Optimization by gradient methods

COMS 4771 Fall 2025

Unconstrained optimization problems

Common form of optimization problem in machine learning:

$$\min_{w \in \mathbb{R}^d} \quad J(w)$$

We would like an algorithm that, given the objective function J, finds particular setting of w so that $J(w)$ is as small as possible

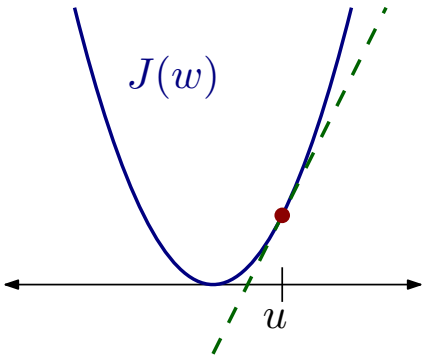
- What does it mean to be "given J"?
- What types of objective functions can we hope to minimize?

Review of multivariate differential calculus

A function $J\colon \mathbb{R}^d \to \mathbb{R}$ is differentiable if, for every $u \in \mathbb{R}^d$, there is an affine function $A\colon \mathbb{R}^d \to \mathbb{R}$ such that

$$\lim_{w \to u} \frac{J(w) - A(w)}{\|w - u\|} = 0$$

Affine function A is called the (best) affine approximation of J at u



A may depend on u—i.e., possibly a different A for each u

About the affine approximation:

- Since A is affine, we can write it as

$$A(w) = \_\_\_\_\_$$

- $m \in \mathbb{R}^d$ is the "slope" (and specifies a linear function)
- $b \in \mathbb{R}$ is the "intercept"
- The intercept must be $b = \_\_\_\_\_$ because

$$J(u) = \_\_\_\_\_$$

- So we can write A as

$$A(w) = J(u) + m^\mathsf{T}(w - u)$$

About the affine approximation:

Letting $e^{(1)}, \ldots, e^{(d)}$ be standard coordinate basis for $\mathbb{R}^d$, write $m = \sum_{i=1}^{d} m_i \, e^{(i)}$

Since $A(w) = J(u) + m^\mathsf{T}(w - u)$ is best affine approximation of J at u,

$$0 = \lim_{t \to 0} \frac{J(u + te^{(i)}) - A(u + te^{(i)})}{|t|} = \lim_{t \to 0} \frac{J(u + te^{(i)}) - (J(u) + tm_i)}{|t|}$$

since $u + te^{(i)}$ differs from u by $t \in \mathbb{R}$ in the i-th coordinate

Whether t approaches zero from left or right, we find

$$m_i = \lim_{t \to 0} \_\_\_\_\_ = \_\_\_\_\_$$

Vector-valued function (a.k.a. vector field) of all partial derivatives of J is called the gradient of J, written $\nabla J \colon \mathbb{R}^d \to \mathbb{R}^d$

$$\nabla J(u) = \left(\frac{\partial J}{\partial w_1}(u), \ldots, \frac{\partial J}{\partial w_d}(u) \right)$$

Summary: If $J \colon \mathbb{R}^d \to \mathbb{R}$ is differentiable, then for any $u \in \mathbb{R}^d$,

$$\lim_{w \to u} \frac{J(w) - (J(u) + \nabla J(u)^\intercal (w - u))}{\|w - u\|} = 0$$

Gradient descent

(Back to $\min_{w \in \mathbb{R}^d} J(w)$ where J is differentiable)

Question: Given candidate setting of variables $w = u \in \mathbb{R}^d$, achieving objective value $J(u)$, how can we change u to achieve a lower objective value?

Upshot: Modify u by subtracting $\eta \nabla J(u)$ for some $\eta > 0$

Caveat: Approximations in our argument are OK only if "change" is "small enough" (which means η should be "small enough")

Gradient descent: iterative method that attempts to minimize $J : \mathbb{R}^d \to \mathbb{R}$

- Initialize $w^{(0)} \in \mathbb{R}^d$
- For iteration $t = 1, 2, \ldots$ until "stopping condition" is satisfied:

$$w^{(t)} \leftarrow w^{(t-1)} - \eta_t \nabla J(w^{(t-1)}) \qquad \text{(update rule)}$$

- Return final $w^{(t)}$

What's missing in this algorithm description?

