DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 4 | Part 1

Introduction

Empirical Risk Minimization (ERM)

- Step 1: choose a **hypothesis class**
  - We’ve chosen linear predictors.
- Step 2: choose a **loss function**
- Step 3: find H minimizing **empirical risk**

Minimizing Empirical Risk

- We want to minimize the **empirical risk**:

$$R(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \ell(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}, y_i)$$

- For some choices of loss function, we can find a formula for the minimizer.

Example: Least Squares

- With the square loss, risk becomes:

$$R(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2$$

- Setting gradient to zero, solving for $\vec{w}$ gives:

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

Gradient Descent

- But sometimes we **can't** solve for $\vec{w}$ **directly**.
  - It's too costly.
  - There's no closed-form solution.

- **Idea:** use **gradient descent** to iteratively minimize risk.

Gradient Descent

- Starting from an initial guess $\vec{w}^{(0)}$, iteratively update:
$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \frac{dR}{d\vec{w}}(\vec{w}^{(t)})$$

Today

We'll address two issues with gradient descent.

1. Can be **expensive** to compute the exact gradient.
   - Especially when we have a large data set.
   - **Solution:** **stochastic gradient descent**.
2. Doesn't work as-is if risk is **not differentiable**.
   - Such as with the absolute loss.
   - **Solution:** **subgradient descent**.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 4 | Part 2

Motivation: Large Scale Learning

Example

- Suppose you're doing **least squares regression** on a medium-to-large data set.
- Say, $n = 200{,}000$ examples, $d = 5{,}000$ features.
- Encoded as 64 bit floats, X is 8 GB.
  - Fits in your laptop's memory, but barely.
- **Example:** predict sentiment from text.

Attempt 0: Normal Equations

- You start by solving the normal equations:

```
np.linalg.solve(X.T @ X, X.T @ y)
```

- **Time:** 30.7 seconds.
- **Mean Squared Error:** 7.2×10^{-7}.
- Can we speed this up?

Attempt 1: Gradient Descent

- Recall[1] that the gradient of the MSE is:

$$\frac{dR}{d\vec{w}}(\vec{w}) = \frac{2}{n}\sum_{i=1}^{n}\left(\text{Aug}(\vec{x}^{(i)})\cdot\vec{w} - y_i\right)\text{Aug}(\vec{x}^{(i)})$$
$$= \frac{1}{n}\left(2X^T X\vec{w} - 2X^T\vec{y}\right)$$

- You code up a function:[2]

```
def gradient(w):
    n = len(y)
    return (2/n) * X.T @ (X @ w - y)
```

[1] From Lecture 02, where we derived this.

[2] There's a good and a bad way to do this.

Attempt 1: Gradient Descent

- You plug this into `gradient_descent` from last lecture, run it, and...

- **Time: 8.6 seconds** total
  - 14 iterations
  - ≈ 0.6 seconds per iteration

- **Mean Squared Error:** 9.4×10^{-7}.

Trivia: why is it faster?

- **Solving normal equations** takes $\Theta(nd^2 + d^3)$ time.
  - $\Theta(nd^2)$ time to compute $X^T X$.
  - $\Theta(d^3)$ time to solve the system.

- **Gradient descent** takes $\Theta(nd)$ time per iteration.
  - $\Theta(nd)$ time to compute $X\vec{w}$.
  - $\Theta(nd)$ time to compute $X^T(X\vec{w} - \vec{y})$.

Looking Ahead

- What if you had a **larger** data set?
- Say, n = 10,000,000 examples, d = 5,000 features.
- Encoded as 64 bit floats, X is **400 GB**.
  - Doesn't fit in your laptop's memory!
  - Barely fits on your hard drive.

Approach 0: Normal Equations

- You can try solving the normal equations:

```
np.linalg.solve(X.T @ X, X.T @ y)
```

- One of three things will happen:
  1. You will receive an **out of memory** error.
  2. The process will be killed (or your OS will freeze).
  3. It will run, but take a **very long time** (paging).

Approach 1: Gradient Descent

- We can't store the data in memory all at once.
- But we can **still** compute the **gradient**, $\frac{dR}{d\vec{w}}$.
  - Read a little bit of data at once.
  - Or, distribute the computation to several machines.
- Computing gradient involves a loop over data:

$$\frac{dR}{d\vec{w}}(\vec{w}) = \frac{2}{n} \sum_{i=1}^{n} \left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i \right) \text{Aug}(\vec{x}^{(i)})$$

Problem

$$\frac{dR}{d\vec{w}}(\vec{w}) = \frac{2}{n}\sum_{i=1}^{n}\left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i\right)\text{Aug}(\vec{x}^{(i)})$$

- In machine learning, the number of training points n can be **very large**.
- Computing the gradient can be **expensive** when n is large.
  - So each step of gradient descent is **expensive**.

Idea

- Don’t worry about computing the **exact** gradient.
- An **approximation** will do.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 4 | Part 3

Stochastic Gradient Descent

Gradient Descent for Minimizing Risk

- In ML, we often want to minimize a **risk function**:

$$R(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

Observation

- The gradient of the risk is the average of the gradient of the losses:

$$\frac{d}{d\vec{w}} R(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \frac{d}{d\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- The averaging is over **all training points**.

- This can take a long time when n is large.[3]

[3]Trivia: this usually takes $\Theta(nd)$ time.

Idea

- The (full) gradient of the risk uses all of the training data:

$$\frac{d}{d\vec{w}} R(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \frac{d}{d\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- **Idea:** instead of using all n training points, randomly choose a smaller set, B:

$$\frac{d}{d\vec{w}} R(\vec{w}) \approx \frac{1}{|B|} \sum_{i \in B} \frac{d}{d\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

Stochastic Gradient

- The smaller set B is called a **mini-batch**.
- We now compute a **stochastic gradient**:

$$\frac{d}{d\vec{w}} R(\vec{w}) \approx \frac{1}{|B|} \sum_{i \in B} \frac{d}{d\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- "Stochastic," because it is a random.

Stochastic Gradient

$$\frac{d}{d\vec{w}} R(\vec{w}) \approx \frac{1}{|B|} \sum_{i \in B} \frac{d}{d\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- The stochastic gradient is an **approximation** of the full gradient.
- When $|B| \ll n$, it is **much faster** to compute.
- But the approximation is **noisy**.

Stochastic Gradient Descent for ERM

To minimize empirical risk $R(\vec{w})$:

- Pick starting weights $\vec{w}^{(0)}$, learning rate $\eta > 0$, batch size m.
- Until convergence, repeat:
  - **Randomly sample** a batch B of m training data points.
  - **Compute stochastic gradient:**

$$\vec{g} = \frac{1}{|B|} \sum_{i \in B} \frac{d}{d\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

  - **Update:** $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \vec{g}$
- When converged, return $\vec{w}^{(t)}$.

Note

- A **new batch** should be randomly sampled on each iteration!
- This way, the entire training set is used over time.
- Size of batch should be **small** compared to n.
  - Think: $m = 64$, $m = 32$, or even $m = 1$.

Example: Least Squares

- We can use SGD to perform least squares regression.

