

Optimization by gradient methods

COMS 4771 Fall 2025

Unconstrained optimization problems

Common form of optimization problem in machine learning:

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

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

1 / 45

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

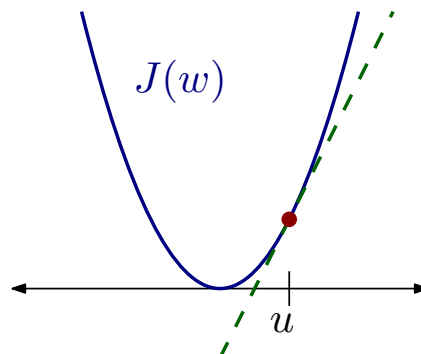
2 / 45

Review of multivariate differential calculus

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

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

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

$$A(w) = \underline{\hspace{2cm}}$$

▶ $m \in \mathbb{R}^d$ is the “slope” (and specifies a linear function)

▶ $b \in \mathbb{R}$ is the “intercept”

▶ The intercept must be $b = \underline{\hspace{2cm}}$ because

$$J(u) = \underline{\hspace{2cm}}$$

▶ So we can write A as

$$A(w) = J(u) + m^\top(w - u)$$

4 / 45

About the affine approximation:

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

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

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

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

$$m_i = \lim_{t \rightarrow 0} \underline{\hspace{2cm}} = \underline{\hspace{2cm}}$$

5 / 45

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

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

6 / 45

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

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

7 / 45

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”)

9 / 45

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

▶ Initialize $w^{(0)} \in \mathbb{R}^d$

▶ For iteration $t = 1, 2, \dots$ until “stopping condition” is satisfied:

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

▶ Return final $w^{(t)}$

10 / 45

What's missing in this algorithm description?

Examples of gradient descent algorithms

Sum of squared errors objective from OLS

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

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

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

$$\frac{\partial J}{\partial w_i}(w) = \sum_{(x,y) \in \mathcal{S}} \underline{\hspace{10em}}$$

► Therefore

$$\nabla J(w) = \sum_{(x,y) \in \mathcal{S}} \underline{\hspace{10em}}$$

► Update rule in iteration t :

$$w^{(t)} \leftarrow w^{(t-1)} - \eta_t \sum_{(x,y) \in \mathcal{S}} \underline{\hspace{10em}}$$

12 / 45

Negative log-likelihood from logistic regression

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

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

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

$$\frac{\partial J}{\partial w_i}(w) = \sum_{(x,y) \in \mathcal{S}} \underline{\hspace{10em}}$$

► Therefore

$$\nabla J(w) = \sum_{(x,y) \in \mathcal{S}} \underline{\hspace{10em}}$$

► Update rule in iteration t :

$$w^{(t)} \leftarrow w^{(t-1)} - \eta_t \sum_{(x,y) \in \mathcal{S}} \underline{\hspace{10em}}$$

13 / 45

```

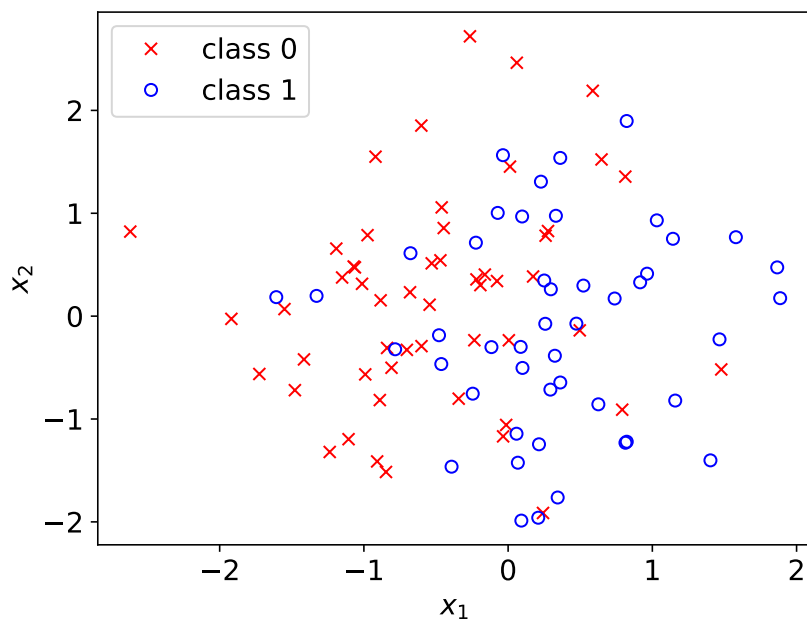
def learn(train_x, train_y, eta=0.1, num_steps=1000):
    w = np.zeros(train_x.shape[1])
    for t in range(num_steps):
        w += eta * (train_y - 1/(1+np.exp(-train_x.dot(w)))) .dot(train_x)
    return w

