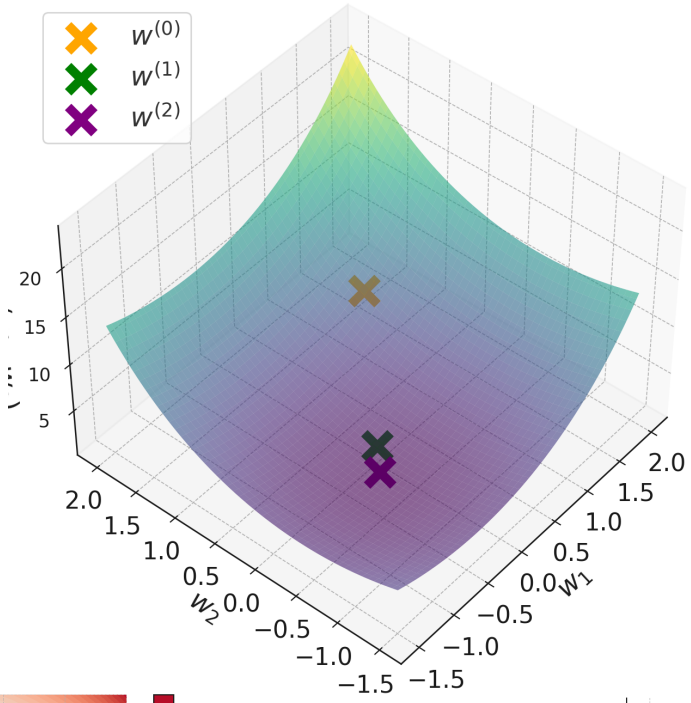
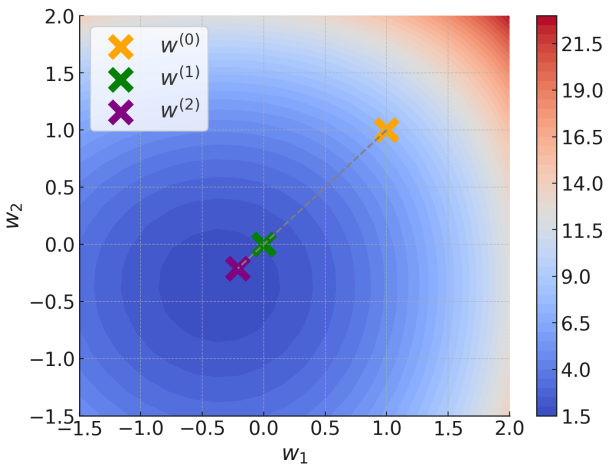
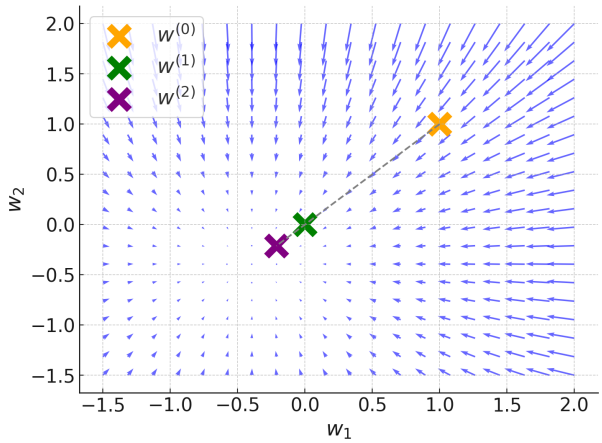
Important announcements

