Lecture 10: Least Mean Squares Regression

COMP 411, Fall 2021

Victoria Manfredi

WESLEYAN
UNIVERSITY

Acknowledgements: These slides are based primarily on those created by Vivek Srikumar (U of Utah), Russ Grenier (U of Alberta), and lectures by Balaraman Ravindran (IIT Madras)

Today's Topics

Homework 4 out

- Due out Thursday October 7, by 11:59p

Least squares method for regression

- Examples
- The LMS objective
- Gradient descent
- Incremental/stochastic gradient descent

LMS Regression

OVERVIEW

What's the mileage?

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

Weight (x100lb) 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

Prediction of continuous variables

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

Predicting continuous values using a linear model

Assumption: the output is a linear function of the inputs

Mileage = $w_0 + w_1 x_1 + w_2 x_2$

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

Learning: using 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: The strategy

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

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

We have a training data set:

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

We want to approximate y as

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

$$f(\mathbf{x}) = y = w_1 + \sum_{i=2}^{d} w_j x_j$$

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

We can write compactly as

$$f(\mathbf{x}) = y = \mathbf{w}^T \mathbf{x}$$

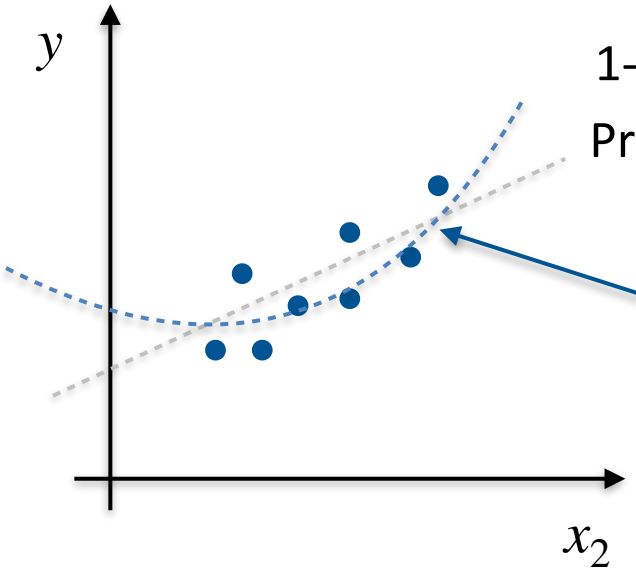
For simplicity, we will assume that the first feature is always 1, to make notation easier

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

To avoid special treatment of w_1

Have made assumption that f is a linear function!

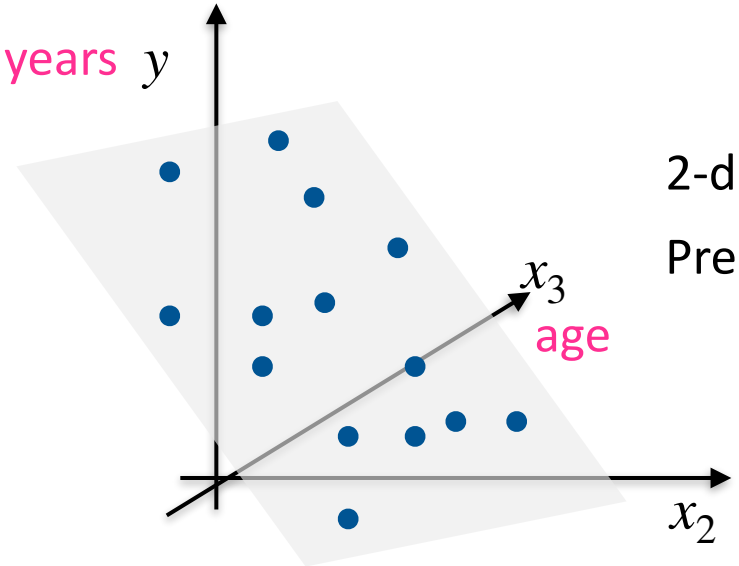
Examples



1-dimensional input

Predict using $y = w_1 + w_2 x_2$

The linear function is not our only choice. We could have tried to fit the data as another polynomial



2-dimensional input

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

LMS Regression

OBJECTIVE

What is the best weight vector?