```

14 / 45

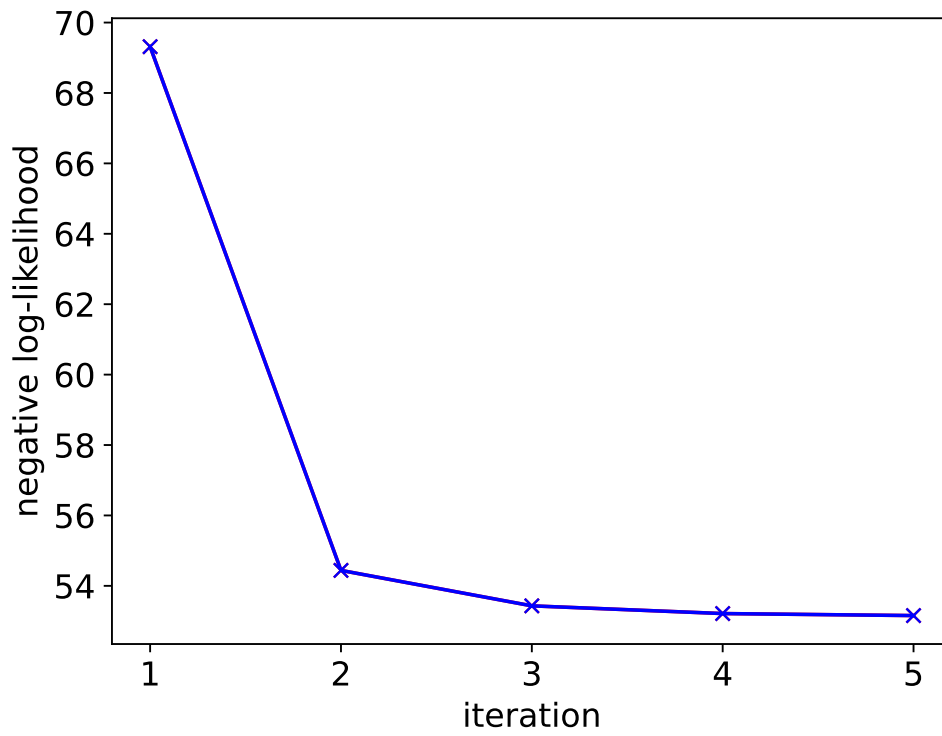
Synthetic example: $X \sim N((0,0), I)$, conditional distribution of Y given $X = x$ is Bernoulli(logistic($w^T x$)) for $w = (3/2, -1/2)$