- Need to compute the gradient of the square loss:

$$\ell_{\text{sq}}(H(\vec{x}^{(i)}; \vec{w}), y_i) = \left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i\right)^2$$

Exercise

What is the gradient of the square loss of a linear predictor? That is, what is $\frac{d}{d\vec{w}}\left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i\right)^2$?

$$\frac{d}{dw}(xw - y)^2 = 2(xw - y) \times \frac{d}{dw}(xw - y)$$

$$= 2(xw - y) \times$$

Example: Least Squares

- The gradient of the square loss of a linear predictor is:

$$
\begin{aligned}
\frac{d}{d\vec{w}} \ell_{\text{sq}}(H(\vec{x}^{(i)}; \vec{w}), y_i) \\
&= \frac{d}{d\vec{w}} \left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i\right)^2 \\
&= 2\left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i\right) \frac{d}{d\vec{w}} \left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i\right) \\
&= 2\left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i\right) \text{Aug}(\vec{x}^{(i)})
\end{aligned}
$$

Example: Least Squares

- Therefore, on each step we compute the stochastic gradient:

$$\vec{g} = \frac{2}{m} \sum_{i \in B} \left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i \right) \text{Aug}(\vec{x}^{(i)})$$

- The update rule is:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \vec{g}$$

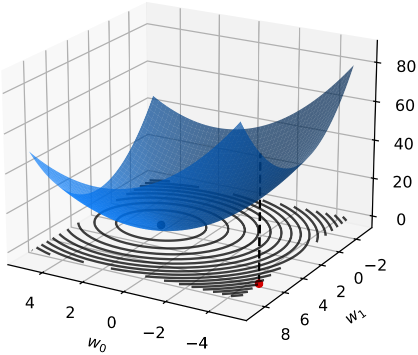
Example: Least Squares

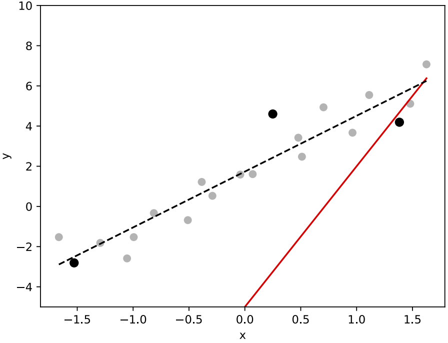
- We can write in matrix-vector form, too:
  - Let X_B be the design matrix using only the examples in batch B.
  - Let y_B be the corresponding vector of labels.

- Then:

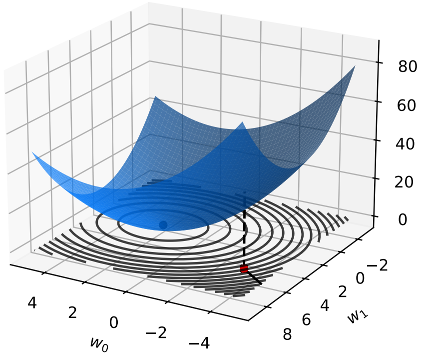
$$\vec{g} = \frac{2}{m} X_B^T (X_B \vec{w} - y_B)$$

