Do not distribute course material

Gradient Descent: http://theory.stanford.edu/~tim/s16/l/l5.pdf

Topic 2
Multiple Linear Regression

INTRODUCTION TO MACHINE LEARNING

Thanks to:

- Much of the material from this course is from Prof. Sundeep Rangan
  - This includes many of the slides, the outline of the course, the demos, many of the labs and many of the homework assignments
- Some approaches to introducing the material have been taken from Hsuan'Tien Lin's lecture slides (He is one of the authors of Learning from Data: A Short Course)
- Some approaches are taken from CMU 18-661 Introduction to Machine Learning

Learning Objectives

- ❑ Formulate a machine learning model as a multiple linear regression model
  - Identify features and label/target for the problem
- ❑ Understand the data matrix/design matrix
- ❑ Find the solution using gradient descent
- ❑ Write the regression model in vectorized form
- ❑ Derive the closed form solution for multiple linear regression
- ❑ Compute the least-squares solution for the regression coefficients on training data
- ❑ Understand feature scaling/data normalization

Finding Parameters via Optimization
A general ML recipe

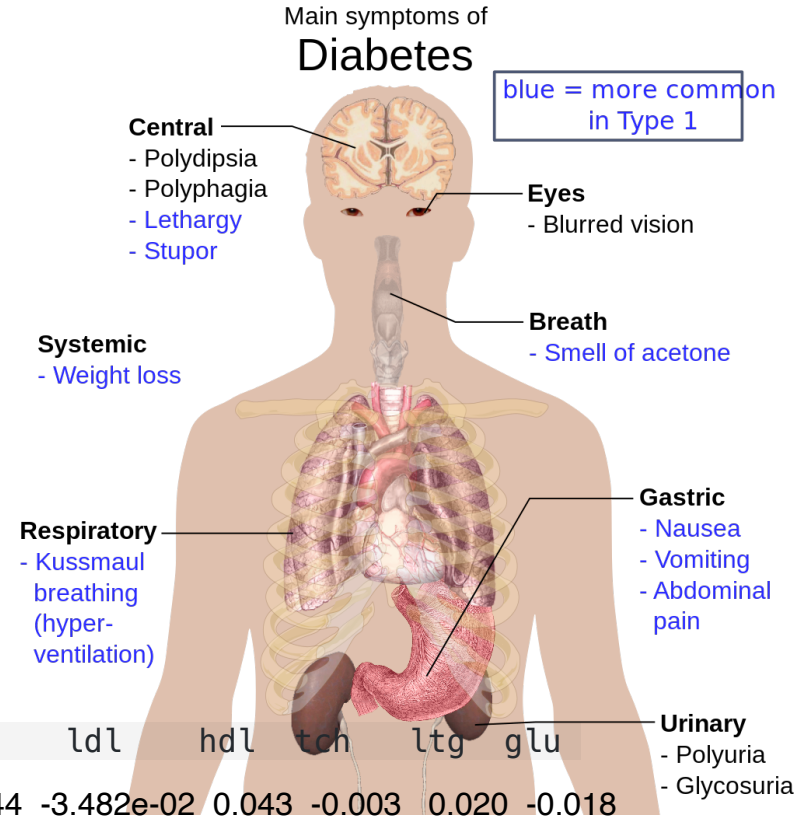
General ML problem	Multiple linear regression
❑Get data	Data: $(\mathbf{x}^{(i)}, y^{(i)}), i = 1,2,\ldots,N$
❑Pick a model with parameters	Linear model: $\hat{y}^{(j)} = w_0 + w_1 x_1^{(j)} + w_2 x_2^{(j)} + \cdots + w_d x_d^{(j)}$
❑Pick a Objective function/loss function ◦ Measures goodness of fit model to data ◦ Function of the parameters	Loss function: $\text{RSS}(\mathbf{w}) = \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2$ $= (y^{(1)} - \hat{y}^{(1)})^2 + (y^{(2)} - \hat{y}^{(2)})^2 + \cdots + (y^{(N)} - \hat{y}^{(N)})^2$
❑Optimizer: Find parameters that minimizes loss	Select $\mathbf{w} = [w_0, w_1, w_2, \ldots, w_d]^T$ to minimize $\text{RSS}(\mathbf{w})$

Outline

- Motivating Example: Understanding glucose levels in diabetes patients
- Multiple variable linear models
- Least squares solutions
  - Gradient descent
  - Normal Equations
  - Feature scaling
- Evaluating our hypothesis
- Computing the solutions in python
- Special case: Simple linear regression
- Extensions
- Removing features

Example: Diabetes Patients Progression

- Can we predict diabetes patients' condition a year after taking 10 baseline measurements?
- The 10 baseline measurements are: age, sex, body mass index. average blood pressure, and 6 blood serum measurements
- Many factors affect diabetes
- are difficult to obtain
  - Hard to derive from first principles
  - Difficult to model physiological process precisely
- Can machine learning help?

