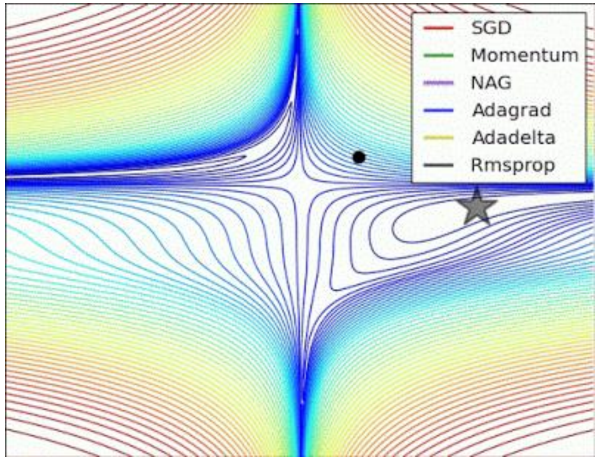
Test accuracy

Animations for Optimization Algorithms









Reading assignments

- “Understanding deep learning”
  - Chapter 6
- “Dive into deep learning”
  - Section 12