Examples of gradient descent algorithms

Sum of squared errors objective from OLS

$$J(w) = \sum_{(x,y)\in\mathcal{S}} (x^\mathsf{T} w - y)^2$$

for dataset $\mathcal{S}$ from $\mathbb{R}^d \times \mathbb{R}$

- Use linearity and chain rule to get formula for $\frac{\partial J}{\partial w_i}$:

$$\frac{\partial J}{\partial w_i}(w) = \sum_{(x,y)\in\mathcal{S}} \_\_\_\_\_$$

- Therefore

$$\nabla J(w) = \sum_{(x,y)\in\mathcal{S}} \_\_\_\_\_$$

- Update rule in iteration t:

$$w^{(t)} \leftarrow w^{(t-1)} - \eta_t \sum_{(x,y)\in\mathcal{S}} \_\_\_\_\_$$

Negative log-likelihood from logistic regression

$$J(w) = \sum_{(x,y)\in\mathcal{S}} \left(\ln(1 + e^{x^\mathsf{T} w}) - y x^\mathsf{T} w\right)$$

for dataset $\mathcal{S}$ from $\mathbb{R}^d \times \{0, 1\}$

- Use linearity and chain rule to get formula for $\frac{\partial J}{\partial w_i}$:

$$\frac{\partial J}{\partial w_i}(w) = \sum_{(x,y)\in\mathcal{S}} \_\_\_\_\_$$

- Therefore

$$\nabla J(w) = \sum_{(x,y)\in\mathcal{S}} \_\_\_\_\_$$

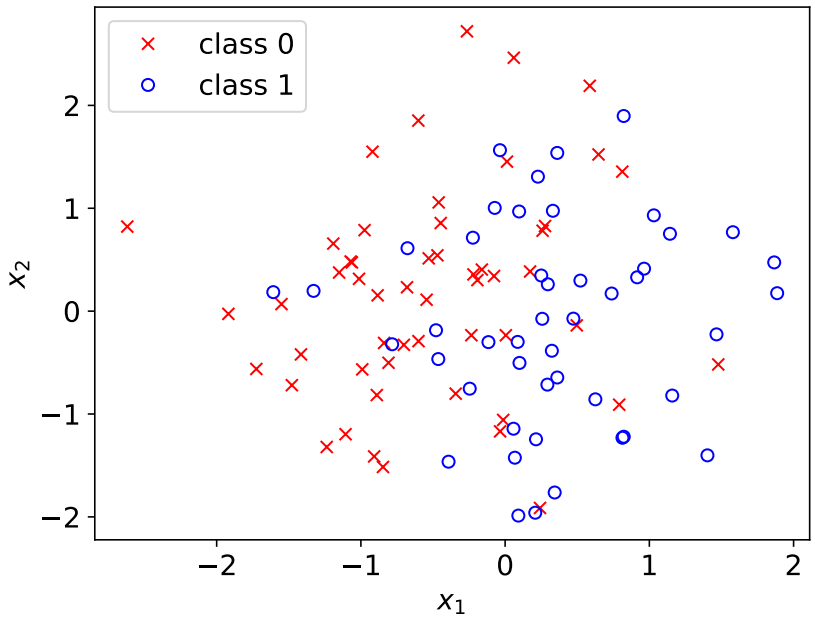
- Update rule in iteration t:

