Lecture 14: Least Mean Squares Regression

COMP 343, Spring 2022

Victoria Manfredi

WESLEYAN UNIVERSITY

Acknowledgements: These slides are based primarily on material from the book Machine Learning by Tom Mitchell (and associated slides), the book Machine Learning, An Applied Mathematics Introduction by Paul Wilmott, slides created by Vivek Srikumar (Utah), Dan Roth (Penn), Julia Hockenmaier (Illinois Urbana-Champaign), Jessica Wu (Harvey Mudd) and C. David Page (U of Wisconsin-Madison)

Today's Topics

Midterm

- Wednesday, March 30

Least Mean Squares (LMS) Regression

- Overview
- Objective
- Gradient descent
- Convergence
- Stochastic gradient descent
- Regularization

Recap

Learning as loss minimization

The setup

- Examples $\mathbf{x}$ with labels y drawn from a fixed, unknown distribution $P(X, Y)$
- Hidden oracle classifier f labels examples
- We wish to find a hypothesis h that mimics f

The ideal situation

- Define a function L that penalizes bad hypotheses
- **Learning**: pick a function $h \in H$ to minimize expected loss

$$\min_{h \in H} \sum_{(\mathbf{x}, y) \in D} L(f(\mathbf{x}), h(\mathbf{x})) P(X = \mathbf{x}, Y = y)$$

We cannot minimize this since we don't know $P(X, Y)$!

D is set of all possible $(\mathbf{x}, y)$ pairs

Empirical loss minimization

Learning = minimize empirical loss on the training set

$$\min_{h \in H} \frac{1}{m} \sum_{i=1:m} L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

where m is # of training examples $\{(\mathbf{x}_i y_i)\}_{i=1,m}$

Is there a problem here? Overfitting!

What is a loss function?

Loss functions should penalize mistakes

- We are minimizing average loss over the training data

$$\min_{h \in H} \frac{1}{m} \sum_{i=1:m} L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

What is the ideal loss function for classification?

- What loss function would you minimize if computation were free?

0-1 loss

Simplest loss function counts # of mistakes, aka 0-1 loss:

- Penalizes classification mistakes between true label y and prediction y'

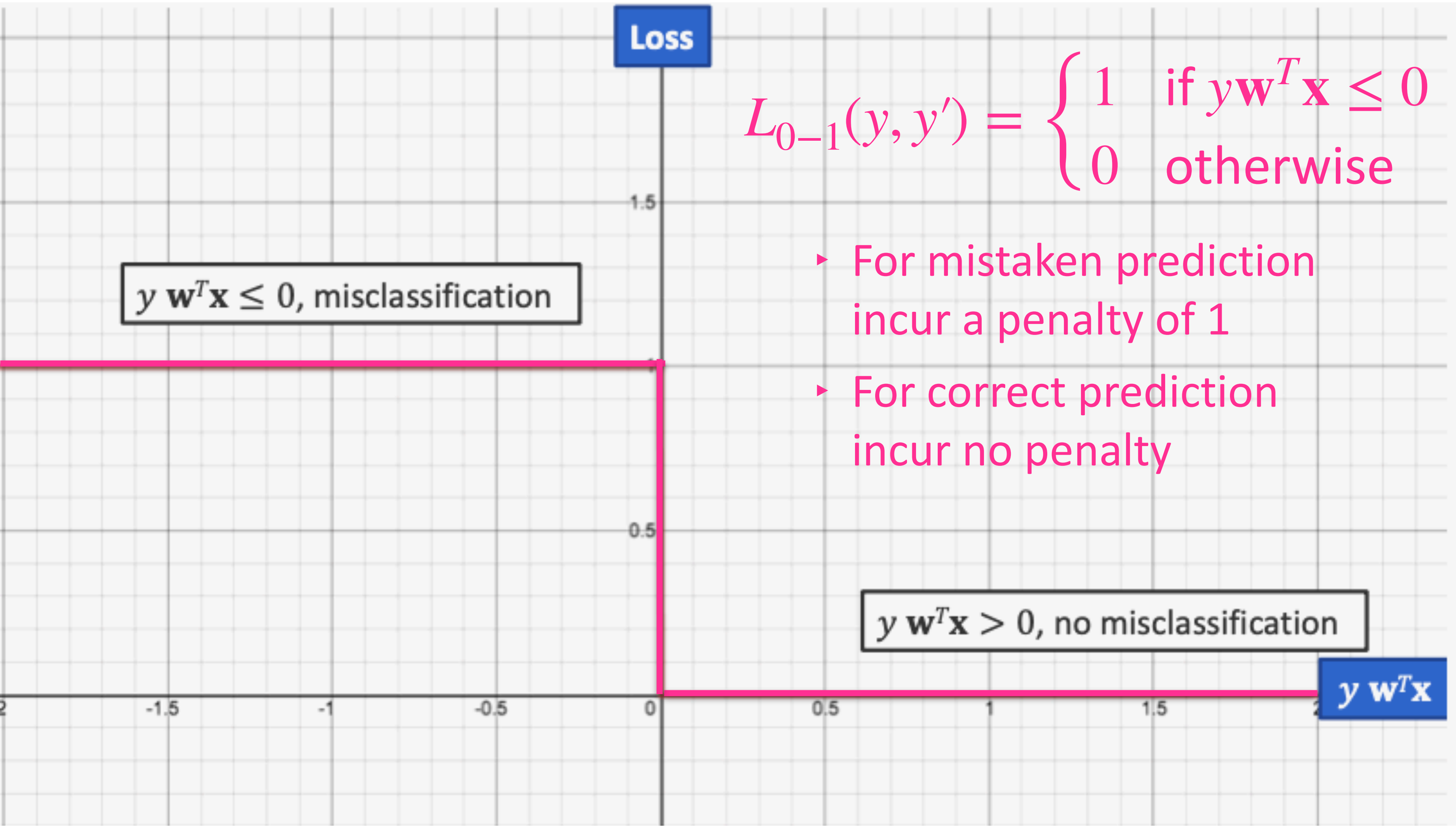
$$L_{0-1}(y, y') = \begin{cases} 1 & \text{if } y \neq y \\ 0 & \text{if } y = y \end{cases}$$

For linear classifiers the prediction is $y' = \text{sgn}(\mathbf{w}^T\mathbf{x})$

- Mistake if $y\mathbf{w}^T\mathbf{x} \leq \mathbf{0}$

$$L_{0-1}(y, y') = \begin{cases} 1 & \text{if } y\mathbf{w}^T\mathbf{x} \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

0-1 loss



$$L_{0-1}(y, y') = \begin{cases} 1 & \text{if } y\mathbf{w}^T\mathbf{x} \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

- For mistaken prediction incur a penalty of 1
- For correct prediction incur no penalty

Problems with 0-1 loss

- 0-1 loss incurs the same loss value of 1 for all wrong predictions, no matter how far a wrong prediction is from the hyperplane

- Knowing the current value of loss function doesn't indicate how to modify current weights to further improve

- Computationally intractable to minimize 0-1 loss function: with m examples need to check 2^m permutations (since can't minimize directly)

- What can we do? Make some assumptions: minimize "surrogate function" that is good enough