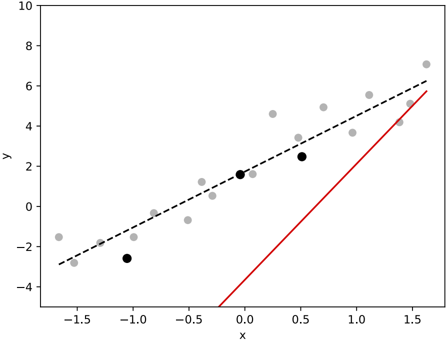
Example: SGD



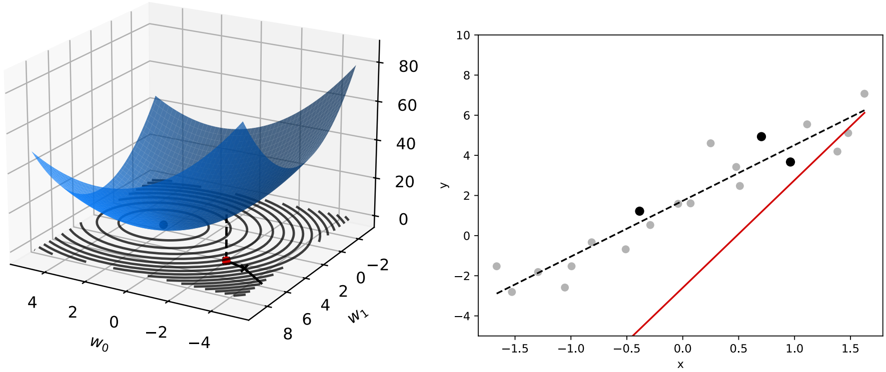


Example: SGD

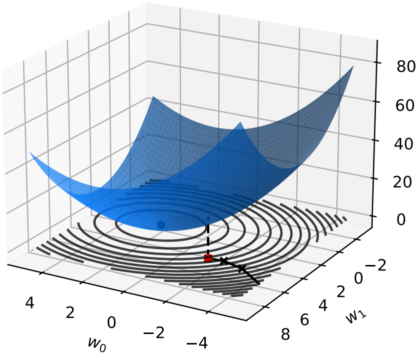


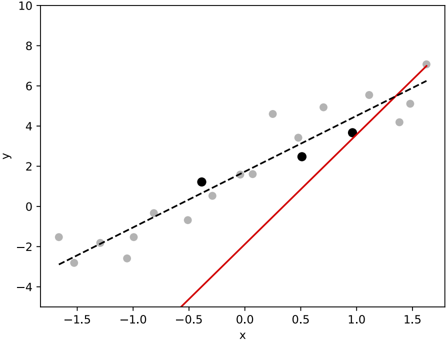


Example: SGD

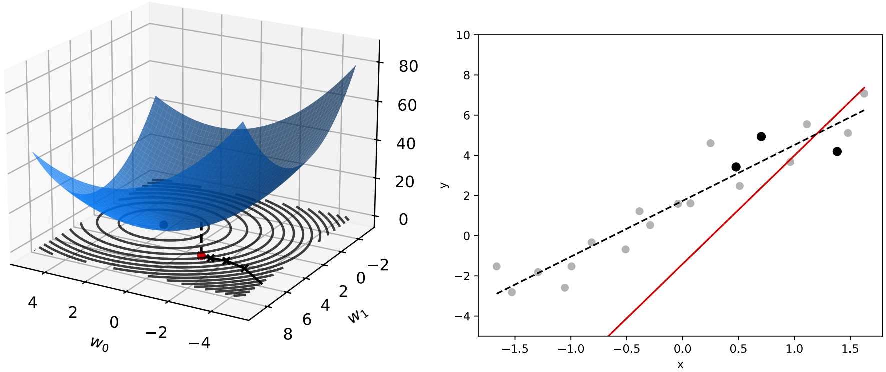


Example: SGD

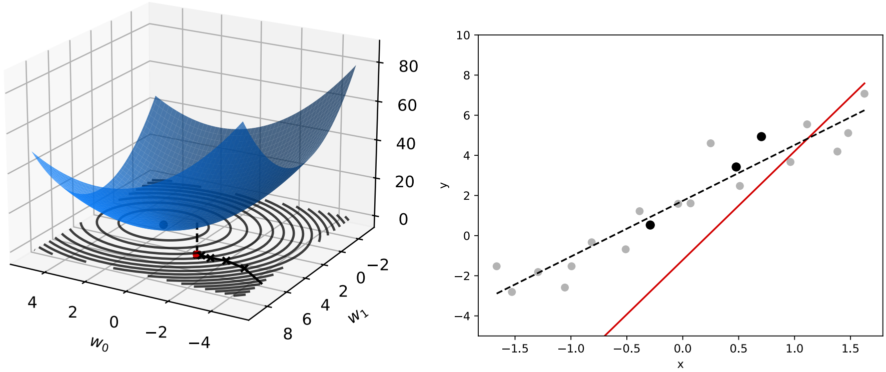




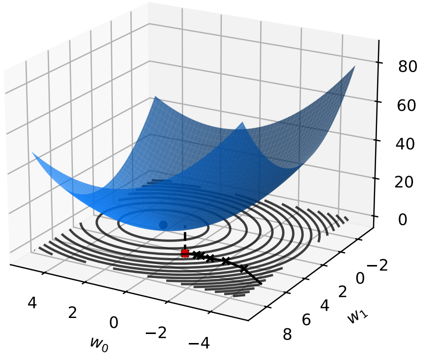
Example: SGD

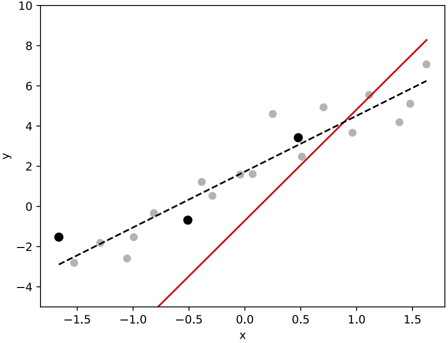


Example: SGD

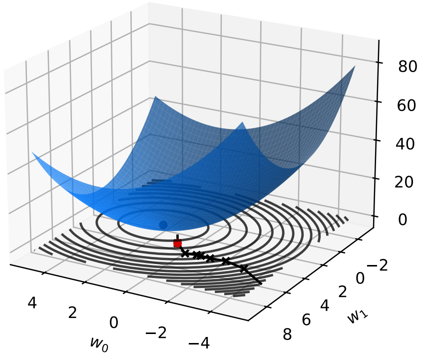


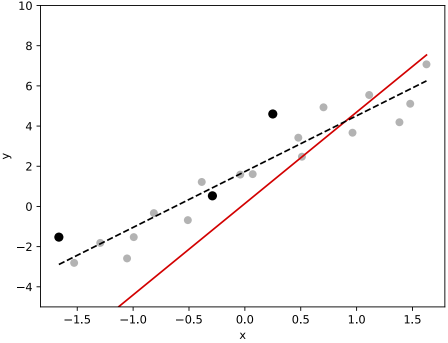
Example: SGD



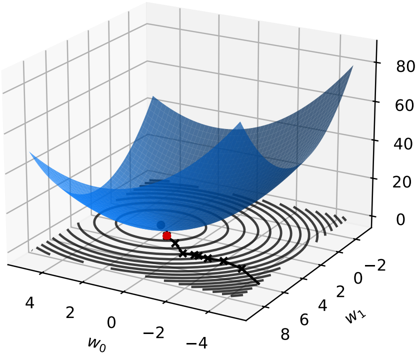


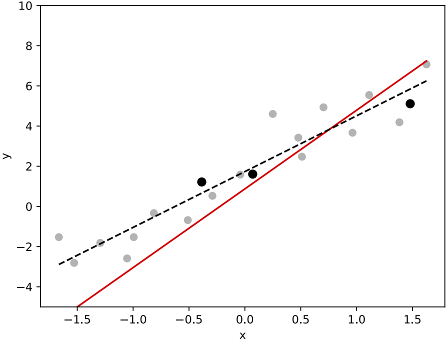
Example: SGD



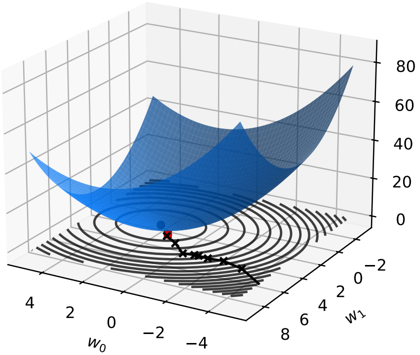


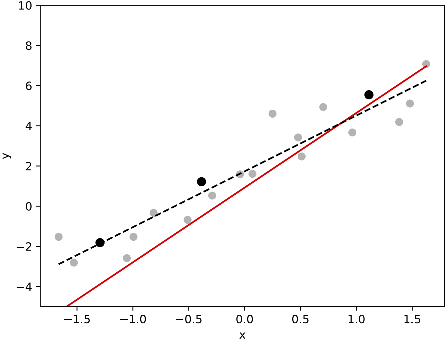
Example: SGD



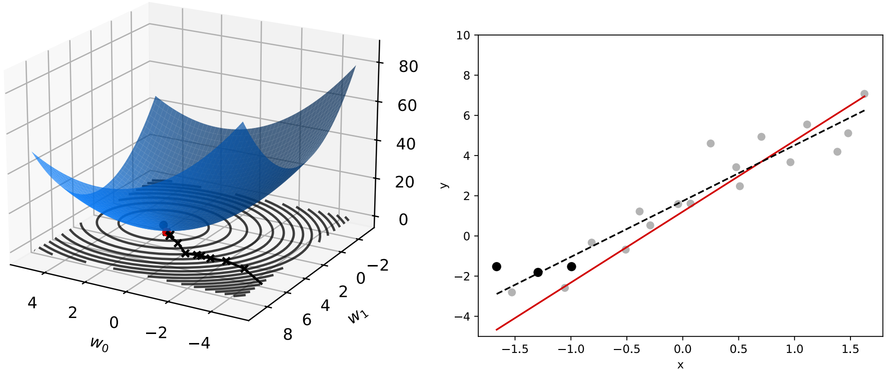


Example: SGD

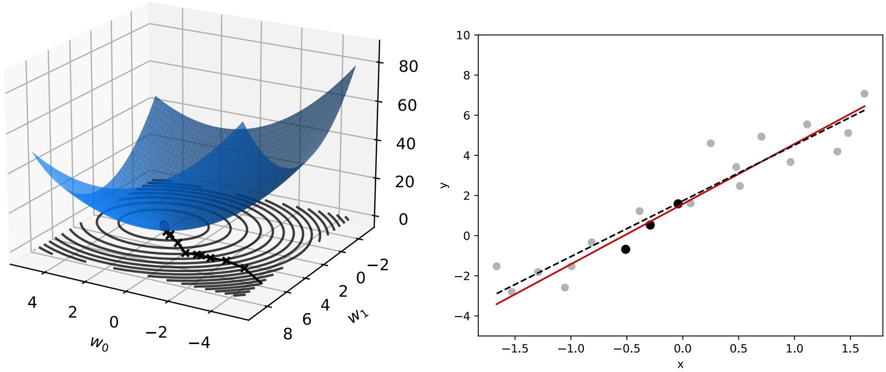




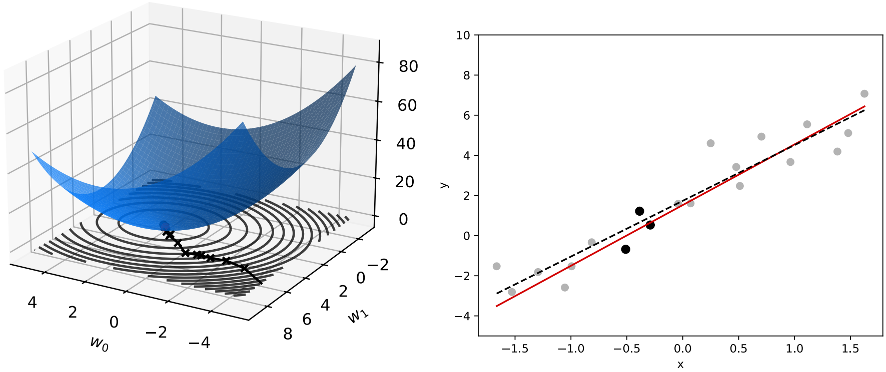
Example: SGD



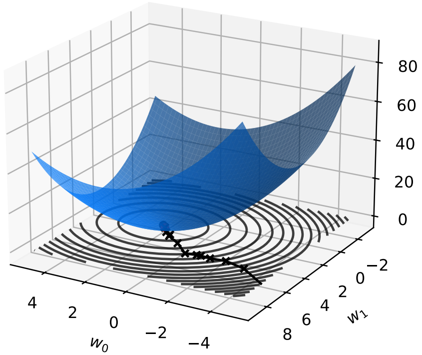
Example: SGD

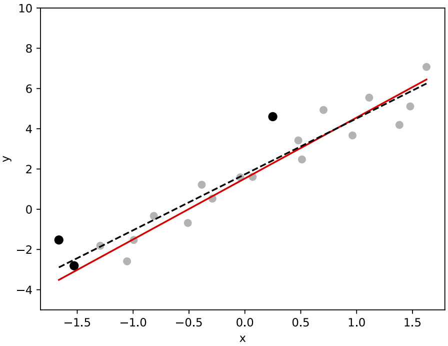


Example: SGD

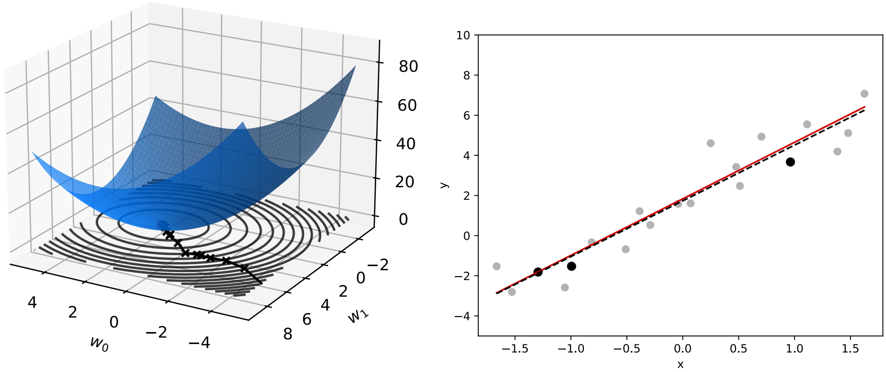


Example: SGD

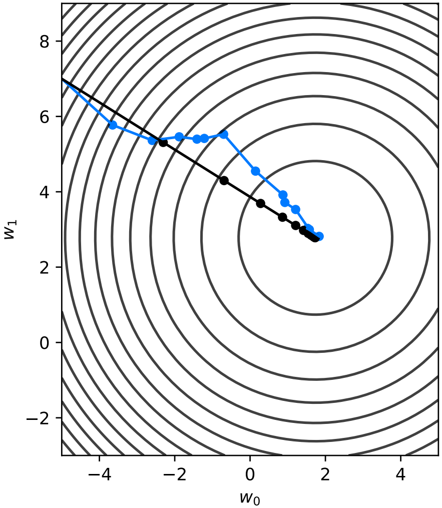




Example: SGD



SGD vs. GD



Tradeoffs

- In each step of GD, move in the “best” direction.
  - But **slowly!**
- In each step of SGD, move in a “good” direction.
  - But **quickly!**
- SGD may take more steps to converge, but can be faster overall.

Example

- Suppose you're doing **least squares regression** on a medium-to-large data set.
- Say, $n = 200{,}000$ examples, $d = 5{,}000$ features.
- Encoded as 64 bit floats, X is 8 GB.
  - Fits in your laptop's memory, but barely.
- **Example:** predict sentiment from text.

We saw...

- Solving the normal equations took **30.7 seconds**.
- Gradient descent took **8.6 seconds**.
  - 14 iterations, ≈ 0.6 seconds per iteration.
- Stochastic gradient descent takes **3 seconds**.
  - Batch size m = 16.
  - 13,900 iterations, ≈ 0.0002 seconds per iteration.

Aside: Terminology

- Some people say “stochastic gradient descent” only when batch size is 1.
- They say “mini-batch gradient descent” for larger batch sizes.
- **In this class**: we’ll use “SGD” for any batch size, as long as it’s chosen randomly.

Aside: A Popular Variant

- One variant of SGD uses **epochs**.
- During each epoch, we:
  - Randomly shuffle the training data.
  - Divide the training data into n/m mini-batches.
  - Perform one step for each mini-batch.

Usefulness of SGD