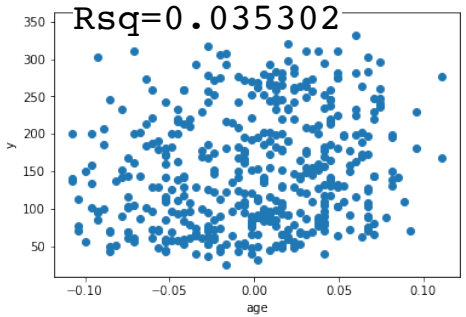


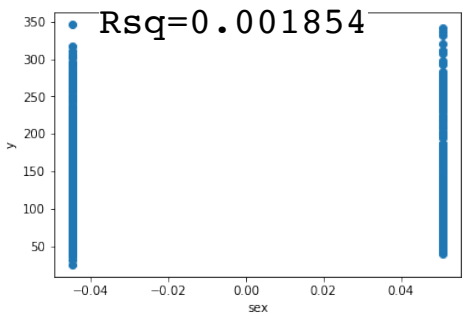
By Mikael Häggström - See above. (Altered image use in public domain.) [Public domain], via Wikimedia Commons

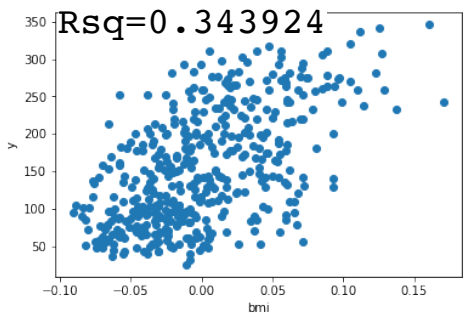
age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
59	2	32.1	101	157	93.2	38	4	4.8598	87
48	1	21.6	87	183	103.2	70	3	3.8918	69
72	2	30.5	93	156	93.6	41	4	4.6728	85
24	1	25.3	84	198	131.4	40	5	4.8903	89
50	1	23	101	192	125.4	52	4	4.2905	80
23	1	22.6	89	139	64.8	61	2	4.1897	68
36	2	22	90	160	99.6	50	3	3.9512	82

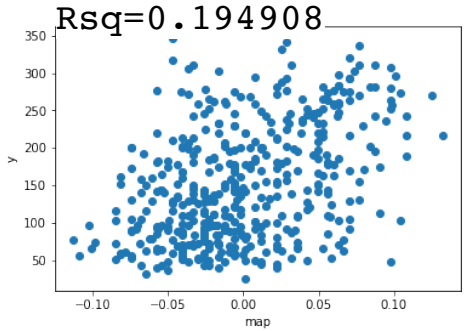
age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
0.038	0.051	0.062	2.187e-02	-0.044	-3.482e-02	0.043	-0.003	0.020	-0.018
-0.002	-0.045	-0.051	-2.633e-02	-0.008	-1.916e-02	-0.074	-0.039	-0.068	-0.092
0.085	0.051	0.044	-5.670e-03	-0.046	-3.419e-02	0.032	-0.003	0.003	-0.026
-0.089	-0.045	-0.012	-3.666e-02	0.012	2.499e-02	0.036	0.034	0.023	-0.009
0.005	-0.045	-0.036	2.187e-02	0.004	1.560e-02	-0.008	-0.003	-0.032	-0.047
-0.093	-0.045	-0.041	-1.944e-02	-0.069	-7.929e-02	-0.041	-0.076	-0.041	-0.096
-0.046	0.051	-0.047	-1.600e-02	-0.040	-2.480e-02	-0.001	-0.039	-0.063	-0.038