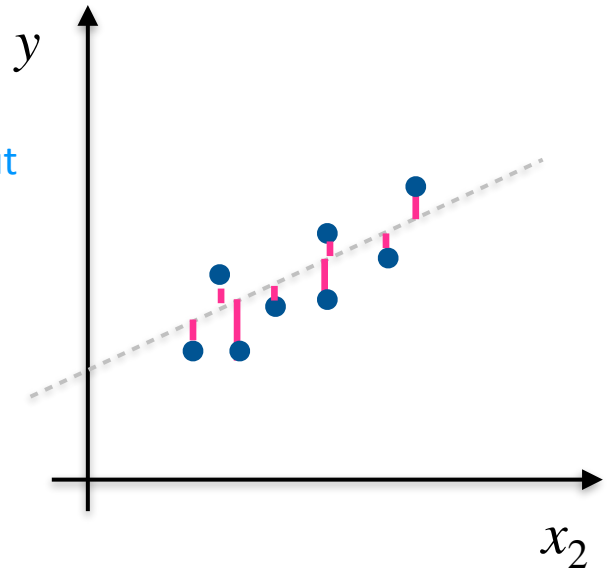
Question: how do we know which weight vector is the best one for a training set (having made assumption that f is a linear function)?

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



What is the best weight vector?

Question: how do we know which weight vector is the ***best*** one for a training set (having made assumption that f is a linear function)?

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

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

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

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

A General Framework for Learning

Goal: predict an unobserved output value $y \in Y$ based on an observed input vector $\mathbf{x} \in X$

How? Estimate a functional relationship $y \sim f(x)$ from a set $\{(\mathbf{x}, y)_i\}_{i=1,n}$

- Classification: $y \in \{0,1\}$ (or $y \in \{1,2,\ldots,k\}$),
- Regression: $y \in \Re$)

Simple loss function: # of mistakes, $[f(\mathbf{x}) \neq y]$ is an indicator function

What do we want $f(x)$ to satisfy?

- We want to minimize the Risk: $L(f()) = E_{X,Y}([f(x) \neq y])$
- Where $E_{X,Y}$ denotes the expectation with respect to the true distribution.

A General Framework for Learning

We want to minimize the Risk: $L(f()) = E_{X,Y}([f(X) \neq Y])$

- Where $E_{X,Y}$ denotes the expectation with respect to the true distribution.

We cannot minimize this loss!

- Instead, we try to minimize the empirical classification error

For a set of training examples $\{(\mathbf{x}_i y_i)\}_{i=1,m}$, where m is # of examples, try to minimize:

$$L'(f()) = \frac{1}{m} \sum_{i=1:m} [f(\mathbf{x}_i) \neq y_i]$$

This minimization is typically NP-hard. To alleviate this computational problem, minimize a new function - a convex upper bound of the classification error

$$I(f(x), y) = [f(x) \neq y] = \{1 \text{ when } f(x) \neq y; 0 \text{ otherwise } \}$$

Algorithmic View of Learning: an Optimization Problem

A Loss Function $L(f(\mathbf{x}), y)$ measures the penalty incurred by a classifier f on example $(\mathbf{x}, y)$.

There are many different loss functions one could define:

- Misclassification Error (0-1 loss):

$$L(f(\mathbf{x}), y) = 0 \text{ if } f(\mathbf{x}) = y; \quad 1 \text{ otherwise}$$

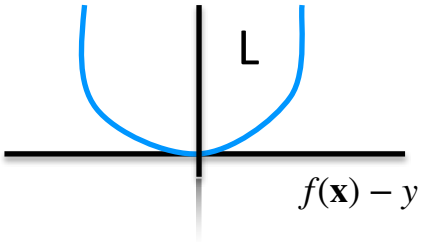
- Squared Loss:

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

- Input dependent loss:

$$L(f(\mathbf{x}), y) = 0 \text{ if } f(\mathbf{x}) = y; \quad c(x) \text{ otherwise.}$$