- SGD **enables** learning on **massive** data sets.
  - Billions of training examples, or more.

- Useful even when exact solutions available.
  - E.g., least squares regression / classification.

History: ADALINE

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 4 | Part 4

Motivation: Minimizing Absolute Loss

Empirical Risk Minimization (ERM)

- Step 1: choose a **hypothesis class**
  - We’ve chosen linear predictors.
- Step 2: choose a **loss function**
- Step 3: find H minimizing **empirical risk**

Loss Functions

- The **absolute loss** is a natural first choice for regression.

- The empirical risk becomes:

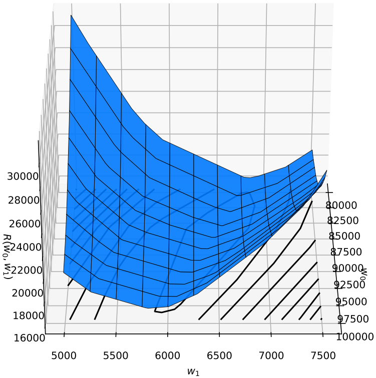
$$
\begin{aligned}
R_{\text{abs}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^{n} |H(\vec{x}^{(i)}) - y_i| \\
&= \frac{1}{n} \sum_{i=1}^{n} |\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i|
\end{aligned}
$$

Minimizing the Risk

$$R(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} |\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i|$$

- We might try computing the gradient, setting to zero, and solving.
- But the risk is **not differentiable**.

Risk for the Absolute Loss



Gradient Descent?

- **Question**: can we use gradient descent if the risk is not differentiable?
- **Answer**: **yes**, with a slight modification.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 4 | Part 5

Subgradient Descent

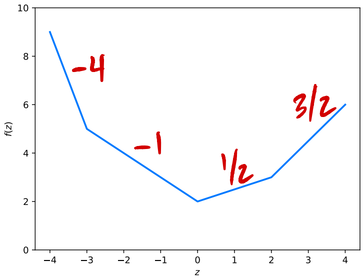
Differentiability

- A function $f(z)$ is **differentiable** if the derivative exists at every point.
- That is, it has a well-defined slope at every point.

Exercise

Where is the derivative **not** defined?

$$f(z) = \begin{cases} -4z - 7 & \text{if } z < -3 \\ -z + 2 & \text{if } -3 \leq z < 0 \\ 0.5z + 2 & \text{if } 0 \leq z < 2 \\ 3z/2 & \text{if } z \geq 2 \end{cases}$$

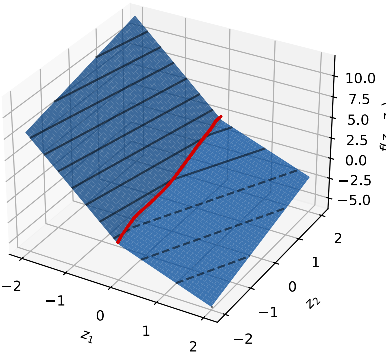


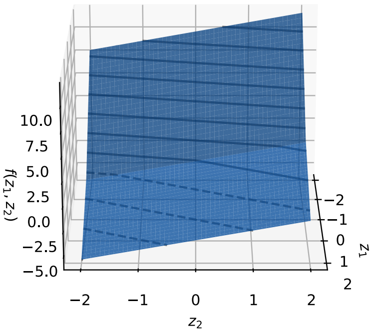
Differentiability

- A function $f(\vec{z})$ is **differentiable** if the **gradient** exists at every point.
- In other words, all of the slopes are well-defined:
  - $\partial f/\partial z_1, \partial f/\partial z_2, \ldots$

Example

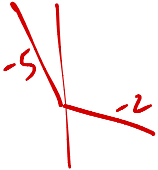
- $f(z_1, z_2) = \begin{cases} -5z_1 + z_2 & \text{if } z_1 \leq 0 \\ -2z_1 + z_2 & \text{if } z_1 > 0 \end{cases}$





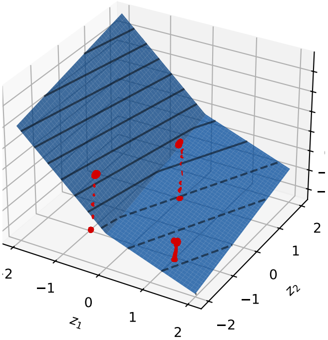
Exercise

What is the gradient at (-1, -1)? (1, -1)? (0, 1)?