Feb 25

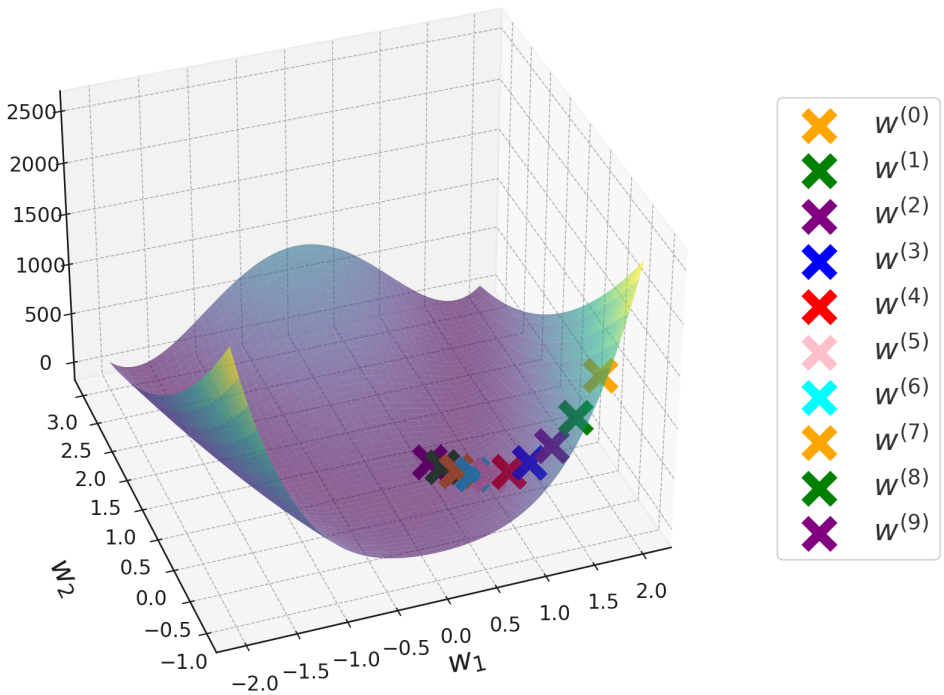
- grades for midterm 1 are out
- example for linear regression

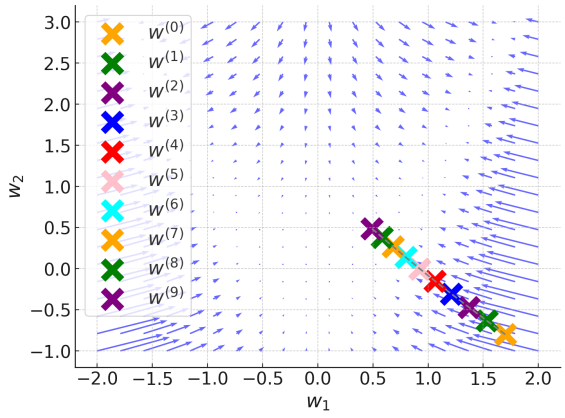


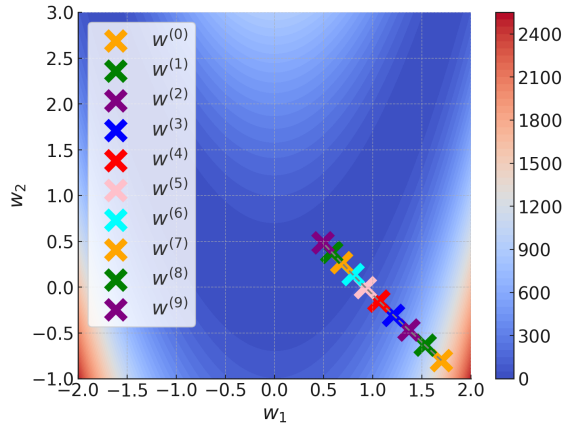




Ex: $g(w_1, w_2) = (1 - w_1)^2 + 100(w_2 - w_1^2)^2$







Selecting the step size

goal: be close to $\frac{1}{g''(w^{(t)})}$

Common options (for all $t \in \mathbb{N}$)

1) constant: $\eta^{(t)} = \eta \in (0, \infty)$

2) inverse decaying: $\eta^{(t)} = \frac{\eta}{1+\lambda t}$ where $\eta, \lambda \in (0, \infty)$

3) exponential decaying: $\eta^{(t)} = \eta e^{-\lambda t}$ where $\eta, \lambda \in (0, \infty)$

