

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**

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 modifications to 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 linear 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

- Recall¹ that the gradient of the MSE is:

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

- You code up a function:²

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

¹From Lecture 02, where we derived this.

²There'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 $X^T(X\vec{w} - \vec{y})$.

More is different...

- ▶ What if you had a **really big** 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**.

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.
- ▶ eg. Computing gradient involves a loop over data:

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

Problem

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

- ▶ 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

³Trivial: 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: Linear Least Squares

- ▶ We can use SGD to perform linear least squares regression.
- ▶ Need to compute the gradient of the square loss:

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

Exercise

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

Example: Linear Least Squares

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

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

Example: Linear Least Squares

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

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

- The update rule is:

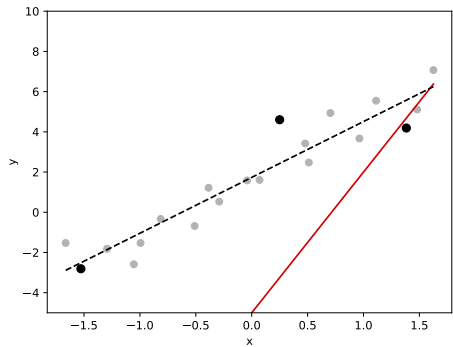
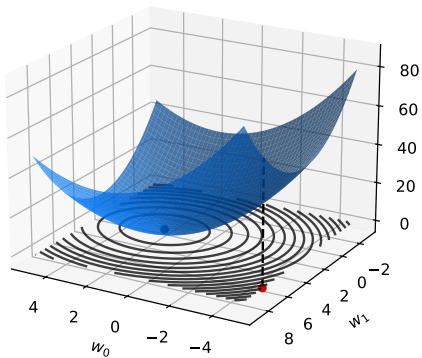
$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \vec{g}$$

Example: Linear Least Squares

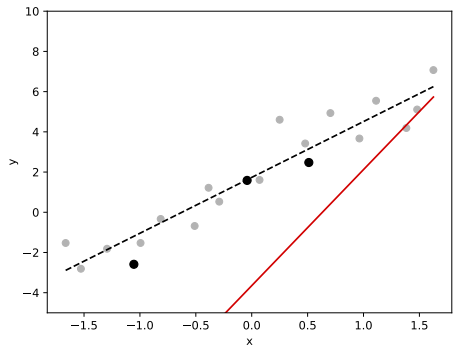
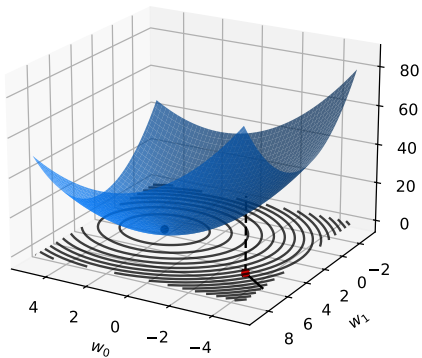
- ▶ We can write in matrix-vector form, too:
 - ▶ Let X_B be the design matrix using only the examples in batch B .
 - ▶ Let y_B be the corresponding vector of labels.
- ▶ Then:

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

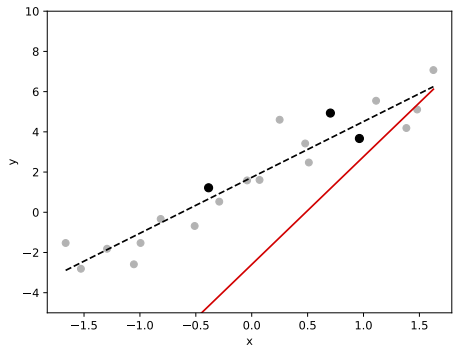
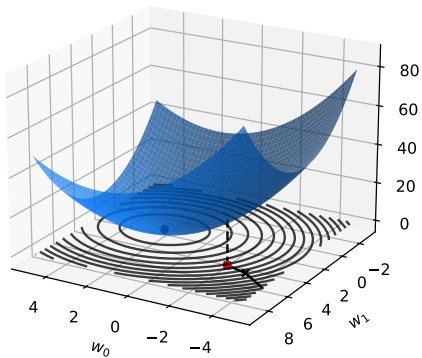
Example: SGD