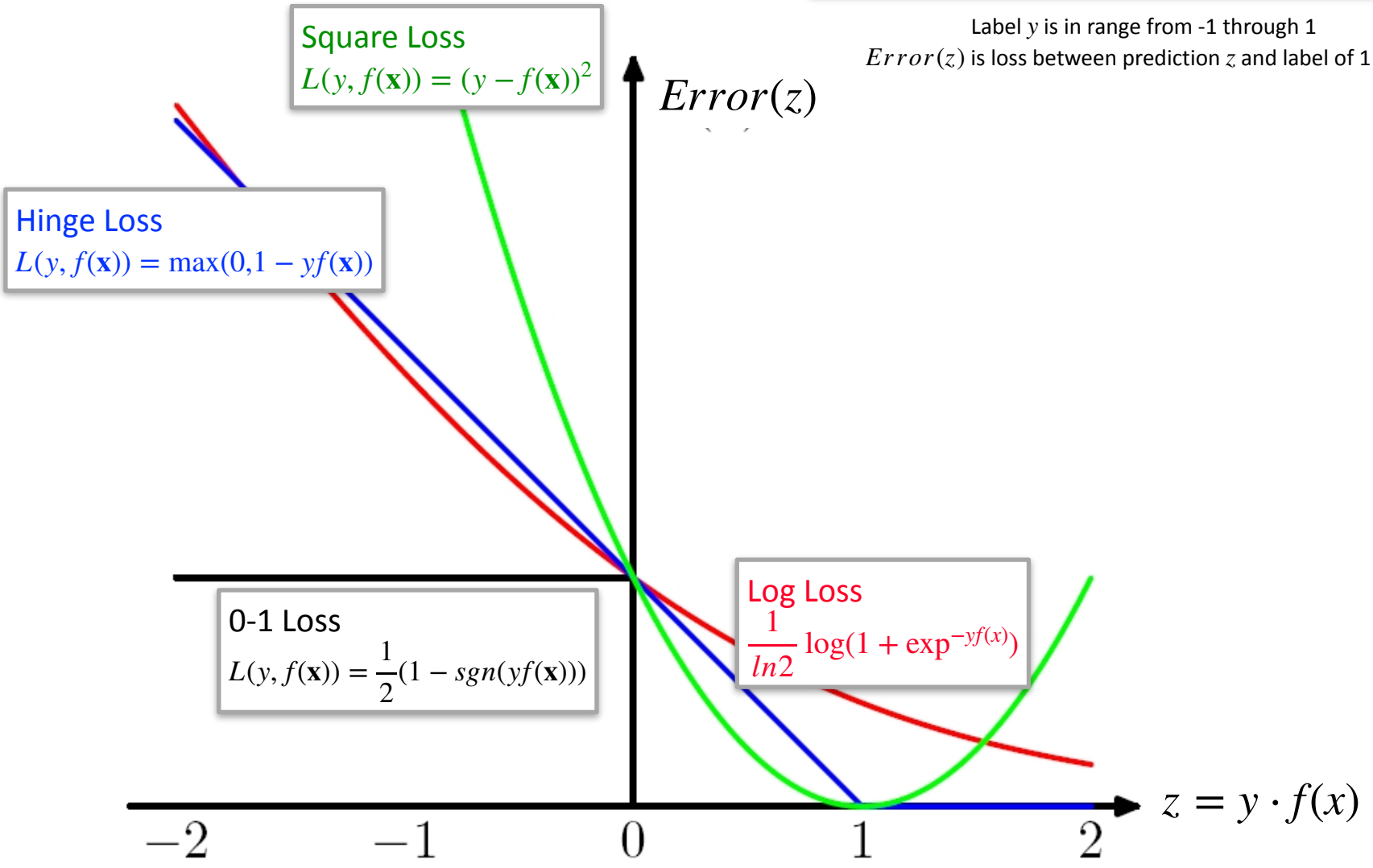
A continuous convex loss function allows a simpler optimization algorithm



Loss

Here f(x) is the prediction $\in \Re$
$y \in \{-1,1\}$ is the correct value

Label y is in range from -1 through 1
$Error(z)$ is loss between prediction z and label of 1 or -1



What causes prediction errors?

Noise in the measurement

- suppose I tell you that I measure the temperature at 3a every day. There will be some kind of natural variation in the temperatures.

Our ignorance about the system

- Maybe it's not important that I take the temperature at 3a. Maybe it's important where in the building I take the temperature. Maybe I am measuring in the kitchen or next to an air conditioner. Or maybe I am measuring inside an air-conditioned room.