4) Normalized gradient decaying $\eta^{(t)} = \frac{\eta}{\varepsilon + \|\nabla g(\vec{w}^{(t)})\|}$ ε is small (ex: $\varepsilon = 10^{-8}$)

where $\|\nabla g(\vec{w}^{(t)})\| = \sqrt{\sum_{j=1}^{d} \frac{\partial}{\partial w_j} g(\vec{w}^{(t)})^2}$

Gradient Descent with $\hat{L}(f)$

$$D = \left((\vec{x}_1, y), \dots, (\vec{x}_n, y_n) \right) \in (\mathcal{X} \times \mathcal{Y})^n$$

$$\mathcal{X} = \mathbb{R}^{d+1}, \quad \mathcal{Y} = \mathbb{R} \quad \text{regression}$$

$$\tilde{\mathcal{F}} = \{ f \mid f: \mathcal{X} \to \mathcal{Y} \text{ and } f(\vec{x}) = \vec{x}^T \vec{w}, \ \vec{w} \in \mathbb{R}^{d+1} \}$$ linear function class

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$ squared loss

Objective: find $\hat{\vec{w}} = \underset{\vec{w} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \hat{L}(\vec{w})$

where $\hat{L}(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(\vec{x}_i^T \vec{w}, y_i)$

$$= \frac{1}{n} \sum_{i=1}^{n} (\vec{x}_i^T \vec{w} - y_i)^2$$

(Batch) Gradient Descent (BGD)

we know the closed form solution $\vec{w} = A^{-1} \vec{b}$ but try to solve the optimization step of ERM with gradient descent anyway

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta^{(t)} \nabla \hat{L}(\vec{w})$$

$$\nabla \hat{L}(\vec{w}) = \left(\frac{\partial}{\partial w_0} \hat{L}(\vec{w}), \dots, \frac{\partial}{\partial w_d} \hat{L}(\vec{w}) \right)^T \in \mathbb{R}^{d+1}$$

$$= \left(\frac{2}{n} \sum_{i=1}^{n} (\vec{x}_i^T \vec{w} - y_i) x_{i,1}, \dots, \frac{2}{n} \sum_{i=1}^{n} (\vec{x}_i^T \vec{w} - y_i) x_{i,d} \right)^T$$

$$= \frac{2}{n} \sum_{i=1}^{n} (\vec{x}_i^T w - y_i) \underbrace{\left(x_{i,1}, \dots, x_{i,d} \right)^T}_{\vec{x}_i}$$

$$= \frac{2}{n} \sum_{i=1}^{n} (\vec{x}_i^T \vec{w} - y_i) \vec{x}_i$$

Algorithm: BGD Linear Regression Learner (with const. step size)

input: $D = \left((\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n) \right)$ (dataset), η (step size), T (number of epochs)

$\vec{w} \leftarrow$ random vector in $\mathbb{R}^{d+1}$

for $t = 1, \dots, T$

$$\nabla \hat{L}(\vec{w}) \leftarrow \frac{2}{n} \sum_{i=1}^{n} (\vec{x}_i^T \vec{w} - y_i) \vec{x}_i$$

$$\vec{w} \leftarrow \vec{w} - \eta \nabla \hat{L}(\vec{w})$$

return: $\hat{f}(\vec{x}) = \vec{x}^T \vec{w}$

The reason to still sometimes use GD is computation

computation: closed form solution vs gradient descent

goal: number of computational steps should be minimal

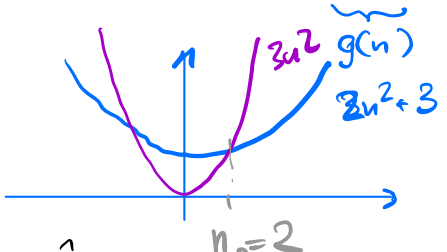
- adding
- substracting
- multiplication