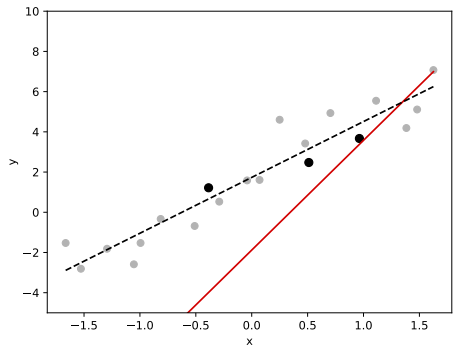
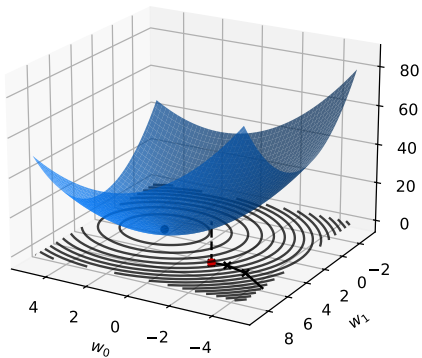
Example: SGD



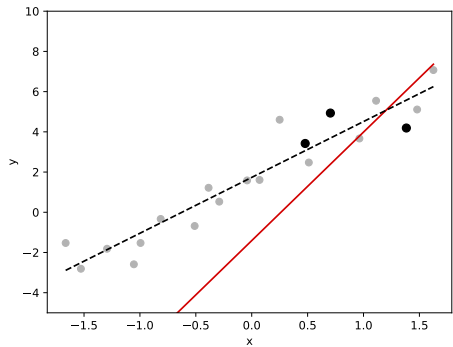
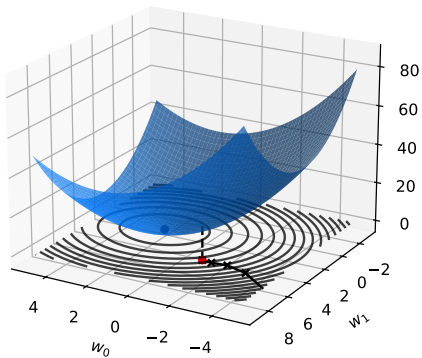
Example: SGD



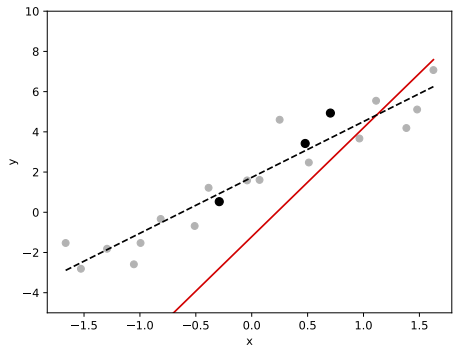
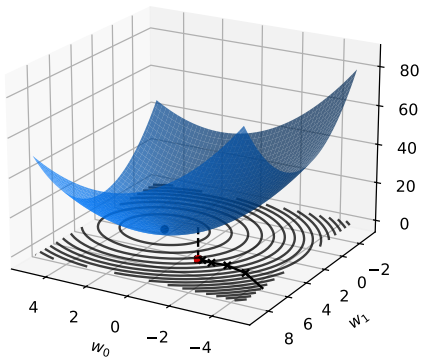
Example: SGD



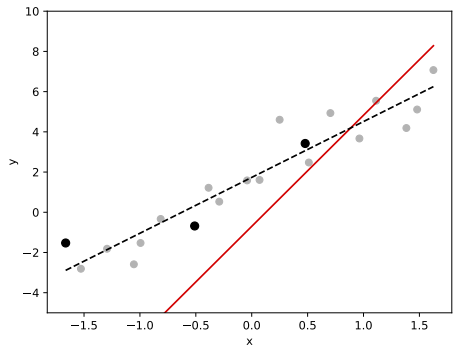
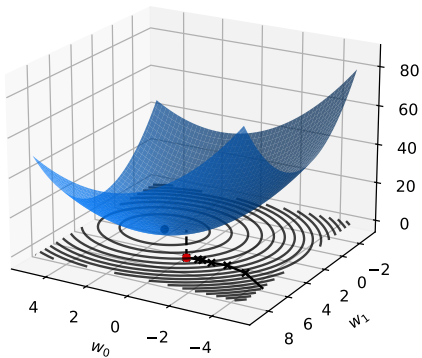
Example: SGD