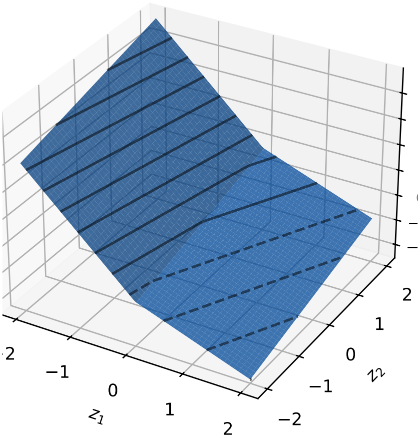
(-5, 1)   (-2, 1)   (?, 1)

$$f(z_1, z_2) = \begin{cases} -5z_1 + z_2 & \text{if } z_1 \leq 0 \\ -2z_1 + z_2 & \text{if } z_1 > 0 \end{cases}$$



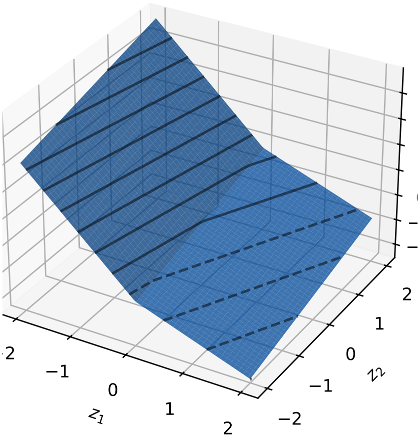
Answer

- $\frac{d}{d\vec{w}} f(\vec{z})$ is defined everywhere except along $z_1 = 0$.
- If $z_1 < 0$, $f(\vec{z}) = -5z_1 + z_2$.
  - gradient is $(-5, 1)^T$ here
- If $z_1 > 0$, $f(\vec{z}) = -2z_1 + z_2$.
  - gradient is $(-2, 1)^T$ here



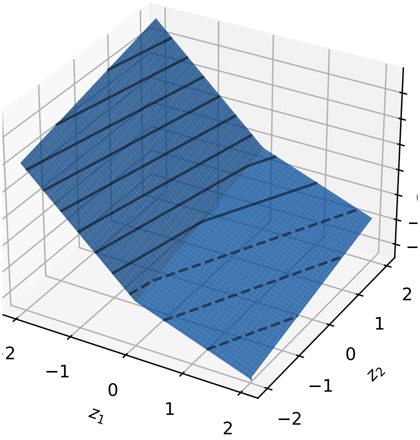
Answer

$$\frac{df}{d\vec{z}}(\vec{z}) = \begin{cases} (-5, 1)^T, & \text{if } z_1 < 0, \\ (-2, 1)^T, & \text{if } z_1 > 0, \\ \text{undefined}, & \text{if } z_1 = 0. \end{cases}$$



Problem

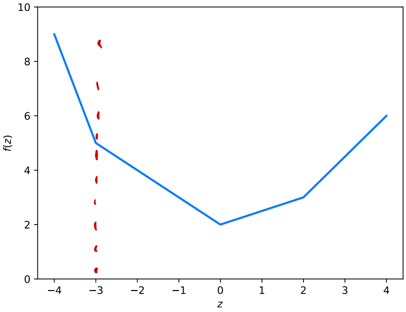
- We can try running gradient descent.
- But what do we do if we reach a point where the gradient is **not defined**?
- We need a **replacement** for the gradient that tells us where to go.



Idea

- Slope is undefined at $z_1 = -3$.
  - To the left, slope is -4
  - To the right, slope is -1

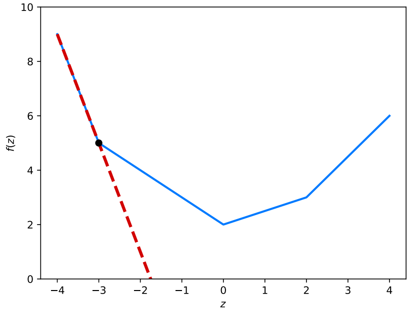
$$f(z) = \begin{cases} -4z - 7 & \text{if } z < -3 \\ -z + 2 & \text{if } -3 \le z < 0 \\ 0.5z + 2 & \text{if } 0 \le z < 2 \\ 3z/2 & \text{if } z \ge 2 \end{cases}$$



Idea

- Slope is undefined at $z_1 = -3$.
  - To the left, slope is -4
  - To the right, slope is -1

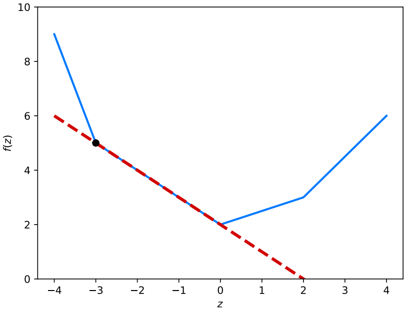
$$f(z) = \begin{cases} -4z - 7 & \text{if } z < -3 \\ -z + 2 & \text{if } -3 \le z < 0 \\ 0.5z + 2 & \text{if } 0 \le z < 2 \\ 3z/2 & \text{if } z \ge 2 \end{cases}$$



Idea

- Slope is undefined at $z_1 = -3$.
  - To the left, slope is -4
  - To the right, slope is -1

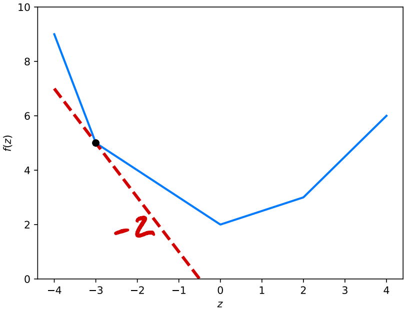
$$f(z) = \begin{cases} -4z - 7 & \text{if } z < -3 \\ -z + 2 & \text{if } -3 \leq z < 0 \\ 0.5z + 2 & \text{if } 0 \leq z < 2 \\ 3z/2 & \text{if } z \geq 2 \end{cases}$$



Idea

- Slope is undefined at $z_1 = -3$.
  - To the left, slope is -4
  - To the right, slope is -1

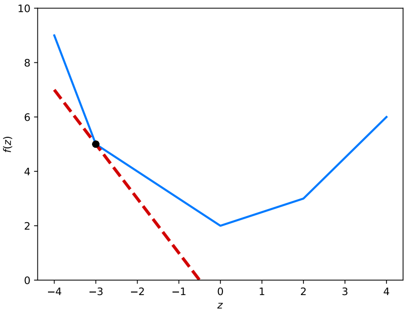
$$f(z) = \begin{cases} -4z - 7 & \text{if } z < -3 \\ -z + 2 & \text{if } -3 \le z < 0 \\ 0.5z + 2 & \text{if } 0 \le z < 2 \\ 3z/2 & \text{if } z \ge 2 \end{cases}$$



Idea

- Any number between -4 and -1 adequately describes the behavior of f at $z = -3$.