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

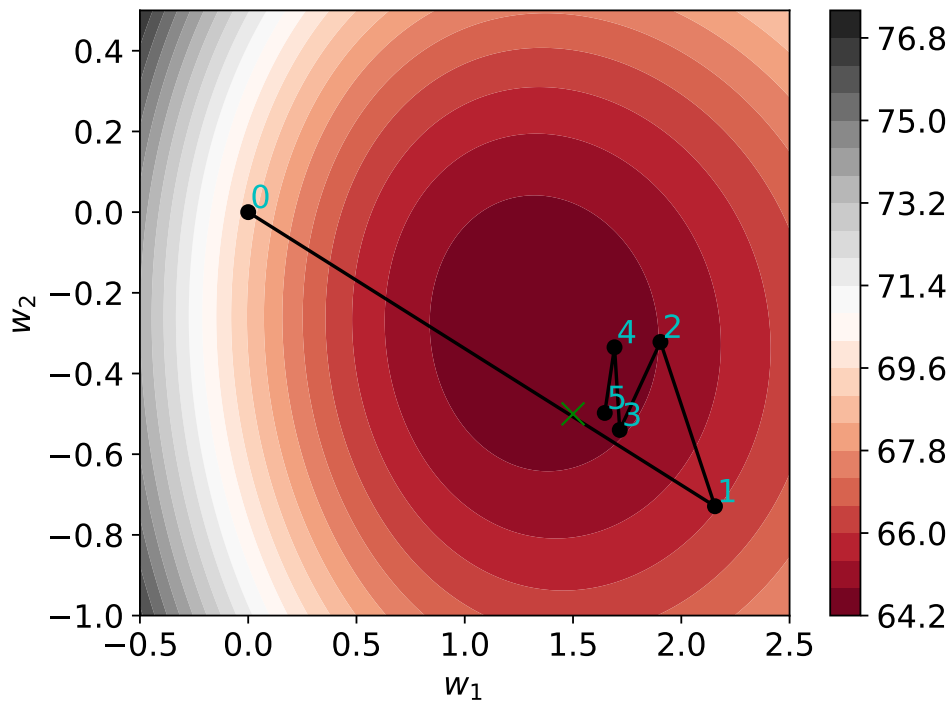


15 / 45

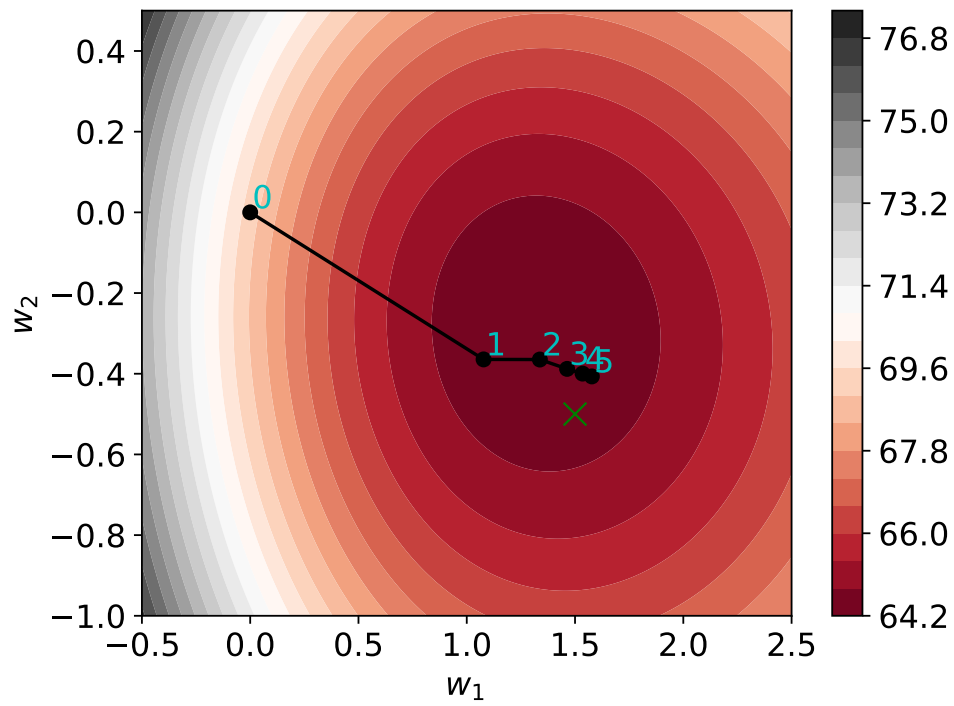
$\eta_t = 0.1$ starting from $w^{(0)} = (0, 0)$



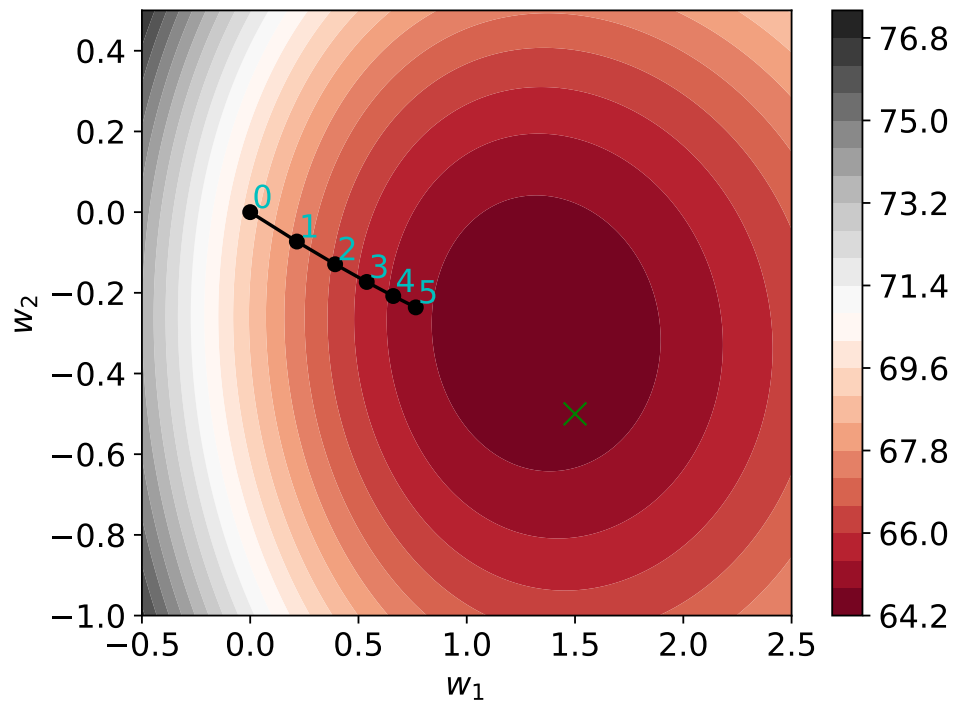
$\eta_t = 0.1$ starting from $w^{(0)} = (0, 0)$