A solution of sorts

Minimize a new function, a convex upper bound of classification error

Perceptron loss

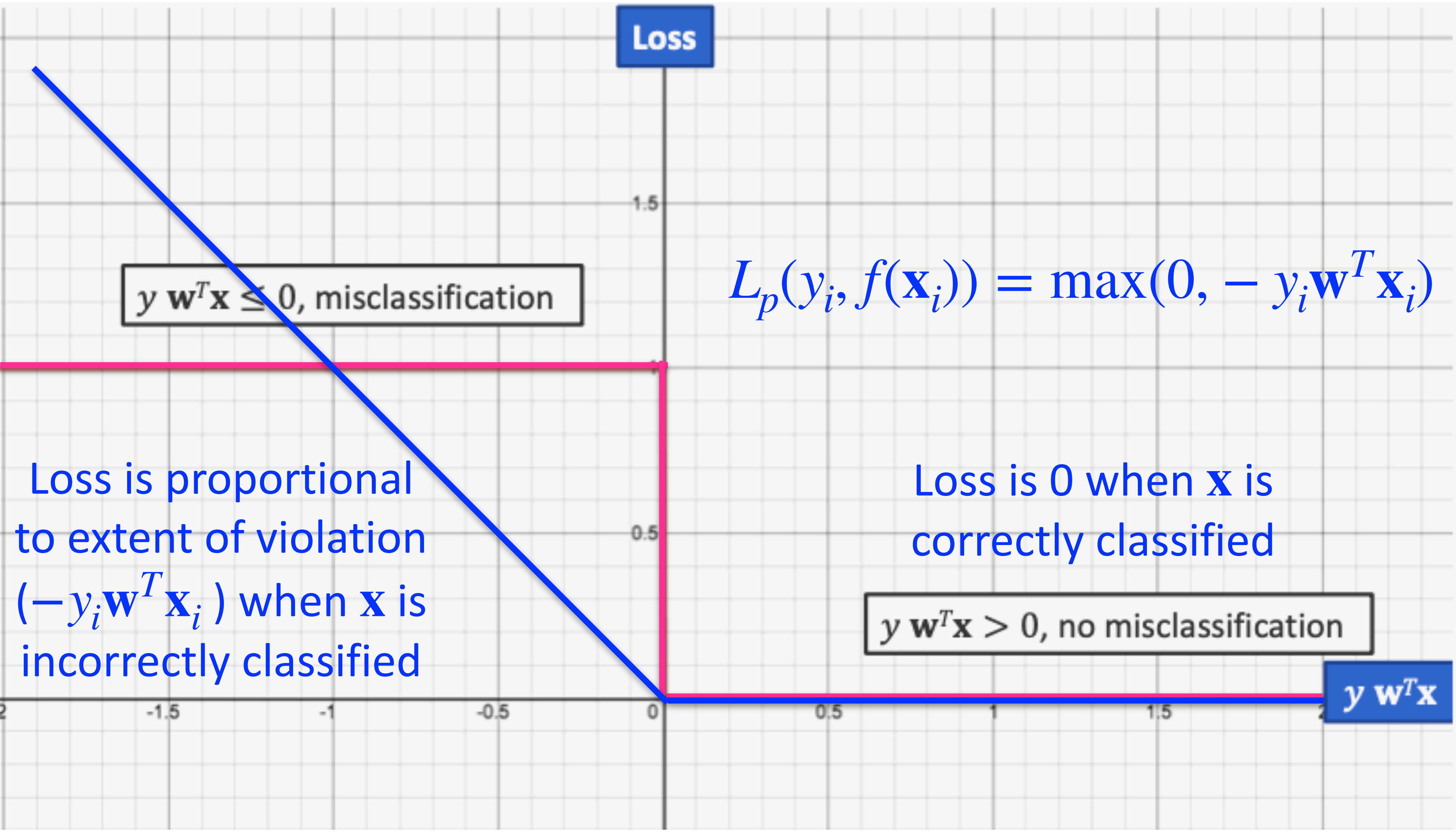
- Penalizes each wrong prediction by extent of violation
- The perceptron loss function is defined as

$$\frac{1}{m}\sum_{i=1:m} L_P(y_i, f(\mathbf{x}_i))$$

where $L_p(y_i, f(\mathbf{x}_i)) = \max(0, -y_i\mathbf{w}^T\mathbf{x}_i)$

- Note: max of constant and linear function is convex function

Perceptron loss



Square loss

$y \in \{-1,1\}$

The square loss function is commonly used for regression problems

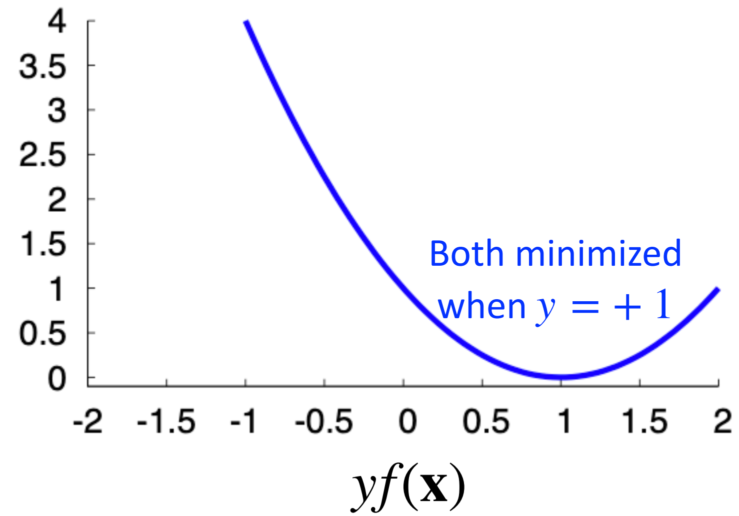
$$L_S(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$$

Square loss can also be used for binary classification problems

$$L_S(y, f(\mathbf{x})) = (1 - yf(\mathbf{x}))^2$$

Square loss tends to penalize wrong predictions excessively

Square loss as a function of $yf(\mathbf{x})$



What is our goal?

Most machine learning algorithms are some combination of a loss function + an algorithm for finding a local minimum.

Our goal is to **minimize the loss function**

- E.g., find a set of values for $\mathbf{w}$ for which the loss from the loss function is a minimum

A continuous convex loss function allows a simpler optimization algorithm: can take derivative of function

How to minimize a loss function?

Goal: find weights for which loss is (global) minimum

Remember, the derivative of a function is zero at any local maximum or minimum. So one way to find a minimum is to set $f'(\mathbf{w}) = 0$ and solve for $\mathbf{w}$. But for most functions, there isn't a way to solve this

Instead use gradient descent: algorithmically search different values of $\mathbf{w}$ until you find one that results in a gradient near 0. More on this next class

Bias

Every learning algorithm requires assumptions about the hypothesis space.

Eg: "My hypothesis space is

- ...linear"
- ...decision trees with 5 nodes"
- ...a three layer neural network with rectifier hidden units"

It's possible given the set of assumptions made, no function in that space is good enough. What is error of the best classifier within your hypothesis set? You cannot do better than that

Bias is the true error (loss) of the *best* predictor in the hypothesis set

Suppose hypothesis space cannot represent a function. What will the bias be?

- Bias will be nonzero, possibly high

Underfitting: when bias is high

Variance

What if the performance of a classifier is dependent on specific training set we have?

- Perhaps the model will change if we slightly change the training set

Variance describes how much the best classifier depends on the training set

- Increases when classifiers become more complex: with sufficient complexity classifier can just remember training set
- Decreases with larger training sets: probability of overfitting a massive training set is low