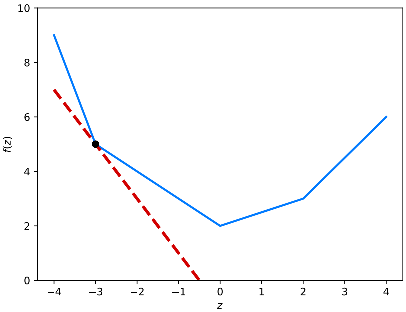
$$f(z) = \begin{cases} -4z - 7 & \text{if } z < -3 \\ -z + 2 & \text{if } -3 \leq z < 0 \\ 0.5z + 2 & \text{if } 0 \leq z < 2 \\ 3z/2 & \text{if } z \geq 2 \end{cases}$$



Idea

- Any number between -4 and -1 is a **subderivative** of f at $z = -3$.

$$f(z) = \begin{cases} -4z - 7 & \text{if } z < -3 \\ -z + 2 & \text{if } -3 \leq z < 0 \\ 0.5z + 2 & \text{if } 0 \leq z < 2 \\ 3z/2 & \text{if } z \geq 2 \end{cases}$$

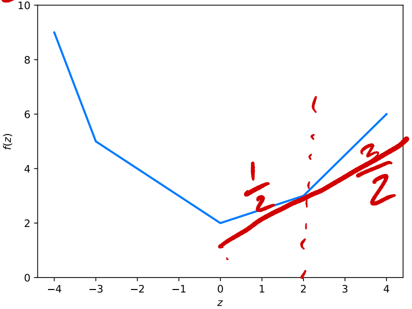


Exercise

What are the valid subderivatives of f at $z = 2$?

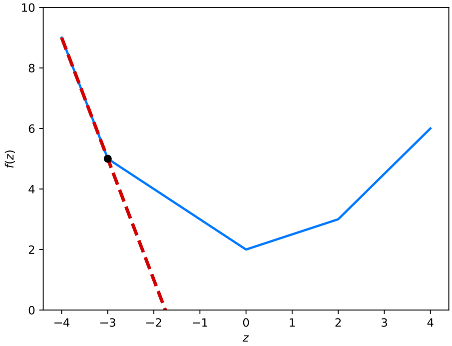
$\frac{1}{2}$, $\frac{3}{2}$, 1

$$f(z) = \begin{cases} -4z - 7 & \text{if } z < -3 \\ -z + 2 & \text{if } -3 \leq z < 0 \\ 0.5z + 2 & \text{if } 0 \leq z < 2 \\ 3z/2 & \text{if } z \geq 2 \end{cases}$$



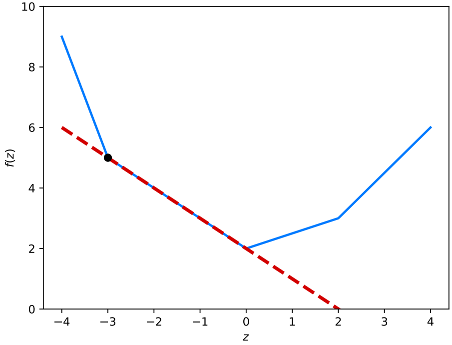
Subderivatives

- Any valid subderivative defines a line that lies below the function.



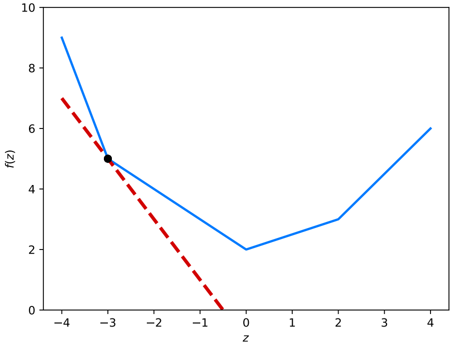
Subderivatives

- Any valid subderivative defines a line that lies below the function.



Subderivatives

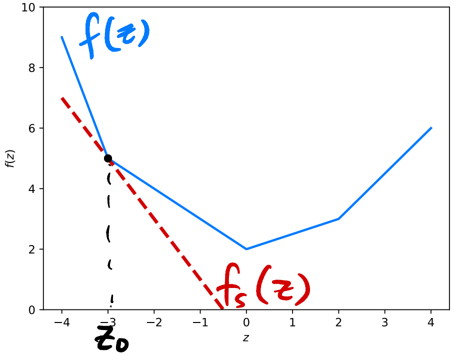
- Any valid subderivative defines a line that lies below the function.



Subderivatives

- The equation of this line is:

$$f_s(z) = f(z_0) + s(z - z_0)$$



Subderivatives

- A number s is a subderivative of f at z_0 if:

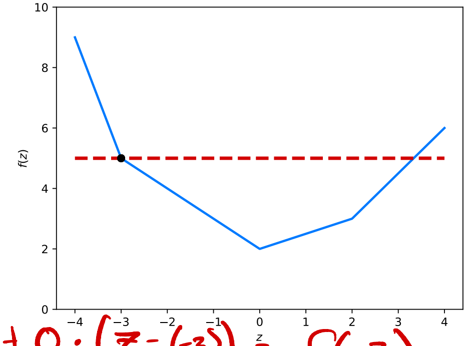
$$f(z) \geq f_s(z) \quad \text{for all } z$$

- That is, if:

$$f(z) \geq f(z_0) + s(z - z_0)$$

Exercise

Is 0 a valid subderivative of f at $z = 2$?



$$f_s(z) = f(-3) + 0 \cdot (z - (-3)) = f(-3)$$

Intuition

- The **subderivative** tells us how the function changes when the slope doesn't exist.
- We can sometimes use it in place of a derivative.

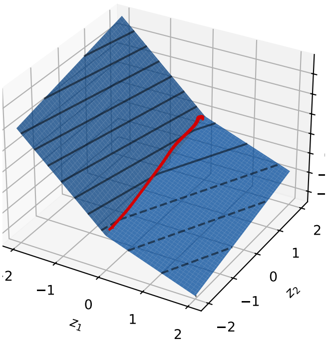
Subgradient

- In higher dimensions, we have multiple slopes to worry about.
- We can use a **subgradient** to generalize the concept of a subderivative.

Example

- There's no well-defined gradient at $z_1 = (0, 0)$.
  - The slope in the z_1 direction is undefined
  - Between -5 and -2?
  - The slope in the z_2 direction is 1

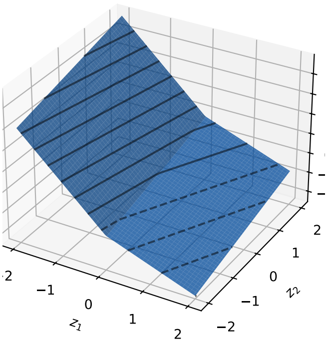
$$f(z_1, z_2) = \begin{cases} -5z_1 + z_2 & \text{if } z_1 \leq 0 \\ -2z_1 + z_2 & \text{if } z_1 > 0 \end{cases}$$



Example

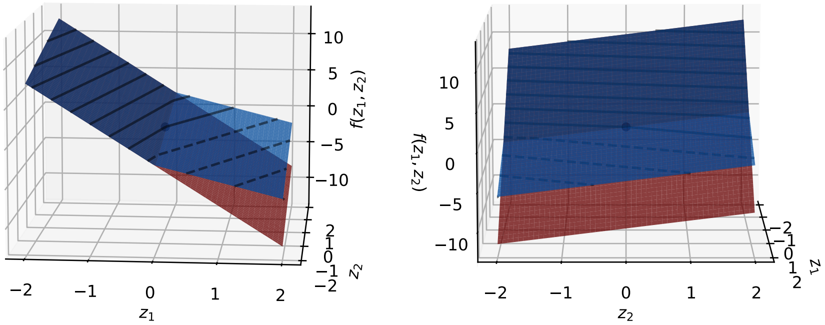
- We will call any vector $(s_1, 1)$ with $-5 \leq s_1 \leq -2$ a **subgradient** at $(0, 0)$.

$$f(z_1, z_2) = \begin{cases} -5z_1 + z_2 & \text{if } z_1 \leq 0 \\ -2z_1 + z_2 & \text{if } z_1 > 0 \end{cases}$$