$\eta_t = 0.05$ 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 $u \in \mathbb{R}^d$,

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

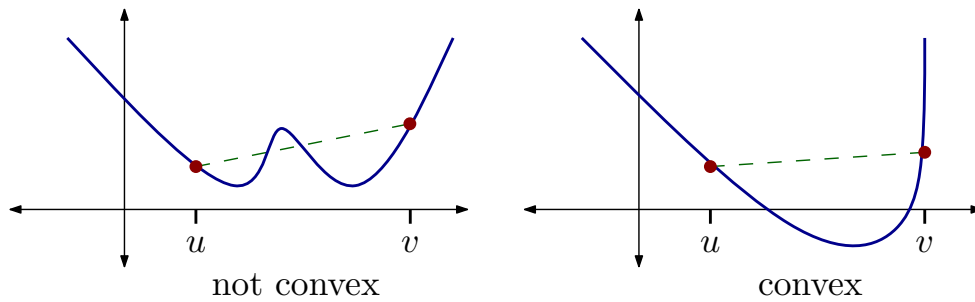
Guarantee about gradient descent for convex objectives: If J is convex and “smooth enough”, then there is a choice for $\eta > 0$ such that, for any $w^{(0)} \in \mathbb{R}^d$, iterates of gradient descent $w^{(1)}, w^{(2)}, \dots$ (with $\eta_t = \eta$) satisfy

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

Convex functions

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

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

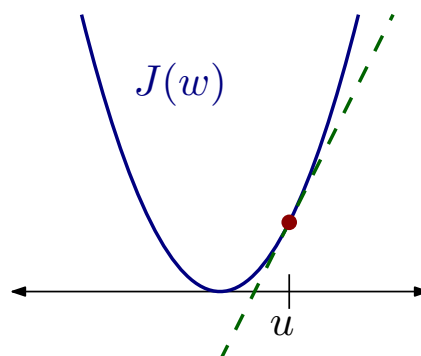


22 / 45

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

$$J(w) \geq J(u) + \nabla J(u)^\top (w - u)$$

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



23 / 45

A continuously twice-differentiable function $J: \mathbb{R}^d \rightarrow \mathbb{R}$ 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

24 / 45

Operations that preserve convexity:

- ▶ Sum of convex functions $J_1: \mathbb{R}^d \rightarrow \mathbb{R}$ and $J_2: \mathbb{R}^d \rightarrow \mathbb{R}$

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

- ▶ Non-negative scalar multiple of a convex function $J_0: \mathbb{R}^d \rightarrow \mathbb{R}$

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

- ▶ Max of convex functions $J_1: \mathbb{R}^d \rightarrow \mathbb{R}$ and $J_2: \mathbb{R}^d \rightarrow \mathbb{R}$

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

- ▶ Composition of convex function $J_0: \mathbb{R}^k \rightarrow \mathbb{R}$ with affine mapping

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

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

25 / 45

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

26 / 45

Why convexity of J helps with gradient descent:

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

$$(-\nabla J(u))^\top (w - u) \geq J(u) - J(w)$$

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: $\delta = w - u$

27 / 45

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 $t = 1, 2, \dots$ until “stopping condition” is satisfied:

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

- ▶ Return final $w^{(t)}$

29 / 45

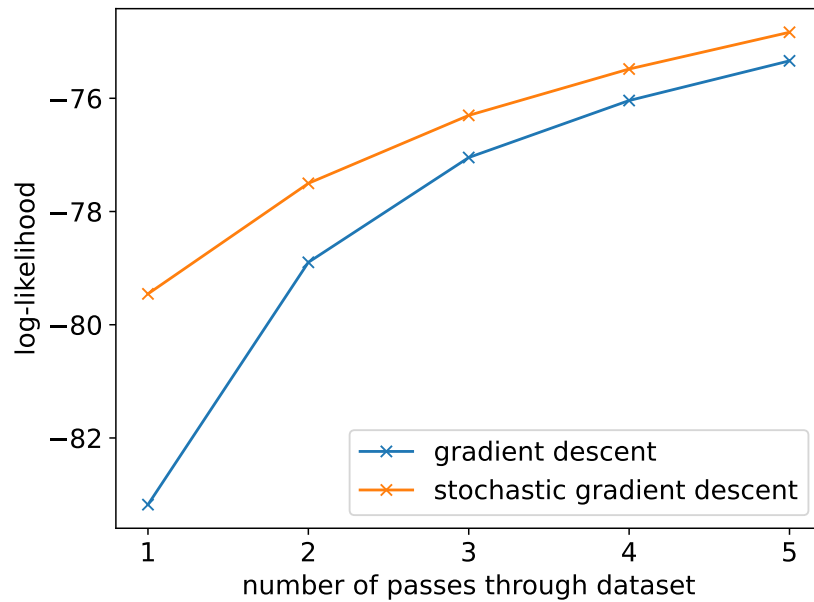
Some practical variants of SGD:

- ▶ Use sampling without replacement to choose I_1, I_2, \dots, 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

30 / 45

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 $w^{(0)} = (0, 0)$)



Additional considerations

▶ Issue #1: Initialization $w^{(0)} \in \mathbb{R}^d$

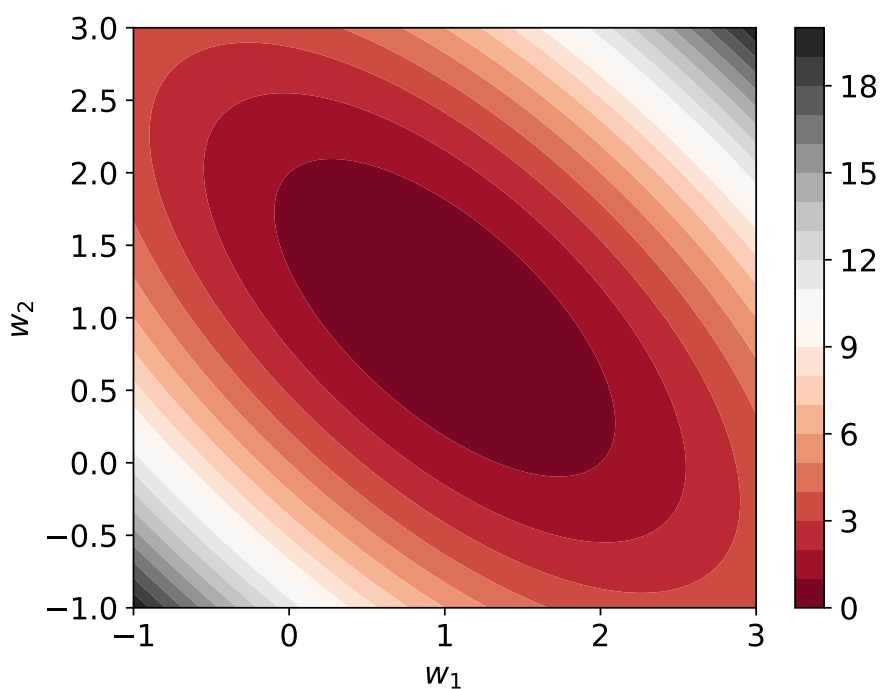
▶ Issue #2: Choice of “step size” $\eta_t > 0$ (a.k.a. “learning rate”)