Overfitting: high variance

Let’s play darts

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

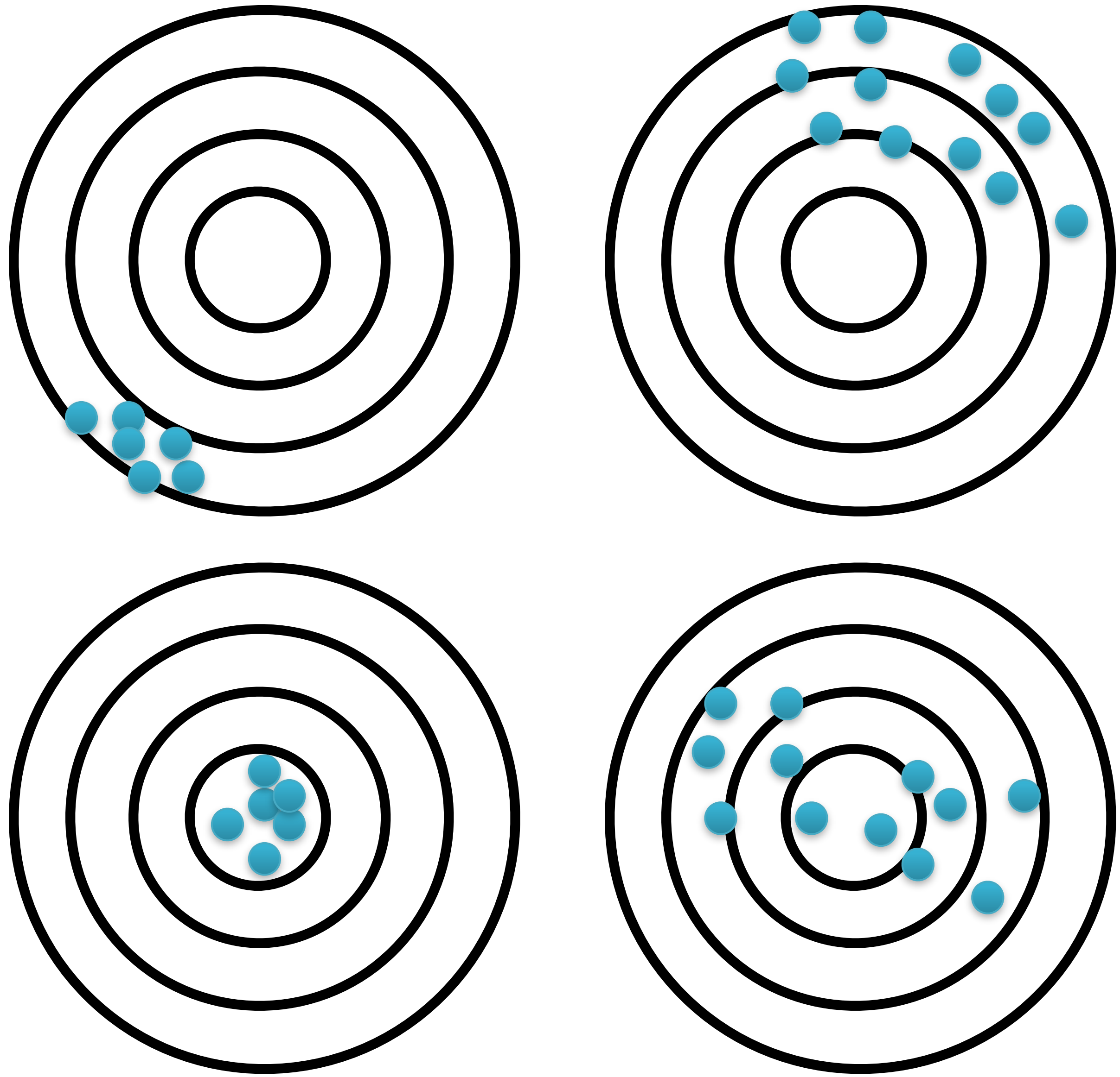
High bias

Hypothesis cannot represent target function

Low bias

Hypothesis can represent target function

Each dot is a model that is learned from a a different dataset



Low variance | **High variance**

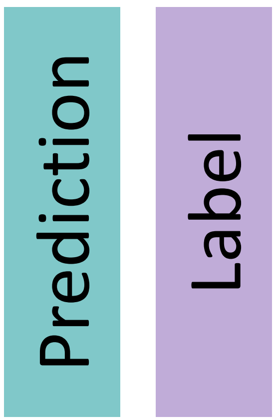
How much does classifier depend on training data?

Evaluation

OTHER METRICS

Evaluation metrics

To evaluate model, compare predicted labels to actual

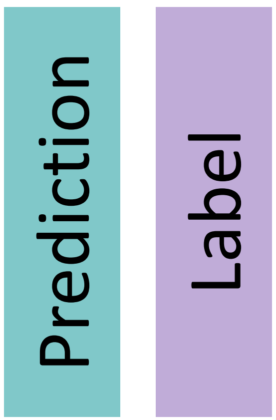


Accuracy: proportion of examples where we predicted correct label

$$\text{accuracy} = \frac{\text{\# of correct predictions}}{\text{\# of examples}}$$

Evaluation metrics

To evaluate model, compare predicted labels to actual



Accuracy: proportion of examples where we predicted correct label

$$\text{accuracy} = \frac{\#\text{ of correct predictions}}{\#\text{ of examples}}$$

Error: proportion of examples where we predicted incorrect label

$$\text{error} = 1 - \text{accuracy}$$

$$\text{accuracy} = \frac{\#\text{ of incorrect predictions}}{\#\text{ of examples}}$$

Problem

Accuracy or error may not always be the right way to measure performance of classifier …

Suppose # of examples for one class is very different than # of examples for another class

E.g.,

- 99 examples are positive
- 1 example is negative

Problem

Accuracy or error may not always be the right way to measure performance of classifier …

Suppose # of examples for one class is very different than # of examples for another class

E.g.,

- 99 examples are positive
- 1 example is negative

Just predicting positive for every example gives 99% accuracy!

Another problem

Accuracy or error may not always be the right way to measure performance of classifier …

Suppose there are differential misclassification costs – say, getting a positive wrong costs more than getting a negative wrong

- Consider a medical domain in which a false positive results in an extraneous test but a false negative results in a failure to treat a disease

The beginnings of a solution

Let's separately ask what is the performance of classifier on positive labels and what is performance on negative labels.

Or more generally, let's ask what is the performance on each class of labels separately

Example

Suppose dataset has two classes of labels, A and B

- 10 examples truly labeled A
- 90 examples truly labeled B
- Classifier predicts A for 8 examples.

Example

Suppose dataset has two classes of labels, A and B

- 10 examples truly labeled A
- 90 examples truly labeled B
- Classifier predicts A for 8 examples.

What fraction of class A examples did the classifier discover?

- 10 examples are truly labeled A
- 8 of truly labeled A examples found by classifier

Example

Suppose dataset has two classes of labels, A and B

- 10 examples truly labeled A
- 90 examples truly labeled B
- Classifier predicts A for 8 examples.

What fraction of class A examples did the classifier discover?

- 10 examples are truly labeled A
- 8 of truly labeled A examples found by classifier

Recall is 80%

Example

Suppose dataset has two classes of labels, A and B

- 10 examples truly labeled A
- 90 examples truly labeled B
- Classifier predicts A for 8 examples.

What fraction of class A examples did the classifier discover?

- 10 examples are truly labeled A
- 8 of truly labeled A examples found by classifier

Recall is 80%

What fraction of classifier's predictions of class A were correct

- 10 examples are truly labeled A
- 80 examples are labeled A by classifier

Example

Suppose dataset has two classes of labels, A and B

- 10 examples truly labeled A
- 90 examples truly labeled B
- Classifier predicts A for 8 examples.

What fraction of class A examples did the classifier discover?

- 10 examples are truly labeled A
- 8 of truly labeled A examples found by classifier

Recall is 80%

What fraction of classifier's predictions of class A were correct

- 10 examples are truly labeled A
- 80 examples are labeled A by classifier

Precision is 10%

Precision-Recall analysis

What fraction of class "label" examples did the classifier discover?

$$\text{Recall(label)} = \frac{\text{Correct predictions(label)}}{\text{Correct predictions(label)} + \text{Missed examples(label)}}$$

What fraction of classifier's predictions of class "label" were correct

$$\text{Precision(label)} = \frac{\text{Correct predictions(label)}}{\text{Correct predictions(label)} + \text{Incorrect predictions(label)}}$$

Precision-Recall analysis

What fraction of class "label" examples did the classifier discover?

$$\text{Recall(label)} = \frac{\text{Correct predictions(label)}}{\text{Correct predictions(label)} + \text{Missed examples(label)}}$$

What fraction of classifier's predictions of class "label" were correct

$$\text{Precision(label)} = \frac{\text{Correct predictions(label)}}{\text{Correct predictions(label)} + \text{Incorrect predictions(label)}}$$

By default, precision and recall computed for the positive label, as that is usually the case of interest and the one usually with fewer example (e.g., diagnosing diseases in patients, identifying spam emails)

Combining into one number

Sometimes easier to work with a single number as performance measure

F1 score balances precision and recall: harmonic mean of precision and recall

$$f_1 = \frac{2pr}{p + r}$$

Training to minimize F1 is difficult, but can choose hyper parameters for which F1 is maximized

Practical advice

What measure to use depends on domain and how balanced your dataset is

All labels matter and dataset is balanced:

- Use accuracy

Not all labels matter or dataset not balanced:

- Use precision, recall, F1

Material up to this point will be on exam

LMS regression

OVERVIEW

Overview

Least squares method for regression

- Examples
- The LMS objective: frame learning as mathematical optimization
- Gradient descent algorithm to do this optimization
- Incremental/stochastic gradient descent algorithm

Problem

Suppose we want to predict the mileage of a car from its weight and age

Weight (x 100 lb) x_1	Age (years) x_2	Mileage
31.5	6	21
36.2	2	25
43.1	0	18
27.6	2	30

Problem

Suppose we want to predict the mileage of a car from its weight and age

Weight (x 100 lb) x_1	Age (years) x_2	Mileage
31.5	6	21
36.2	2	25
43.1	0	18
27.6	2	30

What we want: a function that can predict mileage using x_1 and x_2

What kind of supervised learning problem do we have?