Could say that natural variations are due to factors that you don't know anything about.

- could very well be that there is no real natural variation in the data, no noise. All of the uncertainty about the data arises from my lack of knowledge. But that is a philosophical question. There are things which are measurable which we don't measure.

Least Mean Squares (LMS) Regression

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

Learning: minimizing 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: $b = (X^T X)^{-1} X^T Y$

Gradient descent vs. matrix inversion

Gradient descent pros

- Biologically plausible
- Each iteration cost only $O(mn)$
- If uses $< m$ iterations, faster than Matrix inversion ($\sim O(N^2 \log(N))$
- More easily parallelizable

Gradient descent cons

- It's moronic … essentially a slow way to build $X^T X$ matrix, then solve a set of linear equations
- If n is small, it's especially outrageous. If n is large then direct matrix inversion method can be problematic but not impossible if you want to be efficient
- Need to choose a good learning rate … how?
- Matrix inversion takes predictable time. You can't be sure when gradient descent will stop

LMS Regression

GRADIENT DESCENT

Gradient of the cost

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

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

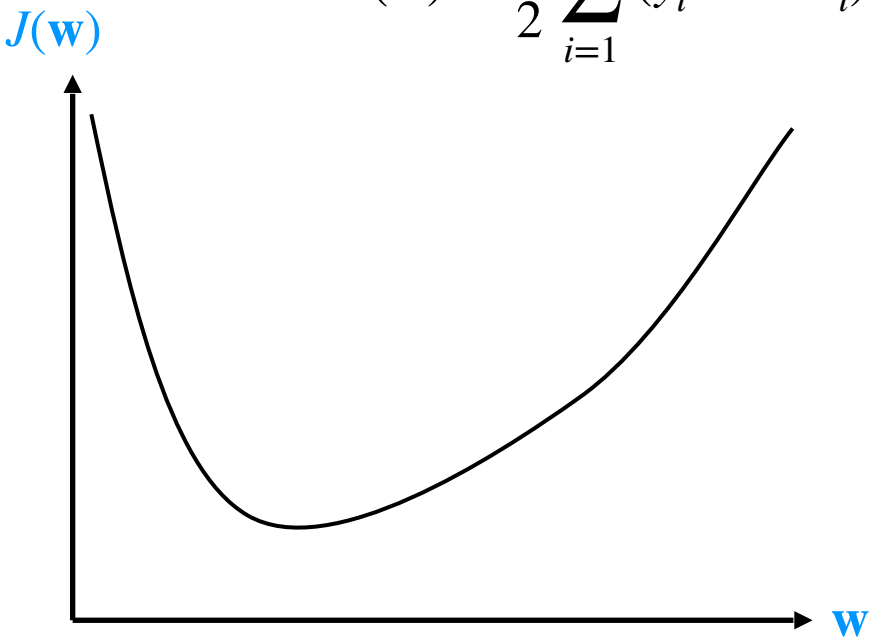
This vector specifies the direction that produces the steepest increase in J. We want to modify $\mathbf{w}$ in the direction of $-\nabla J(\mathbf{w})$, where (with a fixed step size r):