32 / 45

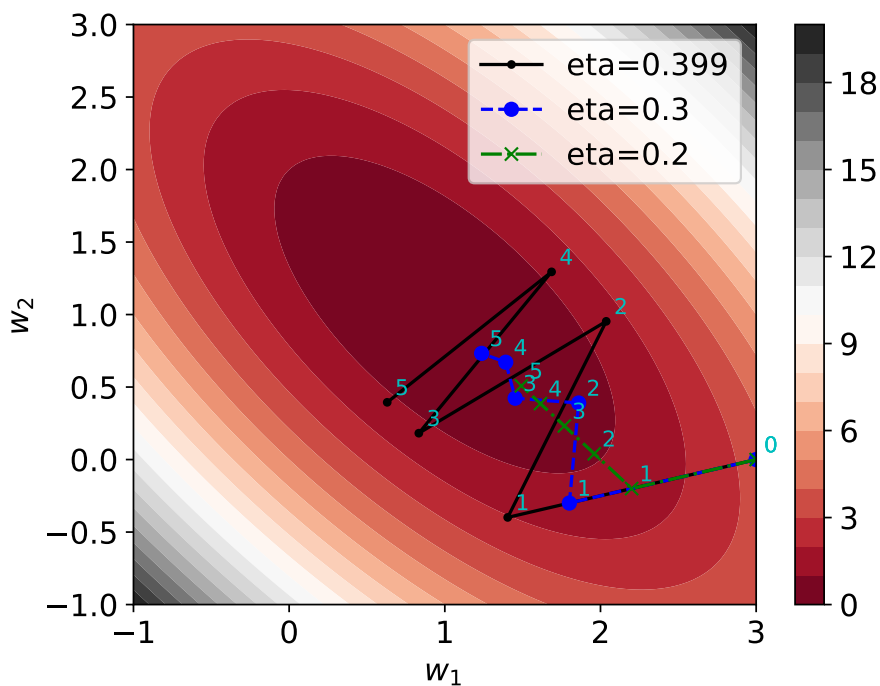
▶ Issue #3: Conditioning

33 / 45

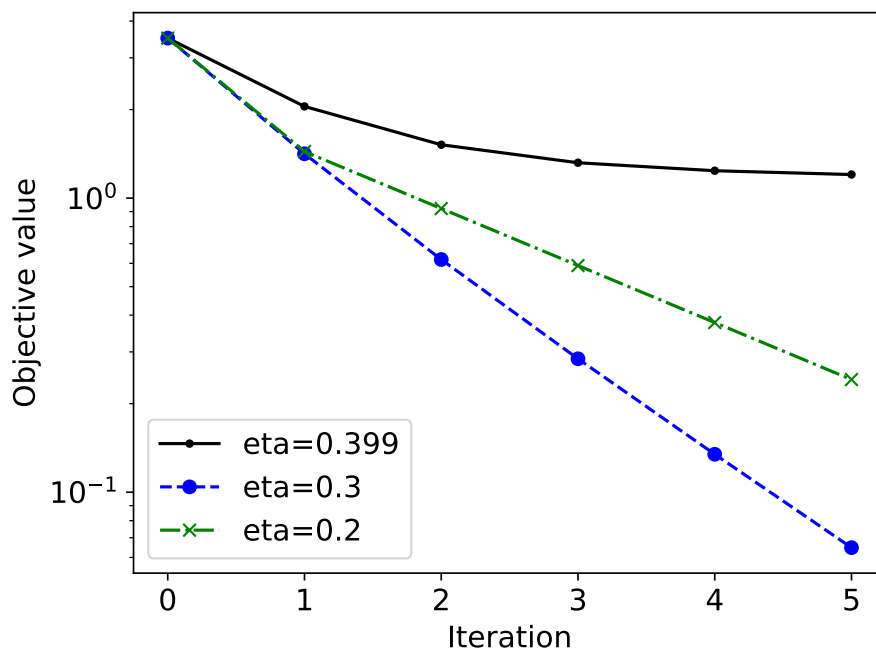
$$J(w) = \frac{1}{2}(w - w^*)^T Q (w - w^*), \text{ where } \frac{\lambda_2(Q)}{\lambda_1(Q)} = 0.2$$



$$J(w) = \frac{1}{2}(w - w^*)^T Q (w - w^*), \text{ where } \frac{\lambda_2(Q)}{\lambda_1(Q)} = 0.2$$



$$J(w) = \frac{1}{2}(w - w^*)^\top Q(w - w^*), \text{ where } \frac{\lambda_2(Q)}{\lambda_1(Q)} = 0.2$$



36 / 45

Pre-conditioning: change-of-variables to (try to) make level sets of objective function more “spherical” (or otherwise better behaved)

- ▶ Change-of-variables: $w = Ru$
- ▶ New objective $\tilde{J}(u) = J(Ru)$
- ▶ GD update for u :

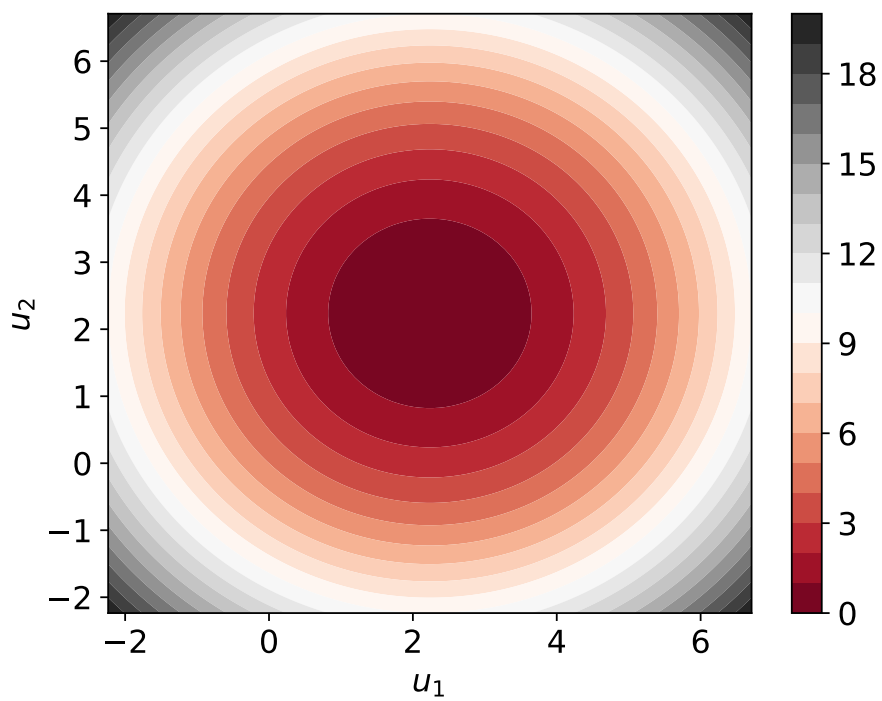
$$\begin{aligned} u^{(t)} &= u^{(t-1)} - \eta \nabla \tilde{J}(u^{(t-1)}) \\ &= u^{(t-1)} - \eta R^\top \nabla J(Ru^{(t-1)}) \end{aligned}$$