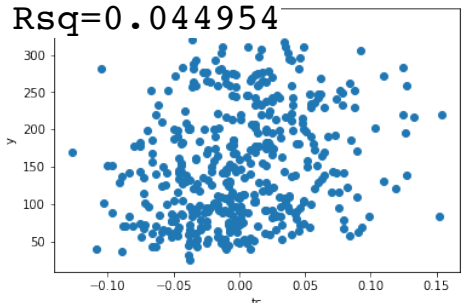
As before, lets plot at the features

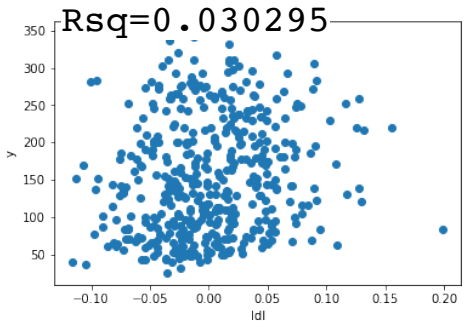


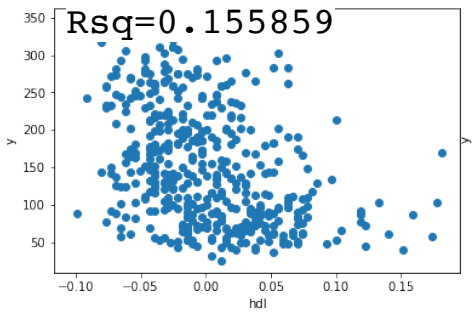


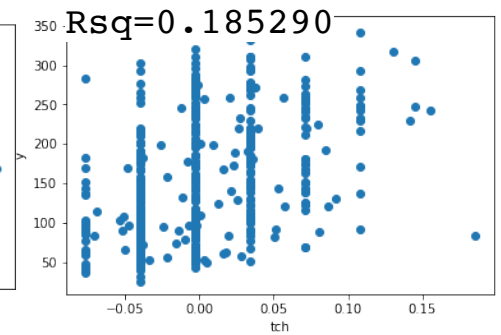


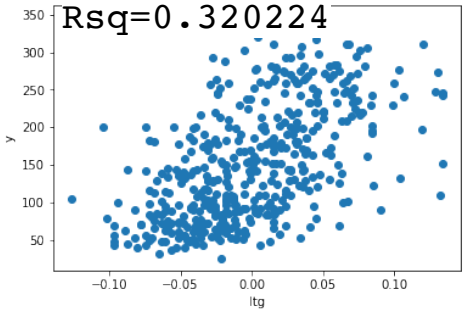


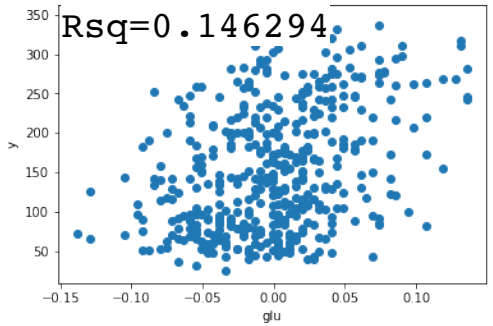












Could we do a better job of predicting if we used more than one feature to make the prediction?

Outline

- Motivating Example: Understanding glucose levels in diabetes patients
- Multiple variable linear models
- Least squares solution:
  - Gradient descent
  - Normal Equations
  - Feature scaling
- Evaluating our hypothesis
- Computing the solutions in python
- Special case: Simple linear regression
- Extensions
- Removing features

Demo on GitHub

- Code will be posted on Brightspace

Demo: Predicting Glucose Levels using Mulitple Linear Regression

In this demo, you will learn how to:

- Fit multiple linear regression models using python's `sklearn` pachage.
- Split data into training and test.
- Manipulating and visualizing multivariable arrays.

We first load the packages as usual.

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

Diabetes Data Example

To illustrate the concepts, we load the well-known diabetes data set. This dataset is included in the `sklearn.da`
can be loaded as follows.

```
from sklearn import datasets, linear_model, preprocessing
```

Loading the Data

```
from sklearn import datasets, linear_model, preprocessing

# Load the diabetes dataset
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
```

- Sklearn package:
  - Many methods for machine learning
  - Datasets
  - Will use throughout this class
- Diabetes dataset is one example

```
nsamp, natt = X.shape
print("num samples={0:d}  num attributes={1:d}".format(nsamp,natt))
```

```
num samples=442  num attributes=10
```