$$w^{(t)} \leftarrow w^{(t-1)} - \eta_t \sum_{(x,y)\in\mathcal{S}} \_\_\_\_\_$$

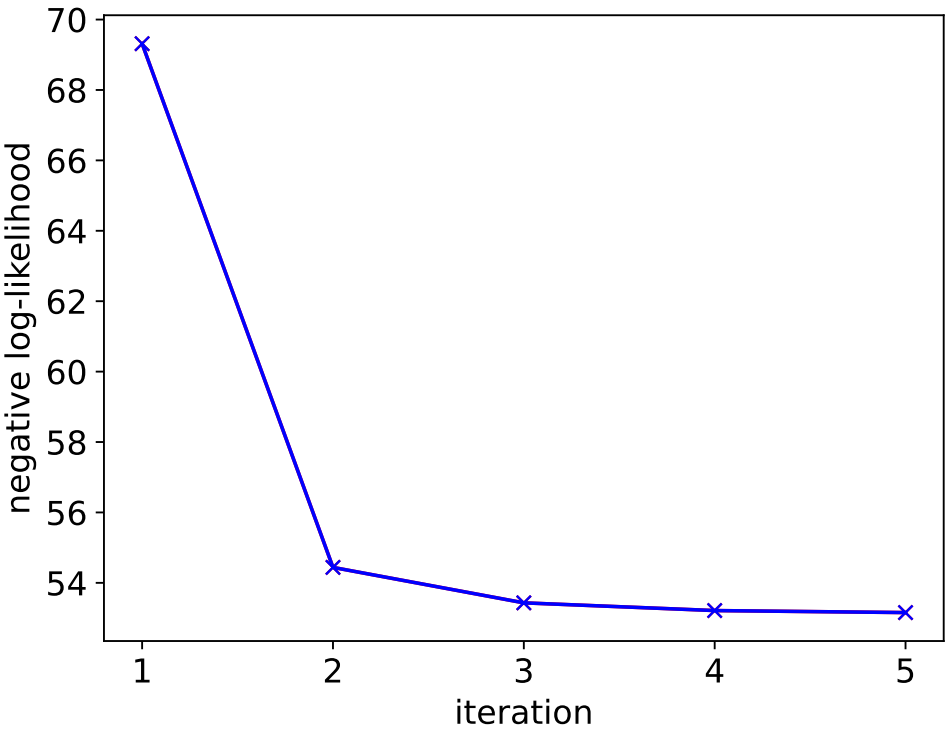
```python
def learn(train_x, train_y, eta=0.1, num_steps=1000):
  w = np.zeros(train_x.shape[1])
  for t in range(num_steps):
    w += eta * (train_y - 1/(1+np.exp(-train_x.dot(w)))).dot(train_x)
  return w
```


Synthetic example: $X \sim \mathrm{N}((0,0), I)$, conditional distribution of Y given $X = x$ is $\mathrm{Bernoulli}(\mathrm{logistic}(w^\intercal x))$ for $w = (3/2, -1/2)$

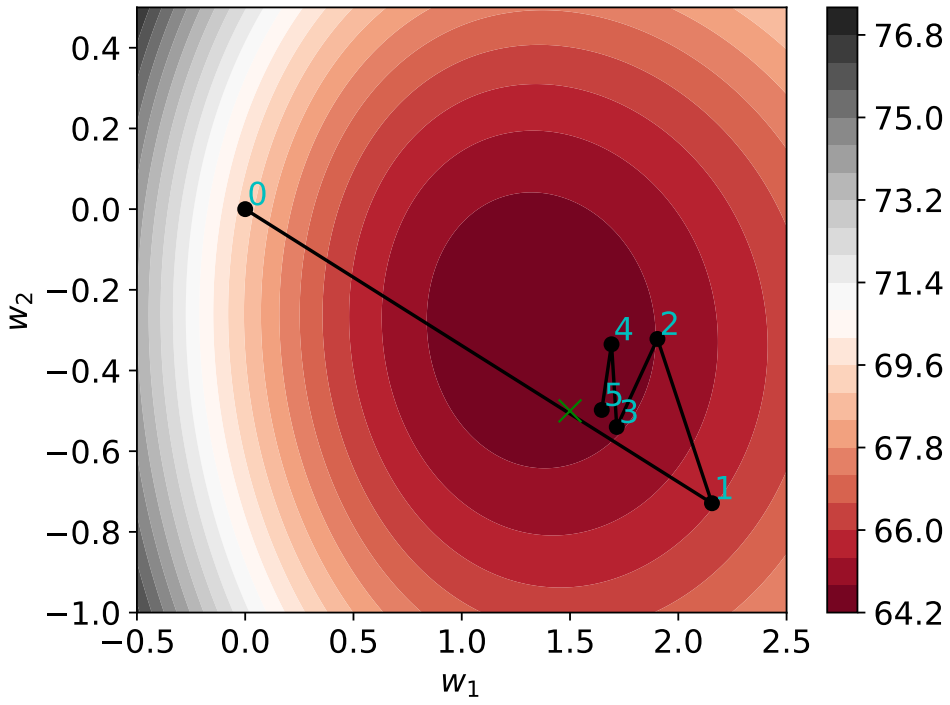
- $n = 100$ training examples $\mathcal{S} \overset{\text{i.i.d.}}{\sim} (X, Y)$