Problem

Suppose we want to predict the mileage of a car from its weight and age

Weight (x 100 lb) x_1	Age (years) x_2	Mileage
31.5	6	21
36.2	2	25
43.1	0	18
27.6	2	30

What we want: a function that can predict mileage using x_1 and x_2

Regression problem: output is a real number

Example regression problems

Predict housing price from house size, lot size, rooms, neighborhood, ….

Predict life expectancy increase from medication, disease state, …

Predict crop yield from precipitation, fertilizer, temperature, …

Linear regression: The strategy

Predict continuous values using a linear model

Assumption: the output is a linear function of the inputs

Mileage = $w_0 + w_1 x_1 + w_2 x_2$

Restricting hypothesis space!

Among all possible functions that take x_1 and x_2 and produce a real number, only consider functions with this shape

Linear regression: The strategy

Predict continuous values using a linear model

Assumption: the output is a linear function of the inputs

Mileage = $\underline{w_0} + \underline{w_1} x_1 + \underline{w_2} x_2$

Parameters of the model, also called weights. Collectively, a vector $\mathbf{w}$

Similar to what we assumed with linear classifier, but no thresholding here!

Linear regression: The strategy

Predict continuous values using a linear model

Assumption: the output is a linear function of the inputs

$$\text{Mileage} = \underline{w_0} + \underline{w_1} x_1 + \underline{w_2} x_2$$

Parameters of the model, also called weights. Collectively, a vector $\mathbf{w}$

What is the goal of learning now?

Linear regression: The strategy

Predict continuous values using a linear model

Assumption: the output is a linear function of the inputs

Mileage = $\underline{w_0} + \underline{w_1} x_1 + \underline{w_2} x_2$

Parameters of the model, also called weights. Collectively, a vector $\mathbf{w}$

Learning: use the training data to find the ***best*** possible value of $\mathbf{w}$

Linear regression: The strategy

Predict continuous values using a linear model

Assumption: the output is a linear function of the inputs

Mileage = $\underline{w_0} + \underline{w_1} x_1 + \underline{w_2} x_2$

Parameters of the model, also called weights. Collectively, a vector $\mathbf{w}$

Learning: use the training data to find the ***best*** possible value of $\mathbf{w}$

How do we do prediction now?

Linear regression: The strategy

Predict continuous values using a linear model

Assumption: the output is a linear function of the inputs

Mileage = $\underline{w_0} + \underline{w_1} x_1 + \underline{w_2} x_2$

Parameters of the model, also called weights. Collectively, a vector $\mathbf{w}$

Learning: use the training data to find the ***best*** possible value of $\mathbf{w}$

Prediction: given the values for x_1, x_2 for a new car, use the learned $\mathbf{w}$ to predict the Mileage for the new car

Linear regression: More formally

Inputs are feature vectors: $\mathbf{x} \in \Re^d$

Outputs are real numbers: $y \in \Re$

For simplicity, we will assume that the first feature is always 1, to make notation easier

$$\mathbf{x}_i = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

To avoid special treatment of w_1

Linear regression: More formally

Inputs are feature vectors: $\mathbf{x} \in \Re^d$

Outputs are real numbers: $y \in \Re$

For simplicity, we will assume that the first feature is always 1, to make notation easier

We have a training data set:

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_d, y_d)\}$$

Labels are real numbers now

$$\mathbf{x}_i = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

Linear regression: More formally

Inputs are feature vectors: $\mathbf{x} \in \Re^d$

Outputs are real numbers: $y \in \Re$

For simplicity, we will assume that the first feature is always 1, to make notation easier

We have a training data set:

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_d, y_d)\}$$

$$\mathbf{x}_i = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

We want to approximate y as

$$y = w_1 + w_2 x_2 + \cdots + w_d x_d$$

$$y = \mathbf{w}^T \mathbf{x}$$

$\mathbf{w}$ is the learned weight vector in $\Re^d$

Making assumption that output y is a linear function of the features $\mathbf{x}$

Linear regression: More formally

Inputs are feature vectors: $\mathbf{x} \in \Re^d$

Outputs are real numbers: $y \in \Re$

For simplicity, we will assume that the first feature is always 1, to make notation easier

We have a training data set:

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_d, y_d)\}$$

$$\mathbf{x}_i = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

We want to approximate y as

$$y = w_1 + w_2 x_2 + \cdots + w_d x_d$$

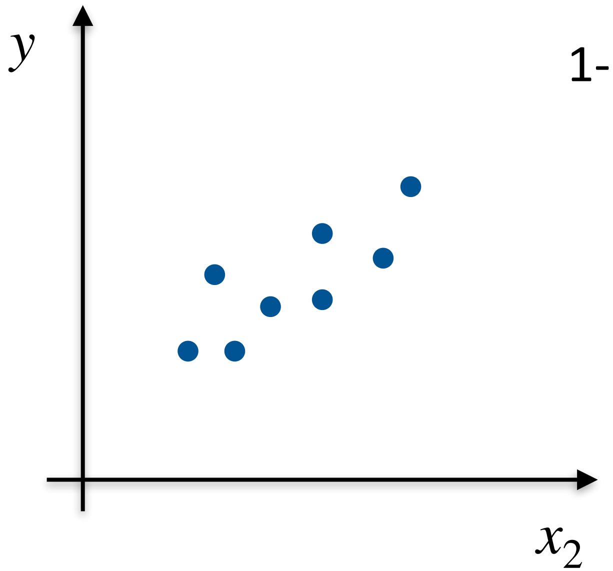
$$y = \mathbf{w}^T \mathbf{x}$$

$\mathbf{w}$ is the learned weight vector in $\Re^d$

Making assumption that output y is a linear function of the features $\mathbf{x}$

This is general setup for linear regression

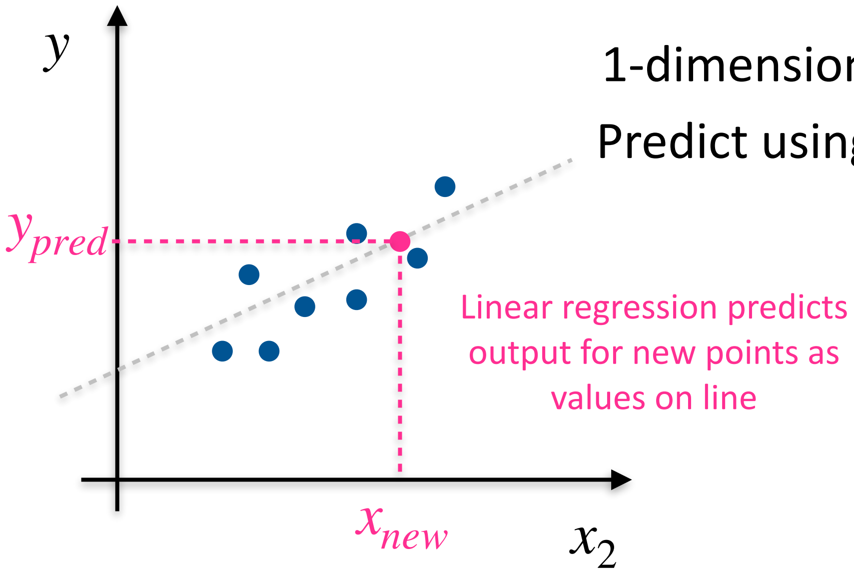
Examples



1-dimensional input

Suppose you have a set of points ….