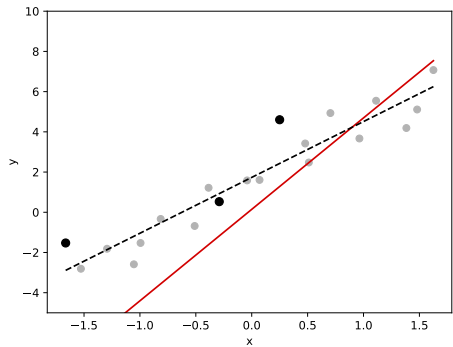
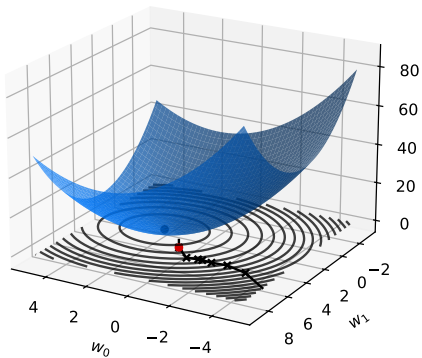
Example: SGD



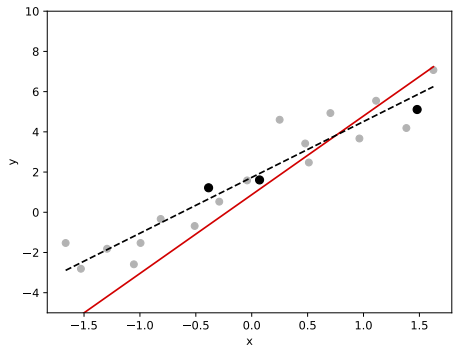
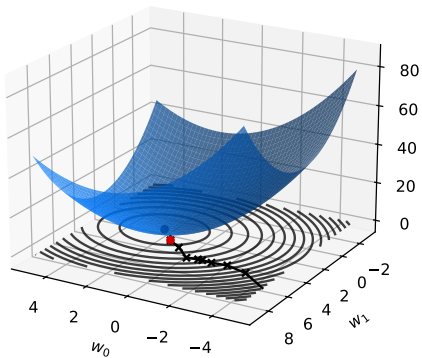
Example: SGD



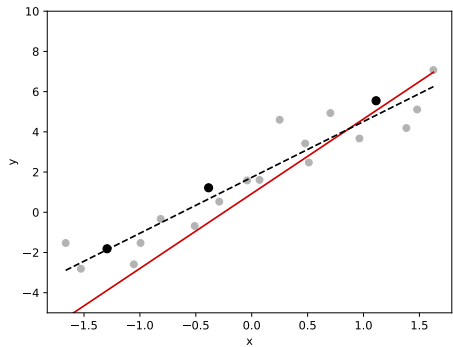
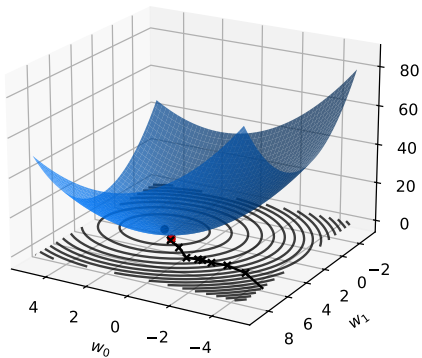
Example: SGD



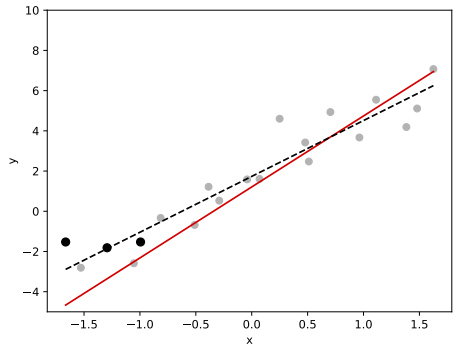
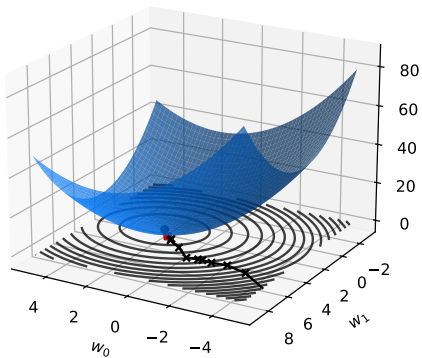
Example: SGD



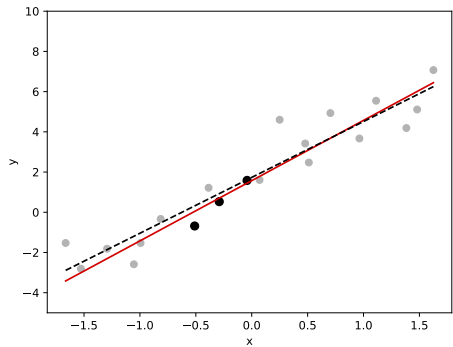
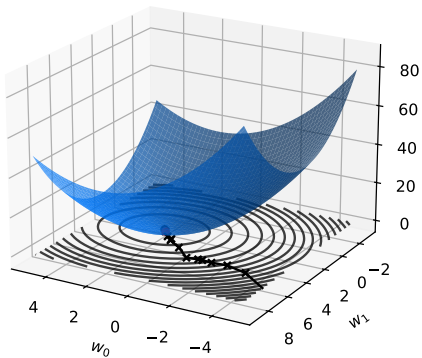
Example: SGD