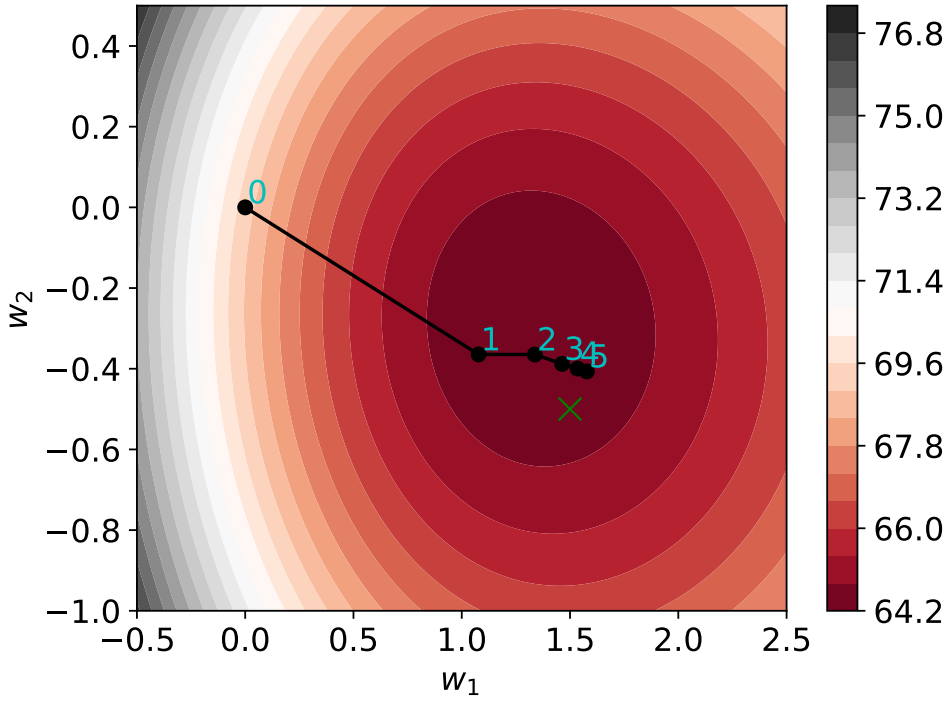
$\eta_t = 0.1$ starting from $w^{(0)} = (0, 0)$



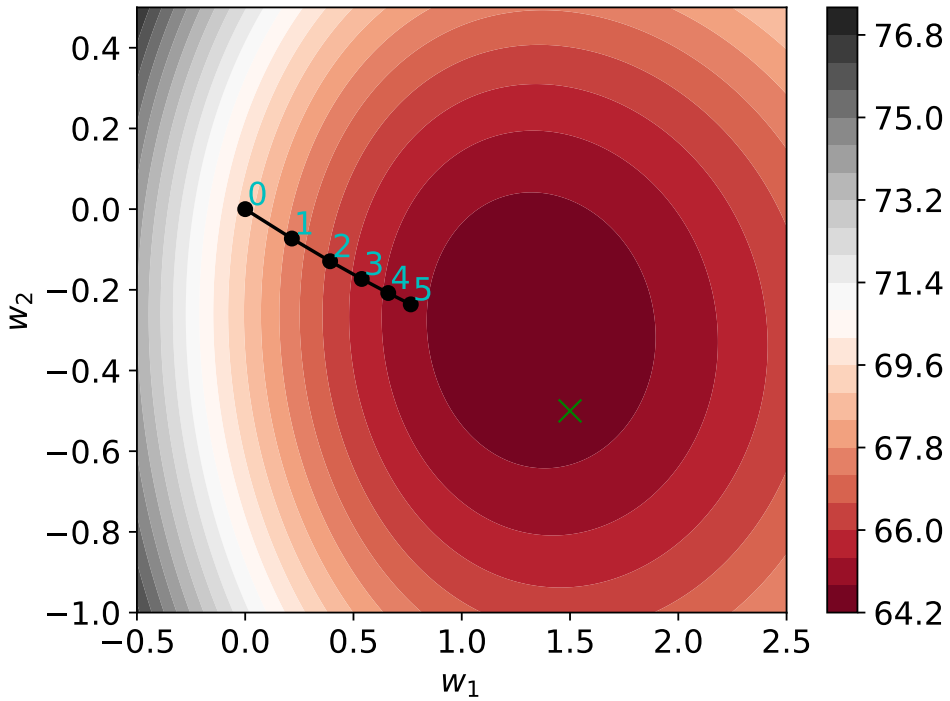
$\eta_t = 0.1$ starting from $w^{(0)} = (0, 0)$