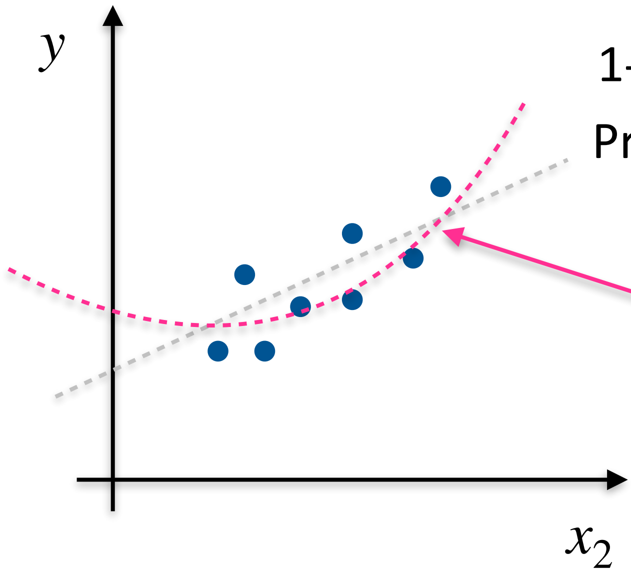
Examples



1-dimensional input

Predict using $y = w_1 + w_2 x_2$

Examples



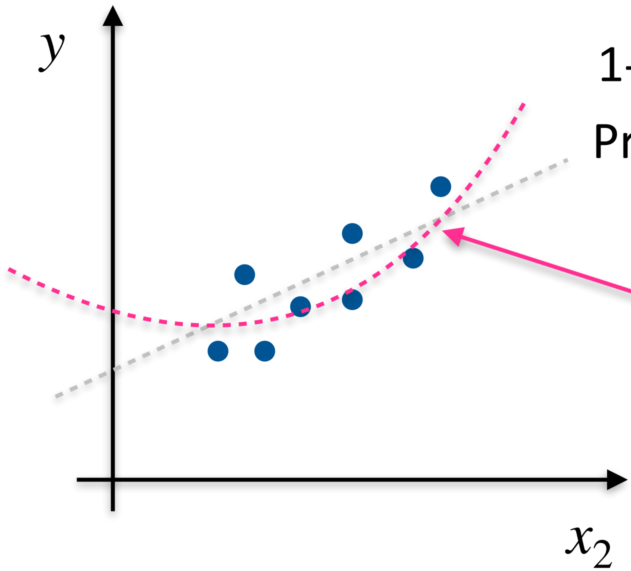
1-dimensional input

Predict using $y = w_1 + w_2 x_2$

Keep in mind, linear function is not our only choice. We could have tried to fit the data as another polynomial

What is the problem with using increasingly complicated functions here?

Examples



1-dimensional input

Predict using $y = w_1 + w_2 x_2$

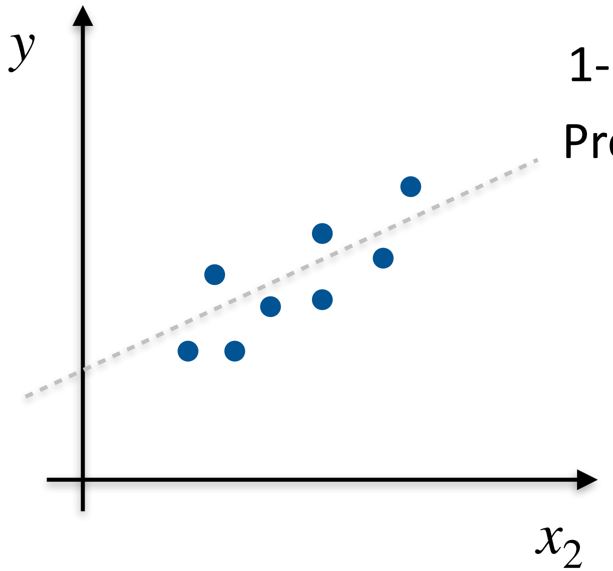
Keep in mind, linear function is not our only choice. We could have tried to fit the data as another polynomial

What is the problem with using increasingly complicated functions here?

May fit the data perfectly, but also fits the noise $\Rightarrow$ bad generalization

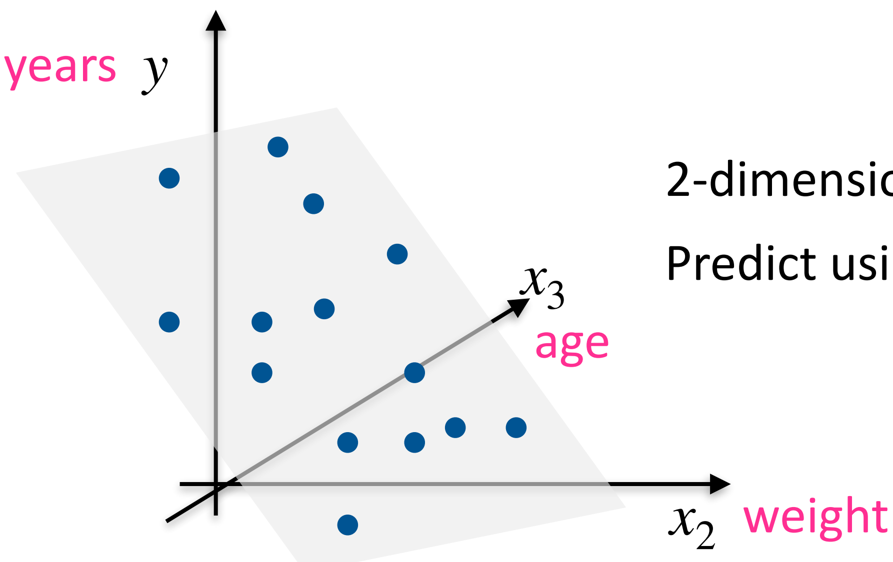
Using linear functions is a hypothesis choice, may not always be best

Examples



1-dimensional input

Predict using $y = w_1 + w_2 x_2$



2-dimensional input

Predict using $y = w_1 + w_2 x_2 + w_3 x_3$

Linear Regression

OBJECTIVE

If our hypothesis space is linear functions …

How do we know which weight vector is best one for a training set?

Or even, what makes a weight vector a good one

.

x_2

If our hypothesis space is linear functions …

How do we know which weight vector is best one for a training set?

For an input $(\mathbf{x}_i, y_i)$ in the training set, the *cost* of a mistake is

$$|\, y_i - \mathbf{w}^T \mathbf{x}_i \,|$$

True output

Predicted output

How far apart is true from predicted in absolute sense?

If very different then weight vector is probably not very good

If our hypothesis space is linear functions …

How do we know which weight vector is best one for a training set?

For an input $(\mathbf{x}_i, y_i)$ in the training set, the *cost* of a mistake is

$$| y_i - \mathbf{w}^T \mathbf{x}_i |$$

True output

Predicted output

How far apart is true from predicted in absolute sense?

If very different then weight vector is probably not very good

$$| y_i - w_1^T x_i | = 60000 \qquad | y_{i+1} - w_1^T x_{i+1} | = 0.1$$
$$| y_i - w_2^T x_i | = 0.1 \qquad | y_{i+1} - w_2^T x_{i+1} | = 0.3$$

But could also be that weight vector is just bad for that example

How do we decide whether weight vector is good?

How do we know which weight vector is best one for a training set?

For an input $(\mathbf{x}_i, y_i)$ in the training set, the *cost* of a mistake is

$$|y_i - \mathbf{w}^T\mathbf{x}_i|$$

This tells us how good for one example

Define the cost (or *loss*) for a particular weight vector $\mathbf{w}$ to be

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\underline{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}$$

This tells us how good for m examples

Squared error is a popular loss function: sum of squared costs over the training set. Dividing by 2 rather than m will make our math work out nicely later

How do we decide whether weight vector is good?

How do we know which weight vector is best one for a training set?

For an input $(\mathbf{x}_i, y_i)$ in the training set, the *cost* of a mistake is

$$|y_i - \mathbf{w}^T \mathbf{x}_i|$$

Define the cost (or *loss*) for a particular weight vector $\mathbf{w}$ to be

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Function of functions

J is a function that evaluates how good other functions or regressors are, e.g., $\mathbf{w}^T \mathbf{x}$. Every choice of $\mathbf{w}$ gives a different regressor. So J evaluates how good a regressor is.

$$J(\mathbf{f}) = \frac{1}{2} \sum_{i=1}^{m} (y_i - f(\mathbf{x}_i))^2$$

How do we decide whether weight vector is good?