To get corresponding $w^{(t)}$, multiply $u^{(t)}$ by R :

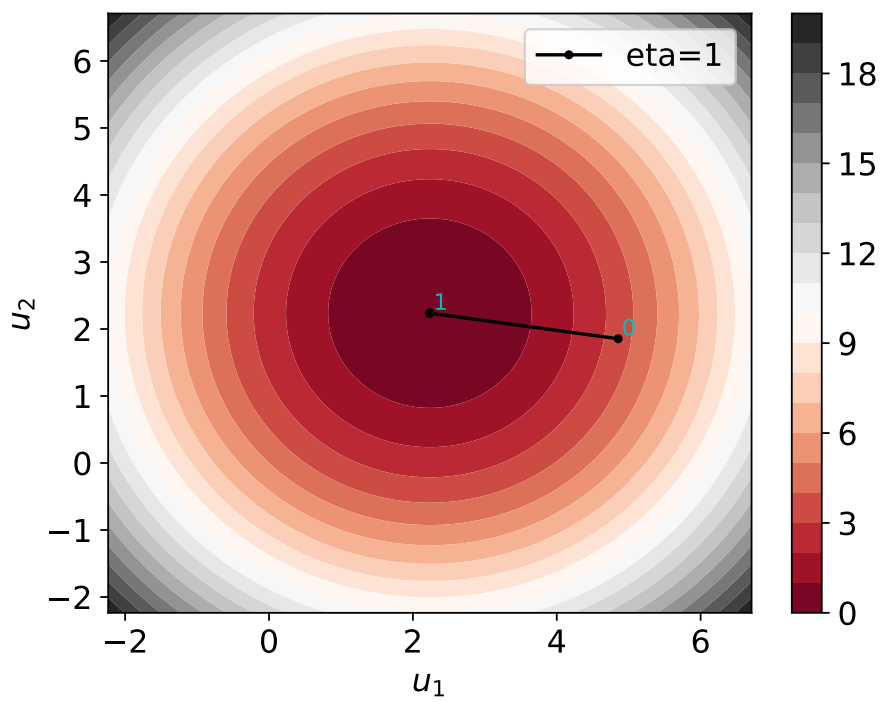
$$\begin{aligned} w^{(t)} &= Ru^{(t)} = R(u^{(t-1)} - \eta R^\top \nabla J(Ru^{(t-1)})) \\ &= w^{(t-1)} - \eta RR^\top \nabla J(w^{(t-1)}) \end{aligned}$$

37 / 45

$$\tilde{J}(u) = J(Ru) \text{ for } R = Q^{-1/2}$$



$$\tilde{J}(u) = J(Ru) \text{ for } R = Q^{-1/2}$$



- ▶ Where does pre-conditioner R come from?
 - ▶ When $J(w) = \|Aw - b\|^2$ (OLS objective), “optimal” pre-conditioner comes from SVD of A
 - ▶ (But computational effort to find R same as solving normal equations)
 - ▶ In general, need domain / objective-specific information
 - ▶ Try PCA + standardization
- ▶ Diagonal pre-conditioner: R is a diagonal matrix
 - ▶ Faster to multiply vectors by diagonal matrix than general matrices
- ▶ Can also use pre-conditioning with SGD

- ▶ Issue #4: Stopping condition

OLS objective: $J(w) = \|Aw - b\|^2$, where SVD of A is $A = \sum_{i=1}^r \sigma_i u_i v_i^\top$

▶ Minimum norm solution:

$$\hat{w} := A^\dagger b = \sum_{i=1}^r \frac{1}{\sigma_i} \langle u_i, b \rangle v_i$$