$\eta_t = 0.05$ starting from $w^{(0)} = (0, 0)$



$\eta_t = 0.01$ starting from $w^{(0)} = (0, 0)$



Properties of gradient descent

Guarantee about gradient descent updates: If J is "smooth enough", then there is a choice for $\eta > 0$ such that, for any $u \in \mathbb{R}^d$,

$$J(u - \eta \nabla J(u)) \leq J(u) - \frac{\eta}{2} \|\nabla J(u)\|^2$$

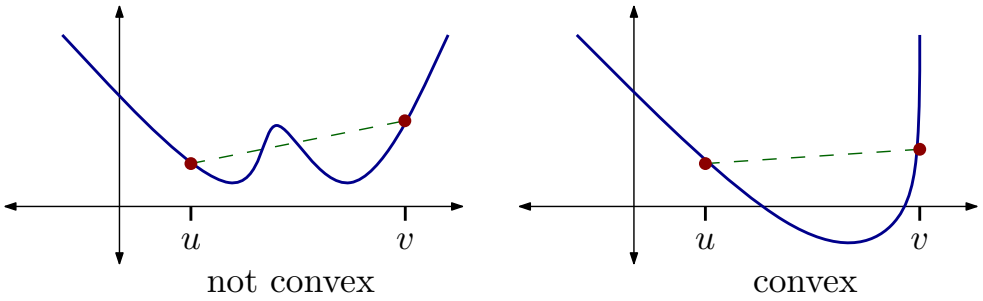
Guarantee about gradient descent for convex objectives: If J is convex and "smooth enough", then there is a choice for $\eta > 0$ such that, for any $w^{(0)} \in \mathbb{R}^d$, iterates of gradient descent $w^{(1)}, w^{(2)}, \ldots$ (with $\eta_t = \eta$) satisfy

$$\lim_{t \to \infty} J(w^{(t)}) = \min_{w \in \mathbb{R}^d} J(w)$$

Convex functions

A function $J\colon \mathbb{R}^d \to \mathbb{R}$ is convex if, for all $u, v \in \mathbb{R}^d$, and all $\alpha \in [0, 1]$,

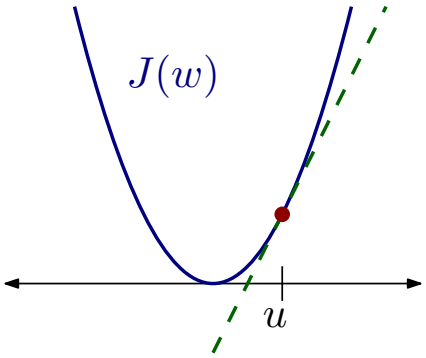
$$J((1-\alpha)u + \alpha v) \leq (1-\alpha)J(u) + \alpha J(v)$$



A differentiable function $J\colon \mathbb{R}^d \to \mathbb{R}$ is convex if, for all $u, w \in \mathbb{R}^d$,

$$J(w) \geq J(u) + \nabla J(u)^\mathsf{T}(w - u)$$

i.e., J lies above all of its affine approximations



A continuously twice-differentiable function $J\colon \mathbb{R}^d \to \mathbb{R}$ is convex if, for all $u \in \mathbb{R}^d$, the $d \times d$ matrix of second derivatives of J at u is positive semidefinite

Operations that preserve convexity:

- Sum of convex functions $J_1\colon \mathbb{R}^d \to \mathbb{R}$ and $J_2\colon \mathbb{R}^d \to \mathbb{R}$

$$J(w) = J_1(w) + J_2(w)$$

- Non-negative scalar multiple of a convex function $J_0\colon \mathbb{R}^d \to \mathbb{R}$

$$J(w) = c\, J_0(w), \quad c \geq 0$$

- Max of convex functions $J_1\colon \mathbb{R}^d \to \mathbb{R}$ and $J_2\colon \mathbb{R}^d \to \mathbb{R}$

$$J(w) = \max\{J_1(w), J_2(w)\}$$

- Composition of convex function $J_0\colon \mathbb{R}^k \to \mathbb{R}$ with affine mapping

$$J(w) = J_0(Mw + b)$$

  for $M \in \mathbb{R}^{k \times d}$ and $b \in \mathbb{R}^k$

Example: sum of squared errors $J(w) = \sum_{(x,y)\in\mathcal{S}} (x^\mathsf{T} w - y)^2$