How do we know which weight vector is best one for a training set?

For an input $(\mathbf{x}_i, y_i)$ in the training set, the *cost* of a mistake is

$$| y_i - \mathbf{w}^T \mathbf{x}_i |$$

Define the cost (or *loss*) for a particular weight vector $\mathbf{w}$ to be

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

One strategy for learning: *Find the $\mathbf{w}$ with least cost on this data*

This is called Least Mean Squares (LMS) Regression

$$\min_{\mathbf{w}} J(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Goal of learning: minimize mean squared error

- This is just the training objective: you can use different learning algorithms to minimize this objective
- Properties of $J(\mathbf{w})$: differentiable, convex, and lower values mean better weight vector $\mathbf{w}$, i.e., regressor.
- Mathematical optimization: focuses on solving problems of the form $\min_{\mathbf{w}} J(\mathbf{w})$. So many algorithms exist to solve problem

This is called Least Mean Squares (LMS) Regression

$$\min_{\mathbf{w}} J(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Goal of learning: minimize mean squared error

Different strategies exist for learning by optimization

- **Gradient descent**: is a popular algorithm
- **Matrix inversion**: for this particular minimization objective, there is also an analytical solution; no need for gradient descent: $b = (X^T X)^{-1} X^T Y$

Linear Regression

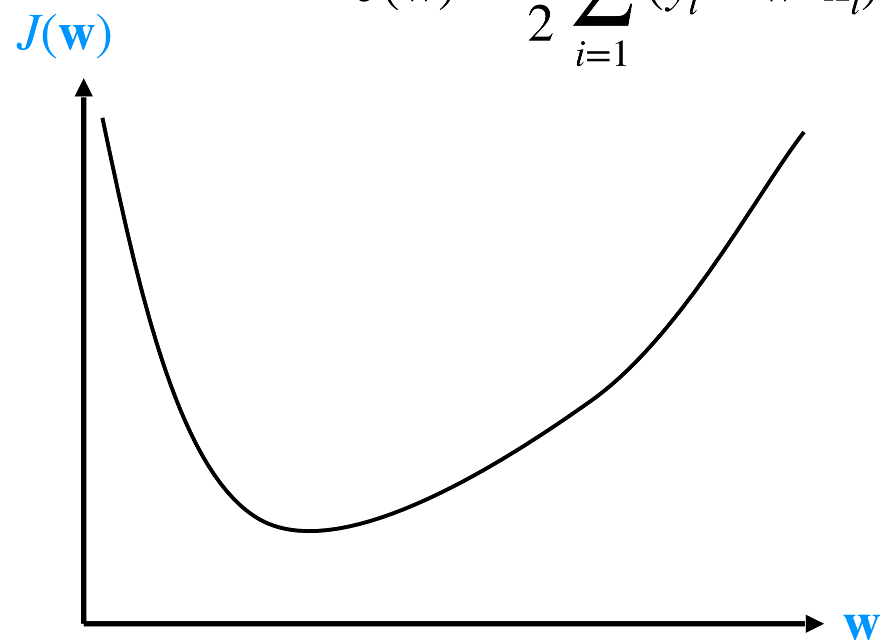
GRADIENT DESCENT

Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$



Why is J a convex function?

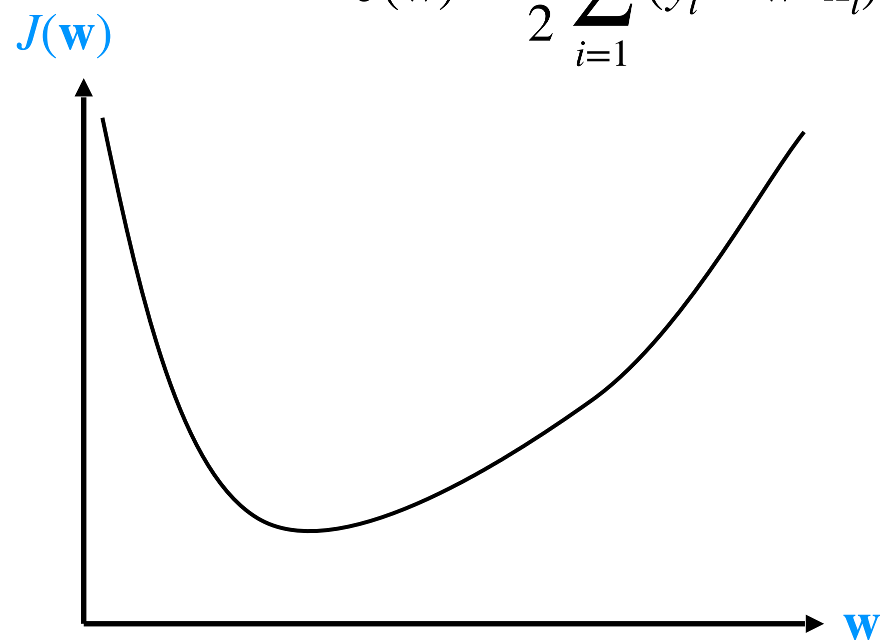
Why is it nice to minimize a convex function?

Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$



Why is J a convex function?

Why is it nice to minimize a convex function?

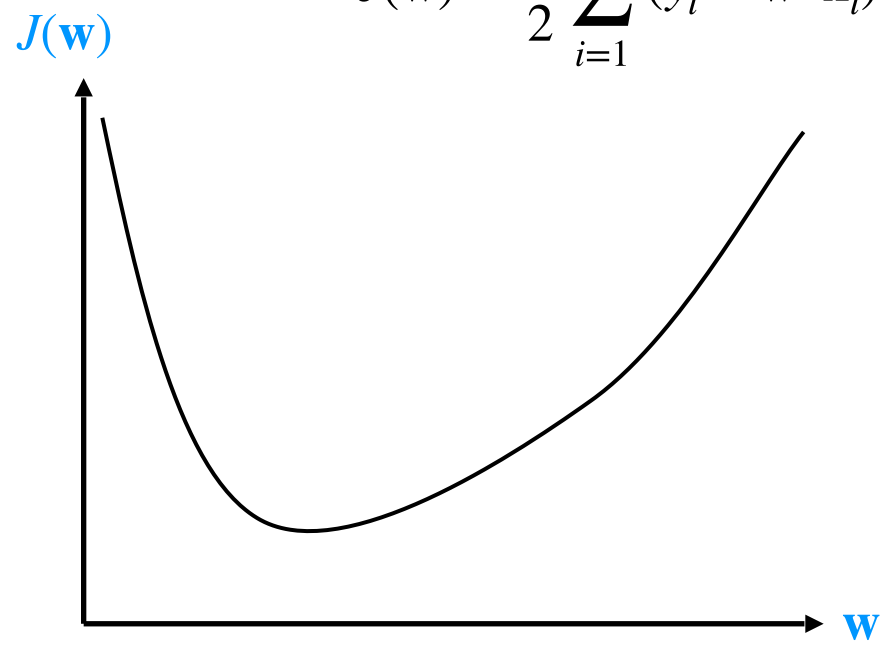
Gradient descent methods guaranteed to find minimum

Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$



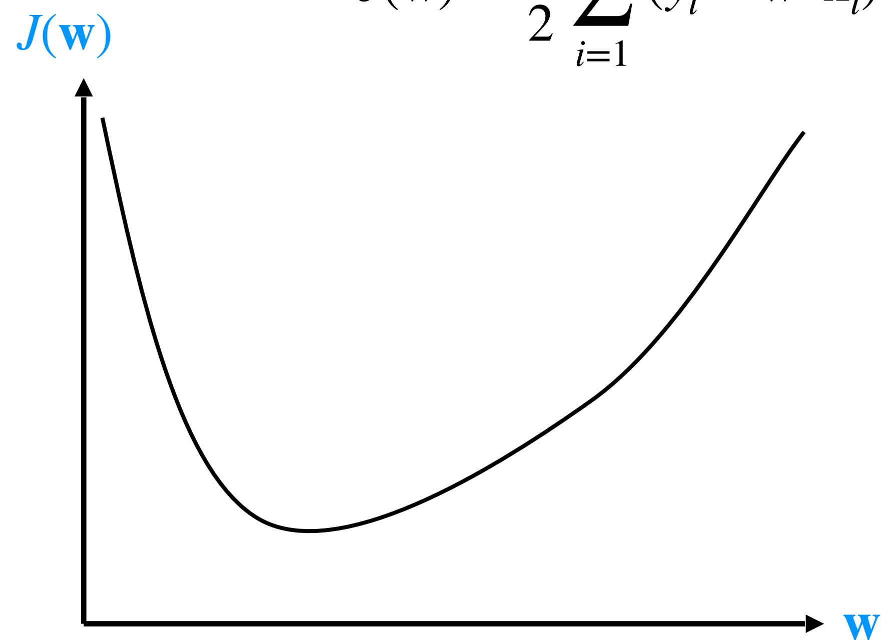
What is gradient of a function?

Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

What is gradient of a function?

In 2-dimensions: slope of a line

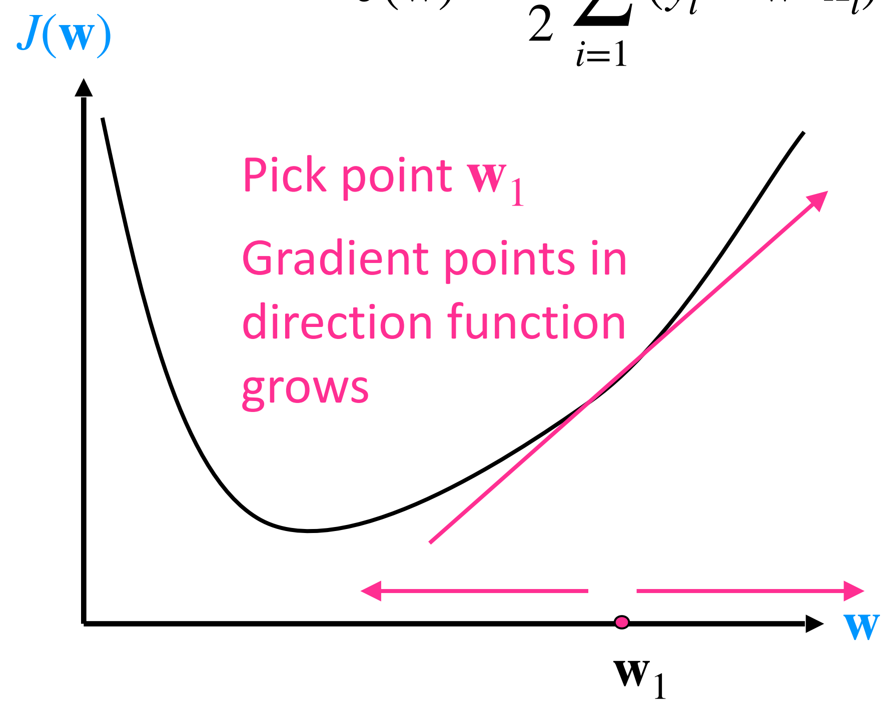
In higher dimensions: direction of the steepest ascent, that is, direction in which function grows the fastest

Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Gradient descent

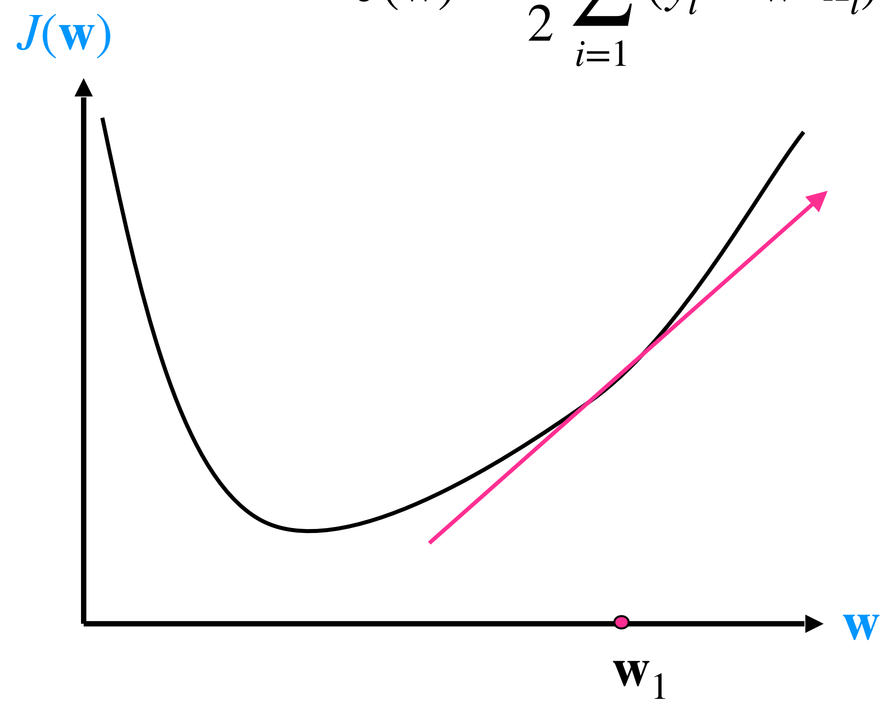
We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$

1. Start with an initial guess for $\mathbf{w}$, say $\mathbf{w}^0$

Gradient descent: initialize your starting point for search for minimum anywhere



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Gradient descent

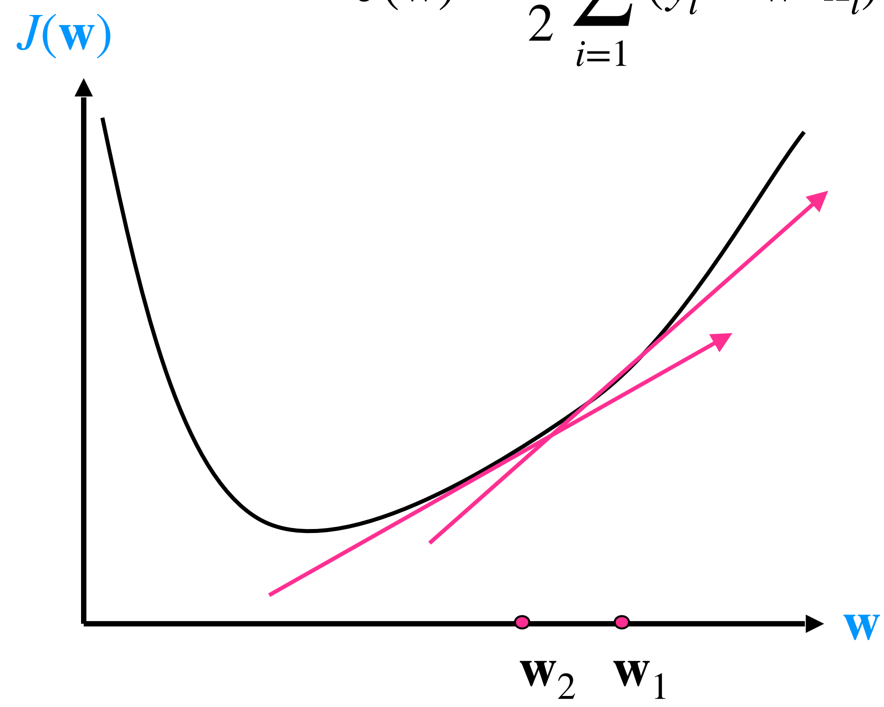
We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$

1. Start with an initial guess for $\mathbf{w}$, say $\mathbf{w}^0$

2. Iterate until convergence:
   - Compute the gradient of J at $\mathbf{w}^t$
   - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a step in the opposite direction of the gradient



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Then at every point, compute the gradient (the arrow), and take a step in direction away from gradient (i.e., move to a point where value of function is lower)