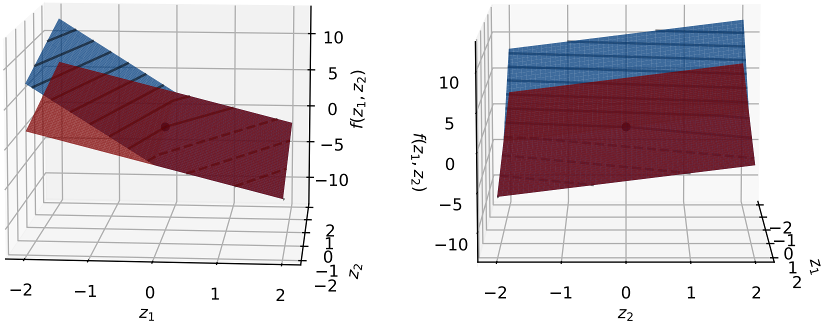
Subgradient

- A vector $\vec{s}$ defines a plane:
  - Example: $(-5, 1)^T$



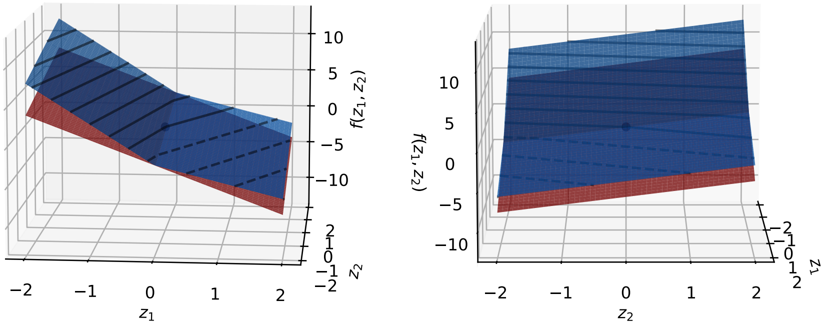
Subgradient

- A vector $\vec{s}$ defines a plane:
  - Example: $(-2, 1)^T$



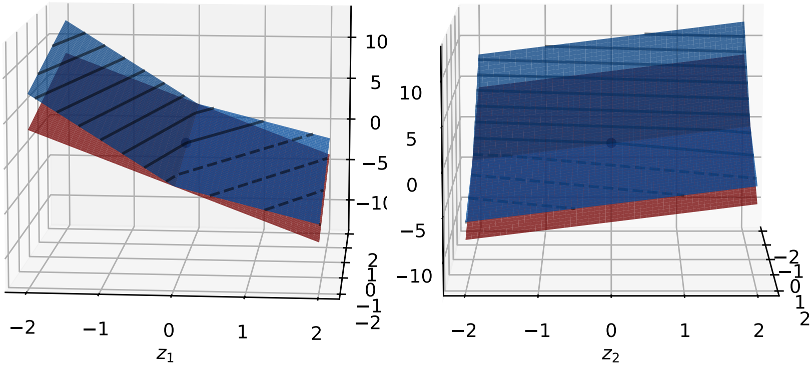
Subgradient

- A vector $\vec{s}$ defines a plane:
  - Example: $(-3, 1)^T$



Subgradient

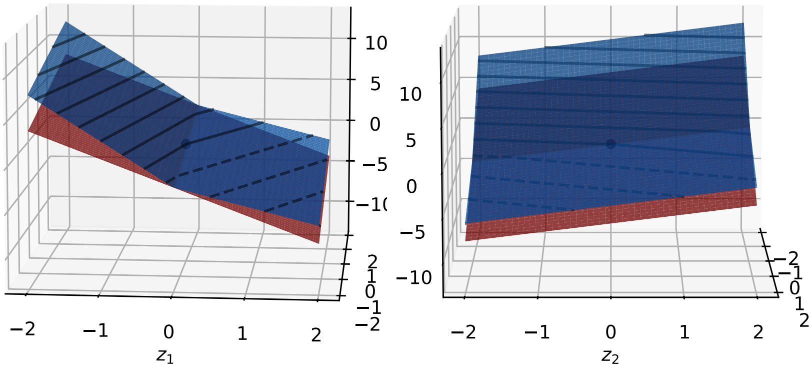
- A vector $\vec{s}$ is a valid **subgradient** at $\vec{z}^{(0)}$ if the plane it defines lies at or below the function f.
  - Example: $(-3, 1)^T$



Subgradient

- The equation of the plane defined by $\vec{s}$ at $\vec{z}^{(0)}$ is:

$$f_s(\vec{z}) = f(\vec{z}^{(0)}) + \vec{s} \cdot (\vec{z} - \vec{z}^{(0)})$$



Subgradients

- $\vec{s}$ is a **subgradient** of $f(\vec{z})$ at $\vec{z}^{(0)}$ if:

$$f(\vec{z}) \geq f_s(\vec{z}) \quad \text{for all } \vec{z}$$

- That is, if:

$$f(\vec{z}) \geq f(\vec{z}^{(0)}) + \vec{s} \cdot (\vec{z} - \vec{z}^{(0)})$$

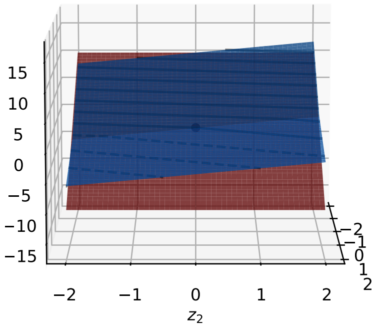
Finding Subgradients

- Here are two suggested ways to check that $\vec{s}$ is a valid subgradient.
- 1) Visualize it.
- 2) Check if the inequality holds.

Example

$$f(z_1, z_2) = \begin{cases} -5z_1 + z_2 & \text{if } z_1 \leq 0 \\ -2z_1 + z_2 & \text{if } z_1 > 0 \end{cases}$$

- Is $(-5, 0)^T$ a valid subgradient? *No.!*



$(-1,-1) \to f(-1,-1) = 5 - 1 = 4$

$\to f_s(-1,-1) = 5$

Example

$$f(z_1, z_2) = \begin{cases} -5z_1 + z_2 & \text{if } z_1 \le 0 \\ -2z_1 + z_2 & \text{if } z_1 > 0 \end{cases}$$

- Is $(-5, 0)^T$ a valid subgradient at the point (0,0)?

$\vec{s}$

- Is $f(0,0) + (-5,0)^T \cdot (z_1, z_2) \le f(z_1, z_2)$ for all z_1, z_2?

$0 - 5z_1 + 0z_2$

$f_s(\vec{z}) = -5z_1$

Tip

- If the slope is defined in a direction, the corresponding entry of the subgradient must be that slope.

Intuition

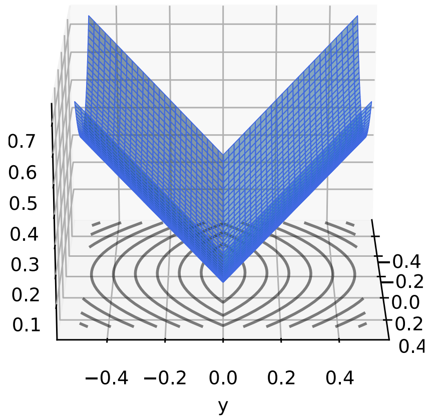
- A **subgradient** tells us where to go when the gradient is undefined.
- We can use it instead of the gradient in gradient descent.