Big O notation: $f(n) = O(g(n))$ $\quad f(n) < c\,g(n) \quad n \geq n_0$

Ex: $f(n) = 2n^2 + 3 = O(n^2)$

$2n^2 + 3 \leq 3n^2$ for $n \geq 2$

$24 + 3 \leq 3 \cdot 4$



closed form: $\vec{w} = A^{-1}\vec{b}$ where $A = \sum_{i=1}^{n} \vec{x}_i \vec{x}_i^T$, $\vec{b} = \sum_{i=1}^{n} \vec{x}_i y_i$

A: $O(d^2 n)$ $\quad A^{-1}$: $O(d^3)$

$\vec{b}$: $O(dn)$ $\quad A^{-1}\vec{b}$: $O(d^2)$

Total computation: $O(d^2 n + dn + d^3 + d^2) = O(d^2 n + d^3)$

BGD $\nabla \hat{L}(\vec{w}^{(t)})$: $O(dn)$ $\vec{w}$: $O(d)$

→ Total computation : $O(dnT)$

Let's compare $O(d^2n + d^3)$ and $O(dnT)$

if $T < d$ (#epochs, #features) : $O(dnT) < O(d^2n) \leq O(d^2n + d^3)$

=> BGD is more computationally efficient if $T < d$

Can we do better in terms of computation ?

Mini-batch gradient descent (MBGD)

$$D = \Big((\vec{x}_1, y_1), \dots, (\vec{x}_b, y_b), (\vec{x}_{b+1}, y_{b+1}), \dots, (\vec{x}_{2b}, y_{2b}), \vdots (\vec{x}_{(M-1)b+1}, y_{(M-1)b+1}), \dots, (\vec{x}_n, y_n) \Big)$$

good : we want to break up the sum into smaller chunks
→ mini-batches
⊕ sum smaller → more efficient
⊖ estimate is worse

where b mini-batch size

$M = \frac{n}{b}$ number of mini-batches

Ex: $n = 8$, $b = 2$, $\mu = \frac{8}{2} = 4$ → integer

$$\mathbb{D} = (\underbrace{(\vec{x}_1, y_1), (\vec{x}_2, y_2)}_{MB\,1}, \underbrace{(\vec{x}_3, y_3), (\vec{x}_4, y_4)}_{MB\,2} \underbrace{(\vec{x}_5, y_5), (\vec{x}_6, y_6)}_{MB\,3} \underbrace{(\vec{x}_7, y_7), (\vec{x}_8, y_8)}_{MB\,4})$$

$\mu = 4$
mini-batches

$\frac{n}{b}$ is not always an integer. We define

$$\mu = \text{floor}\left(\frac{n}{b}\right)$$

→ We will not use $n - \mu b < b$ datapoints

New estimate of the expected loss for each mini-batch

$$\hat{L}_m = \frac{1}{b} \sum_{i=(m-1)b+1}^{mb} (\vec{x}_i^T \vec{w} - y_i)^2 \quad \text{where } m \in \{1, \dots, \mu\}$$

→ different estimate for each batch

$\hat{L}_m$ is a sample mean estimate of $L(\vec{w})$ with b datapoints

→ $\hat{L}_m(\vec{w})$ is a worse estimate than $\hat{L}(\vec{w})$

Algorithm: MBGD Linear Regression Learner
(with constant stepsize)

input: $D = ((\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n))$, η, T, b

$\vec{w} \leftarrow$ random vector in $\mathbb{R}^{d+1}$

$M \leftarrow \text{floor}\left(\frac{n}{b}\right)$

for $t = 1, \ldots, T$

randomly shuffle D

for $m = 1, \ldots, M$

$$\nabla \hat{L}_m(\vec{w}) \leftarrow \frac{2}{b} \sum_{i=mb+1}^{(m+1)b} (\vec{x}_i^T \vec{w} - y_i) \vec{x}_i$$

$$\vec{w} \leftarrow \vec{w} - \eta \nabla \hat{L}_m(\vec{w})$$

return $\hat{f}(\vec{x}) = \vec{x}^T \vec{w}$