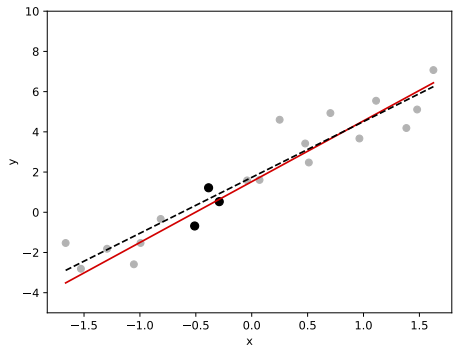
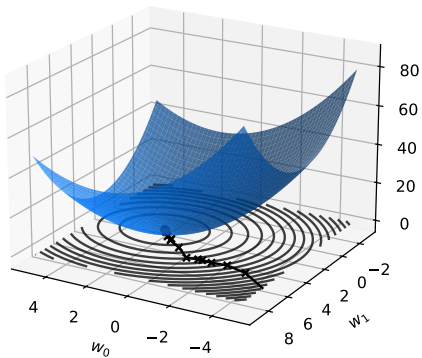
Example: SGD



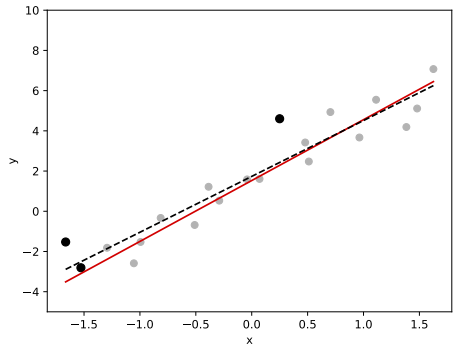
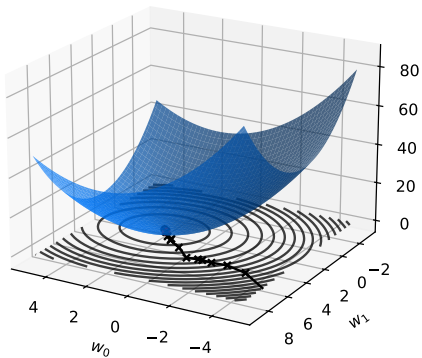
Example: SGD



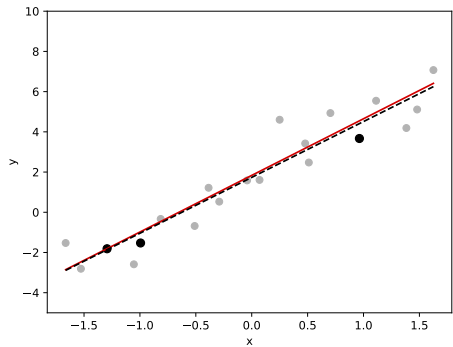
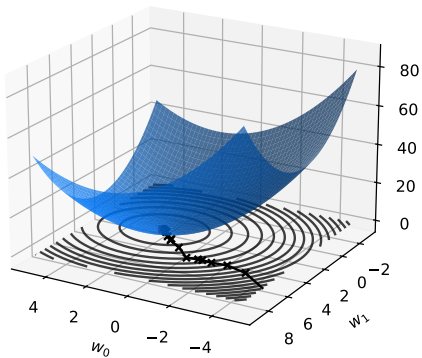
Example: SGD



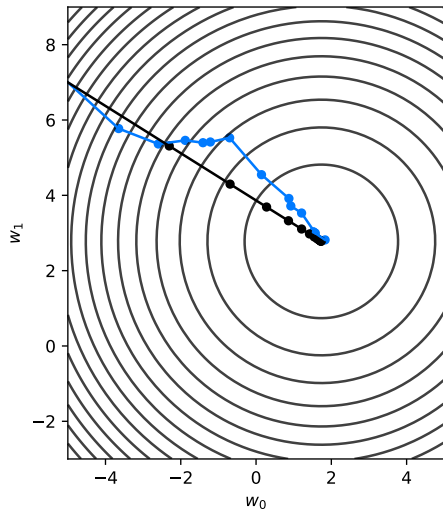
Example: SGD



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.
 - ▶ Partition 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.