Why convexity of J helps with gradient descent:

- Convexity ensures negative gradient $-\nabla J(u)$ satisfies

$$(-\nabla J(u))^\mathsf{T}(w - u) \geq J(u) - J(w)$$

  for all $u, w \in \mathbb{R}^d$
- Suppose w is minimizer of J, and you currently have u in hand
- Ideal direction to move in: $\delta = w - u$

Stochastic gradient descent

Many objective functions in machine learning are decomposable, i.e., can be written as sum

$$J(w) = \sum_{i=1}^{n} J^{(i)}(w)$$

E.g., sum of losses on training examples

$$J^{(i)}(w) = \text{loss}(f_w(x^{(i)}), y^{(i)})$$

Computational cost to compute $\nabla J(w)$?

Alternative: instead of using

$$\nabla J(w) = \sum_{i=1}^{n} \nabla J^{(i)}(w),$$

just use one of the terms in the sum (chosen uniformly at random)

Stochastic gradient descent (SGD) for $J(w) = \sum_{i=1}^{n} J^{(i)}(w)$

- Initialize $w^{(0)} \in \mathbb{R}^d$
- For iteration $t = 1, 2, \ldots$ until "stopping condition" is satisfied:

$$w^{(t)} \leftarrow w^{(t-1)} - \eta_t \nabla J^{(I_t)}(w^{(t-1)}) \quad \text{where } I_t \sim \text{Unif}(\{1, \ldots, n\})$$

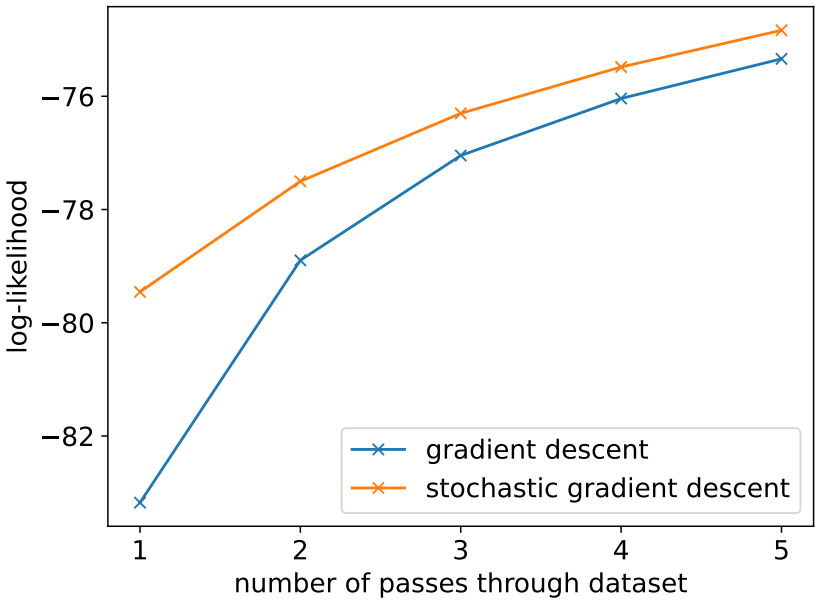
- Return final $w^{(t)}$

Some practical variants of SGD:

- Use sampling without replacement to choose $I_1, I_2, \ldots, I_n$ (i.e., go through terms in a uniformly random order)
  - Called SGD without replacement
- Instead of updating with gradient of single term, update with sum of gradients for next B terms
  - Called minibatch SGD; B is the minibatch size

Iris dataset, treating versicolor and virginica as a single class

- Maximizing log-likelihood in logistic regression with gradient descent and with SGD (both using $\eta_t = 0.01$, starting from $w^{(0)} = (0, 0)$)



Practical considerations

- Conditioning

- Initialization $w^{(0)} \in \mathbb{R}^d$

- Choice of "step size" $\eta_t > 0$ (a.k.a. "learning rate")

- Stopping condition