Example

- $f(z_1, z_2) = z_1^2 + |z_2|$

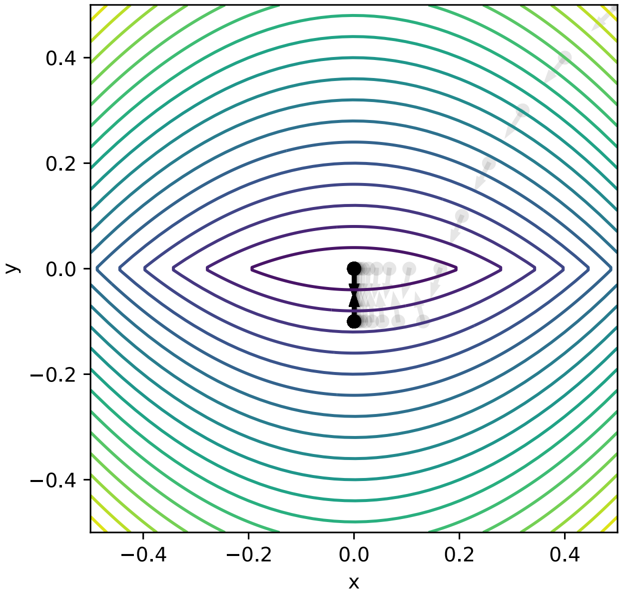
- A subgradient:

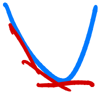
$$\vec{s}(z_1, z_2) = \begin{cases} (2z_1, 1)^T & \text{, if } z_2 > 0, \\ (2z_1, -1)^T & \text{, if } z_2 < 0, \\ (2z_1, 0)^T & \text{, if } z_2 = 0. \end{cases}$$



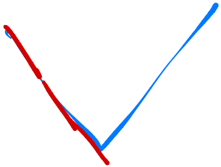
Example

- Subgradient descent on $f(z_1, z_2) = z_1^2 + |z_2|$
- Starting point: $(1/2, 1/2)^T$
- Learning rate: $\eta = 0.1$.


0.4
0.2
0.0
−0.2
−0.4
y
−0.4
−0.2
0.0
0.2
0.4
x



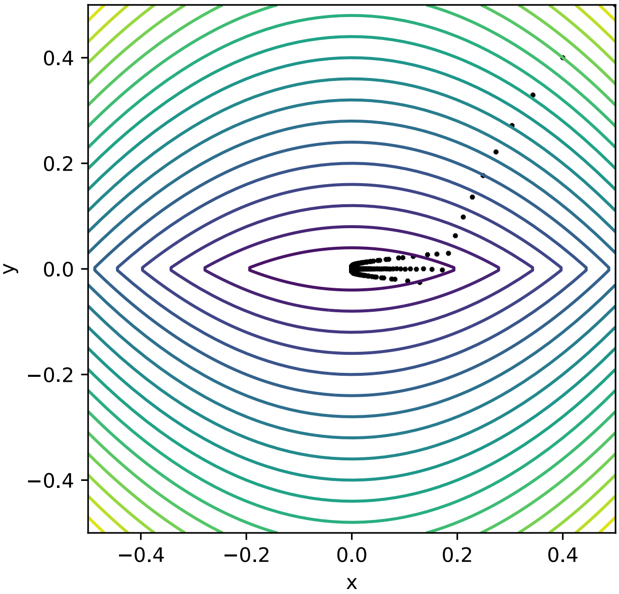
Problem



- Does not converge! Why?
- If f is differentiable, gradient gets smaller as we approach the minimum.
  - Naturally take smaller steps.
- Not true if the function is not differentiable!
  - Steps may stay the same size (too large).

Fix

- Decrease learning rate with each iteration.
- That is, choose a decreasing **learning rate schedule** $\eta(t) > 0$.
- **Theory:** choose $\eta(t) = c/\sqrt{t}$, where t is iteration #, c is a positive constant.


0.4
0.2
y
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
x

Subgradient Descent

To minimize $f(\vec{z})$:

- Pick arbitrary starting point $\vec{z}^{(0)}$, a decreasing **learning rate schedule** $\eta(t) > 0$.
- Until convergence, repeat:
  - **Compute a subgradient** $\vec{s}$ of f at $\vec{z}^{(i)}$.
  - Update $\vec{z}^{(t+1)} = \vec{z}^{(t)} - \eta(t)\vec{s}$
- When converged, return $\vec{z}^{(t)}$.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 4 | Part 6

Minimizing Absolute Loss

Regression with Absolute Loss

- The risk with respect to the absolute loss:

$$R(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n} |\vec{w} \cdot \mathrm{Aug}(\vec{x}^{(i)}) - y_i|$$

- We were stuck before because the risk is not differentiable.

- Now: we can minimize the risk with respect to the absolute loss using **subgradient descent**.

Subgradient of the Absolute Loss

- We need a subgradient of the absolute loss.

$$\ell_{\text{abs}}(\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}), y_i) = |\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i|$$

- If $\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) > y_i$:
    - Loss is $\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i$.
    - Gradient is $\text{Aug}(\vec{x}^{(i)})$.

Subgradient of the Absolute Loss

- We need a subgradient of the absolute loss.

$$\ell_{\text{abs}}(\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}), y_i) = |\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i|$$

- If $\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) < y_i$:
  - Loss is $y_i - \vec{w} \cdot \text{Aug}(\vec{x}^{(i)})$.
  - Gradient is $-\text{Aug}(\vec{x}^{(i)})$.

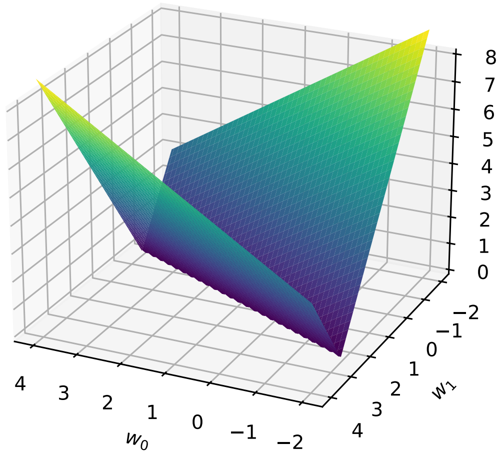
Subgradient of the Absolute Loss

- We need a subgradient of the absolute loss.

$$\ell_{\text{abs}}(\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}), y_i) = |\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i|$$

- If $\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) = y_i$:
  - We need a subgradient.

Subgradient of the Absolute Loss



Subgradient of the Absolute Loss

- The zero vector works as a subgradient.
- Our subgradient of the absolute loss:

$$s(\vec{w}; \vec{x}^{(i)}, y_i) = \begin{cases} \text{Aug}(\vec{x}^{(i)}), & \text{if } \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) > y_i, \\ -\text{Aug}(\vec{x}^{(i)}), & \text{if } \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) < y_i, \\ \vec{0}, & \text{if } \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) = y_i. \end{cases}$$

Minimizing the Absolute Loss

- The subgradient of the empirical risk is the average of the subgradients of the loss:

$$
\begin{aligned}
&\text{subgrad. of } R(\vec{w}) \\
&\quad = \frac{1}{n} \sum_{i=1}^{n} s(\vec{w}, \vec{x}^{(i)}, y_i) \\
&\quad = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} \operatorname{Aug}(\vec{x}^{(i)}), & \text{if } \vec{w} \cdot \operatorname{Aug}(\vec{x}^{(i)}) > y_i, \\ -\operatorname{Aug}(\vec{x}^{(i)}), & \text{if } \vec{w} \cdot \operatorname{Aug}(\vec{x}^{(i)}) < y_i, \\ \vec{0}, & \text{if } \vec{w} \cdot \operatorname{Aug}(\vec{x}^{(i)}) = y_i. \end{cases}
\end{aligned}
$$

Subgradient Descent

- We minimize the empirical risk with respect to the absolute loss using subgradient descent.

- Pick an initial $\vec{w}^{(0)}$, a decreasing learning rate schedule $\eta(t) > 0$.

- Until convergence, repeat:
  - Update

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta(t) \times \frac{1}{n} \sum_{i=1}^{n} \begin{cases} \text{Aug}(\vec{x}^{(i)}), & \text{if } \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) > y_i, \\ -\text{Aug}(\vec{x}^{(i)}), & \text{if } \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) < y_i, \\ \vec{0}, & \text{if } \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) = y_i. \end{cases}$$

In Practice

- Regression with absolute loss has different names:
  - Quantile regression
  - Minimum Absolute Deviations (MAD)
- Solvable by (S)GD, or as a linear program.

Next Time

- When is (S)GD guaranteed to converge?