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

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

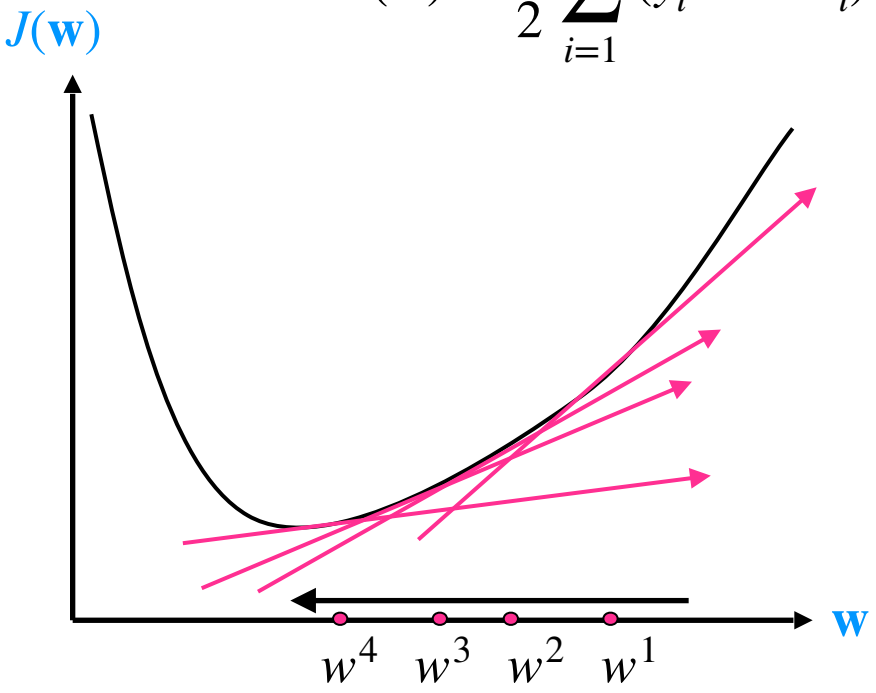
Gradient descent

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

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?

   - Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}^t$. 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 (for now a small constant)

Gradient of the cost

We are trying to minimize

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

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

$$\begin{aligned}
\frac{\partial J}{\partial w_i} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T x_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T x_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T x_i)\frac{\partial}{\partial w_j}(y_i - w_1 x_{i1} - \cdots w_j x_{ij} - \cdots) \\
&= -\frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T x_i)(x_{ij}) \\
&= -\sum_{i=1}^{m}(y_i - \mathbf{w}^T x_i)x_{ij}
\end{aligned}$$

Check whether i's and j's are okay, and why last line is negative

Gradient of the cost

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

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

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

$$= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T x_i)\frac{\partial}{\partial w_j}(y_i - w_1 x_{i1} - \cdots w_j x_{ij} - \cdots)$$

$$= -\frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T x_i)(x_{ij})$$

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

One element of the gradient vector

Sum of Error x Input

Check whether i's and j's are okay, and why last line is negative

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:

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

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

   – 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)

This algorithm is guaranteed to converge to the minimum of J if r is small enough (small enough steps). Why? The objective J is a convex function here (LMS for linear regression): the surface contains only a single global minimum. The surface may have local minimum if the loss function is different.

Linear Regression

INCREMENTAL/STOCHASTIC GRADIENT DESCENT

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

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

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

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

The weight vector is not updated until *all* errors are calculated.

Why not make early updates to the weight vector as soon as we encounter errors instead of waiting for a full pass over the data?

Incremental/stochastic gradient descent

Repeat for each example $(\mathbf{x}_i, y_i)$

- Pretend that entire training set is represented by this single example
- Use this example to calculate the gradient and update the model

Contrast with batch gradient descent which makes one update to the weight vector for every pass over the data

Incremental/stochastic gradient descent

1. Initialize $\mathbf{w}$

2. For $t = 0,1,2,\ldots$ until error is below a threshold
   - For each training example $(\mathbf{x}_i, y_i)$, update $\mathbf{w}$. For each element of the weight vector (w_j)

$$w_j^{t+1} = w_j^t - r(y_i - \mathbf{w}^T \mathbf{x}_i)x_{ij}$$

Contrast with the previous method, where the weights are updated only after all examples are processed once

This update rule is also called the Widrow-Hoff rule in the neural networks literature

May get close to optimum much faster than the batch version. In general - does not converge to global minimum. Decreasing r with time guarantees convergence. But, online/incremental algorithms are often preferred when the training set is very large

Learning Rates and Convergence

- In the general (non-separable) case the learning rate r must decrease to zero to guarantee convergence.

- The learning rate is called the *step size*. There are more sophisticated algorithms that choose the step size automatically and converge faster.

- Choosing a better starting point also has impact.

- The gradient descent and its stochastic version are very simple algorithms, but almost all the algorithms we will learn in the class can be traced back to gradient decent algorithms for different loss functions and different hypotheses spaces.

Linear regression: summary

- What we want: predict a real-valued output using feature representation of the input

- Assumption: output is a linear function of the inputs

- Learning by minimizing total cost
  - gradient descent and stochastic gradient descnt to find the *best* weight vector
  - This particular optimization can be computed directly by framing the problem as a matrix problem