▶ GD starting from $w^{(0)} = 0$ after t steps with step size η :

$$\langle v_i, w^{(t)} \rangle = 2\eta\sigma_i^2 \sum_{k=0}^{t-1} (1 - 2\eta\sigma_i^2)^k \langle v_i, \hat{w} \rangle$$

If $2\eta\sigma_1^2 < 1$, then $w^{(t)} \rightarrow \hat{w}$ and $J(w^{(t)}) \rightarrow \min_{w \in \mathbb{R}^d} J(w)$ as $t \rightarrow \infty$

▶ Early stopping (i.e., using “small” t) = regularization effect

42 / 45

Principal Components Regression (PCR) (with hyperparameter $\lambda \geq 0$)

▶ Conceptually a two-step procedure:

1. Replace A with low-rank approximation based on truncated SVD

$$\hat{A} := \sum_{i=1}^r \mathbb{1}\{\sigma_i^2 > \lambda\} \sigma_i u_i v_i^\top$$

2. Get OLS solution for (\hat{A}, b) in rowspace of \hat{A}

▶ Result:

$$\hat{w}_\lambda^{\text{PCR}} := \hat{A}^\dagger b = \sum_{i=1}^r \frac{\mathbb{1}\{\sigma_i^2 > \lambda\}}{\sigma_i} \langle u_i, b \rangle v_i$$

$$\langle v_i, \hat{w}_\lambda^{\text{PCR}} \rangle = \mathbb{1}\{\sigma_i^2 > \lambda\} \langle v_i, \hat{w} \rangle$$

43 / 45

Minimum norm solution to normal equations (where $A = \sum_{i=1}^r \sigma_i u_i v_i^\top$)

$$\langle v_i, \hat{w} \rangle = \frac{1}{\sigma_i} \langle u_i, b \rangle$$

Ridge(λ)

$$\langle v_i, \hat{w}_\lambda^{\text{Ridge}} \rangle = \frac{1}{1 + \lambda/\sigma_i^2} \langle v_i, \hat{w} \rangle$$

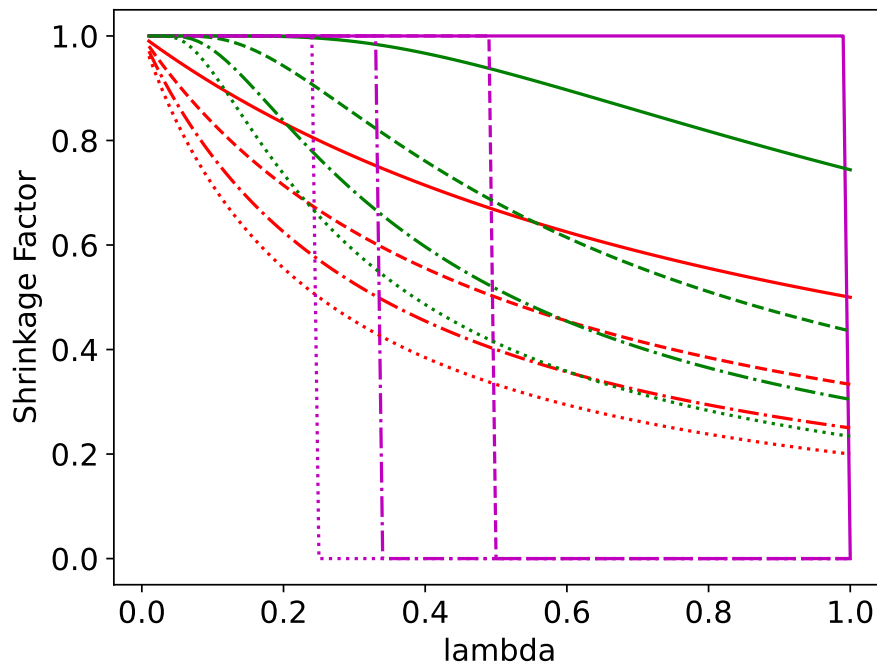
PCR(λ)

$$\langle v_i, \hat{w}_\lambda^{\text{PCR}} \rangle = \mathbb{1}\{\sigma_i^2 > \lambda\} \langle v_i, \hat{w} \rangle$$

GD (starting from $w^{(0)} = 0$ after t steps with step size η , assuming $2\eta\sigma_1^2 < 1$)

$$\langle v_i, \hat{w}^{(t)} \rangle = (1 - (1 - 2\eta\sigma_i^2)^t) \langle v_i, \hat{w} \rangle$$

$$\sigma_i^2 \in \{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\}$$



(For GD, $t = 0.5/(\eta\lambda)$)