Gradient descent

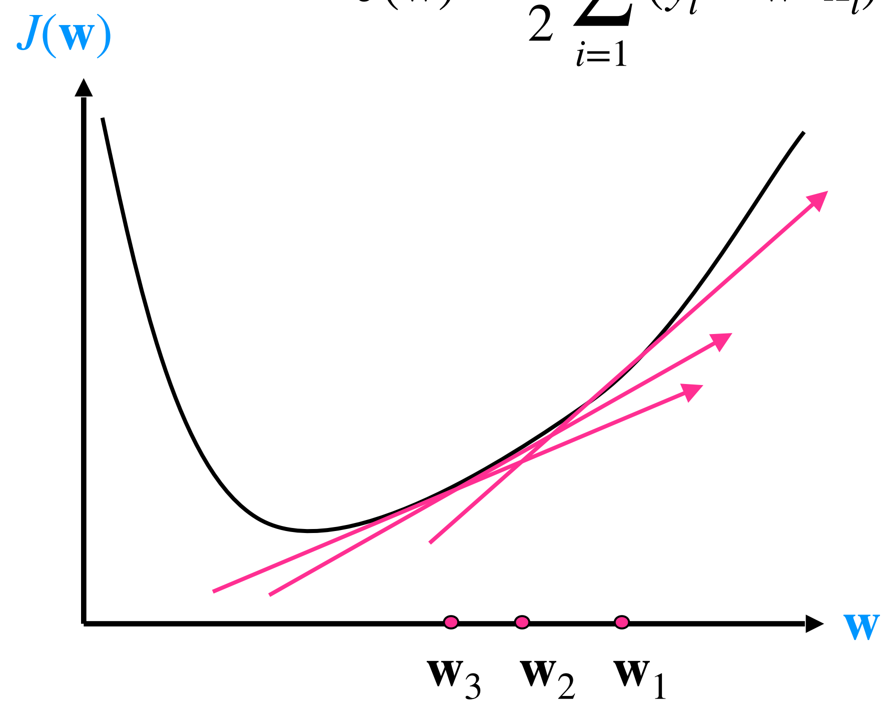
We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$

1. Start with an initial guess for $\mathbf{w}$, say $\mathbf{w}^0$

2. Iterate until convergence:
   - Compute the gradient of J at $\mathbf{w}^t$
   - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a step in the opposite direction of the gradient



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Keep repeating …

Gradient descent

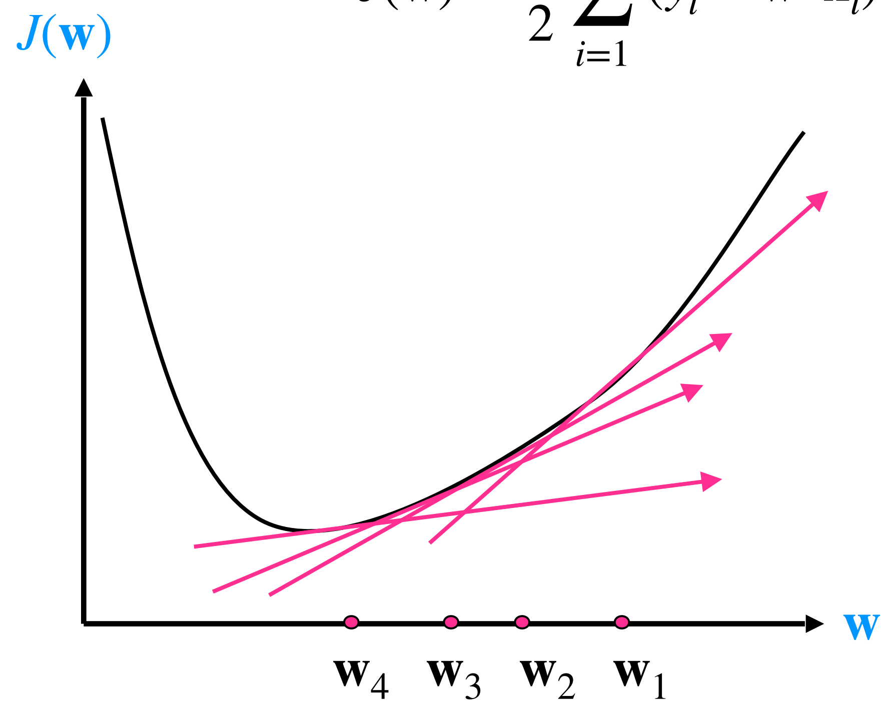
We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$

1. Start with an initial guess for $\mathbf{w}$, say $\mathbf{w}^0$

2. Iterate until convergence:
   - Compute the gradient of J at $\mathbf{w}^t$
   - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a step in the opposite direction of the gradient



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

And eventually you will get to minimum

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}^0$ Initialize to zeroes or random
   (convex function, so doesn’t matter where initialized)

2. For $t = 0,1,2,\ldots$
   - Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}^t$. Call it $\nabla J(\mathbf{w}^t)$

     Grad J or Nabla J

   - Update $\mathbf{w}$ as follows:

     $$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

     Use “-“ since step is in opposite direction

   where r is the learning rate (a small constant)

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}^0$

2. For $t = 0,1,2,\ldots$

What is the gradient of J?

   – $\boxed{\text{Compute gradient of } J(\mathbf{w}) \text{ at } \mathbf{w}^t\text{. Call it } \nabla J(\mathbf{w}^t)}$

   – Update $\mathbf{w}$ as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

where r is the learning rate (a small constant)

Gradient of the cost J at point $\mathbf{w}$

Remember that $\mathbf{w}$ is a vector with d elements

$$\mathbf{w} = [w_1, w_2, w_3, \ldots, w_j, \ldots, w_d]$$

J is a function that maps $\mathbf{w}$ to real number (the total cost)

Gradient of the cost J at point $\mathbf{w}$

Remember that $\mathbf{w}$ is a vector with d elements

$$\mathbf{w} = [w_1, w_2, w_3, \ldots, w_j, \ldots, w_d]$$

To find the best direction in the weight space $\mathbf{w}$ we compute the gradient of J with respect to each of the components of

$$\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$$

Gradient will be vector with d elements since $\mathbf{w}$ is a vector with d elements

Each element is a partial derivative

Need to compute every element to define gradient

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\frac{\partial J}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Let’s compute jth element in vector

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\frac{\partial J}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Gradient of sum is just the sum of gradients

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots)
\end{aligned}$$

Apply chain rule for derivative

Expanded dot product

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots)
\end{aligned}$$

Apply chain rule for derivatives

Expanded dot product

Only one element depends on j

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij})
\end{aligned}$$

Only one element depends on j

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij}) \\
&= -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}
\end{aligned}$$

Move 2 and minus outside, 2s cancel

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

One element of the gradient vector

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij}) \\
&= \boxed{-\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}}
\end{aligned}$$

Negative of this gradient is how much to change jth weight

Sum of Error x Input

Features (x_{ij}) with larger errors will cause larger change

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}^0$

2. For $t = 0,1,2,\ldots$
   – Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}^t$. Call it $\nabla J(\mathbf{w}^t)$

   Evaluate the function for *each* training example to compute the error and construct the gradient vector

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T x_i)x_{ij}$$

One element of $\nabla J(\mathbf{w}^t)$

$$\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$$

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}^0$

2. For $t = 0,1,2,\ldots$
   – Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}^t$. Call it $\nabla J(\mathbf{w}^t)$

   Evaluate the function for *each* training example to compute the error and construct the gradient vector

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T x_i)x_{ij}$$

One element of $\nabla J(\mathbf{w}^t)$

   – Update $\mathbf{w}$ as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

Take step in opposite direction of gradient, so minus

where r is the learning rate (for now a small constant)

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}^0$

2. For $t = 0,1,2,\ldots$ until error is below a threshold
   - Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}^t$. Call it $\nabla J(\mathbf{w}^t)$

     Evaluate the function for *each* training example to compute the error and construct the gradient vector

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T x_i)x_{ij}$$

One element of $\nabla J(\mathbf{w}^t)$

   - Update $\mathbf{w}$ as follows:

Take step in opposite direction of gradient, so minus

$$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

where r is the learning rate (for now a small constant)

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}^0$

2. For $t = 0,1,2,\ldots$ until error is below a threshold
   - Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}^t$. Call it $\nabla J(\mathbf{w}^t)$

     Evaluate the function for *each* training example to compute the error and construct the gradient vector

     $$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T x_i)x_{ij}$$

     One element of $\nabla J(\mathbf{w}^t)$

     After computing error for all training examples, get vector that you use to update weights all at once: basically a batch
   - Update $\mathbf{w}$ as follows:

     $$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

     where r is the learning rate (for now a small constant)