Design matrix: Diabetes Patients Progression

$$X = \begin{bmatrix}
\texttt{age} & \texttt{sex} & \texttt{bmi} & \texttt{map} & \texttt{tc} & \texttt{ldl} & \texttt{hdl} & \texttt{tch} & \texttt{ltg} & \texttt{glu} \\
59 & 2 & 32.1 & 101 & 157 & 93.2 & 38 & 4 & 4.8598 & 87 \\
48 & 1 & 21.6 & 87 & 183 & 103.2 & 70 & 3 & 3.8918 & 69 \\
72 & 2 & 30.5 & 93 & 156 & 93.6 & 41 & 4 & 4.6728 & 85 \\
24 & 1 & 25.3 & 84 & 198 & 131.4 & 40 & 5 & 4.8903 & 89 \\
50 & 1 & 23 & 101 & 192 & 125.4 & 52 & 4 & 4.2905 & 80 \\
23 & 1 & 22.6 & 89 & 139 & 64.8 & 61 & 2 & 4.1897 & 68 \\
36 & 2 & 22 & 90 & 160 & 99.6 & 50 & 3 & 3.9512 & 82
\end{bmatrix} = \begin{bmatrix} \mathbf{x}^{(1)T} \\ \mathbf{x}^{(2)T} \\ \mathbf{x}^{(3)T} \\ \mathbf{x}^{(4)T} \\ \mathbf{x}^{(5)T} \\ \mathbf{x}^{(6)T} \\ \mathbf{x}^{(7)T} \end{bmatrix}$$

$$\mathbf{x}^{(1)} = \begin{bmatrix} 59 \\ 2 \\ 32.1 \\ 101 \\ 157 \\ 93.2 \\ 38 \\ 4 \\ 4.8598 \\ 87 \end{bmatrix}$$

$$\mathbf{x}^{(1)T} = [59 \quad 2 \quad 32.1 \quad 101 \quad 157 \quad 93.2 \quad 38 \quad 4 \quad 4.8598 \quad 87]$$

1-indexed example in book and notes (common in math)
0-indexed in python

e (using y some of the 442 aining exampl

7th feature is in the 7th column

7 x 10 matrix

4th example is in the 4th row

y, a column vector is an n x 1 matrix. E.g. Y is a 7 x 1 matrix. It is also called a 7 dimensional vector

$$X = \begin{bmatrix} \text{age} & \text{sex} & \text{bmi} & \text{map} & \text{tc} & \text{ldl} & \text{hdl} & \text{tch} & \text{ltg} & \text{glu} \\ 59 & 2 & 32.1 & 101 & 157 & 93.2 & 38 & 4 & 4.8598 & 87 \\ 48 & 1 & 21.6 & 87 & 183 & 103.2 & 70 & 3 & 3.8918 & 69 \\ 72 & 2 & 30.5 & 93 & 156 & 93.6 & 41 & 4 & 4.6728 & 85 \\ 24 & 1 & 25.3 & 84 & 198 & 131.4 & 40 & 5 & 4.8903 & 89 \\ 50 & 1 & 23 & 101 & 192 & 125.4 & 52 & 4 & 4.2905 & 80 \\ 23 & 1 & 22.6 & 89 & 139 & 64.8 & 61 & 2 & 4.1897 & 68 \\ 36 & 2 & 22 & 90 & 160 & 99.6 & 50 & 3 & 3.9512 & 82 \end{bmatrix} \quad y = \begin{bmatrix} 151 \\ 75 \\ 141 \\ 206 \\ 135 \\ 97 \\ 138 \end{bmatrix}$$

$y^{(4)}$

Where is the 4th example?

Where is the 7th feature for all the examples?

What is the 7th feature of the 4th example $x_7^{(4)}$? $x_7^{(4)} = 40$

What is the measure of disease progression one year after baseline of the 4th example $y^{(4)}$? $y^{(4)} = 206$

Data from https://web.stanford.edu/~hastie/Papers/LARS/diabetes.data

Standardized Example

$X =$

age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
0.038	0.051	0.062	2.187e-02	-0.044	-3.482e-02	0.043	-0.003	0.020	-0.018
-0.002	-0.045	-0.051	-2.633e-02	-0.008	-1.916e-02	-0.074	-0.039	-0.068	-0.092
0.085	0.051	0.044	-5.670e-03	-0.046	-3.419e-02	0.032	-0.003	0.003	-0.026
-0.089	-0.045	-0.012	-3.666e-02	0.012	2.499e-02	0.036	0.034	0.023	-0.009
0.005	-0.045	-0.036	2.187e-02	0.004	1.560e-02	-0.008	-0.003	-0.032	-0.047
-0.093	-0.045	-0.041	-1.944e-02	-0.069	-7.929e-02	-0.041	-0.076	-0.041	-0.096
-0.046	0.051	-0.047	-1.600e-02	-0.040	-2.480e-02	-0.001	-0.039	-0.063	-0.038

$y =$

-1.133
-77.133
-11.133
53.866
-17.133
-55.133
-14.133

“data are first standardized to have zero mean and unit L2 norm”

Feature vector, data matrix augmented with a 1
Merge bias/intercept into coefficient/weight vector

$x_0 = 1$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \qquad X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(N)} & \dots & x_d^{(N)} \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

Design matrix: $X \in \mathbb{R}^{N \times (d+1)}$

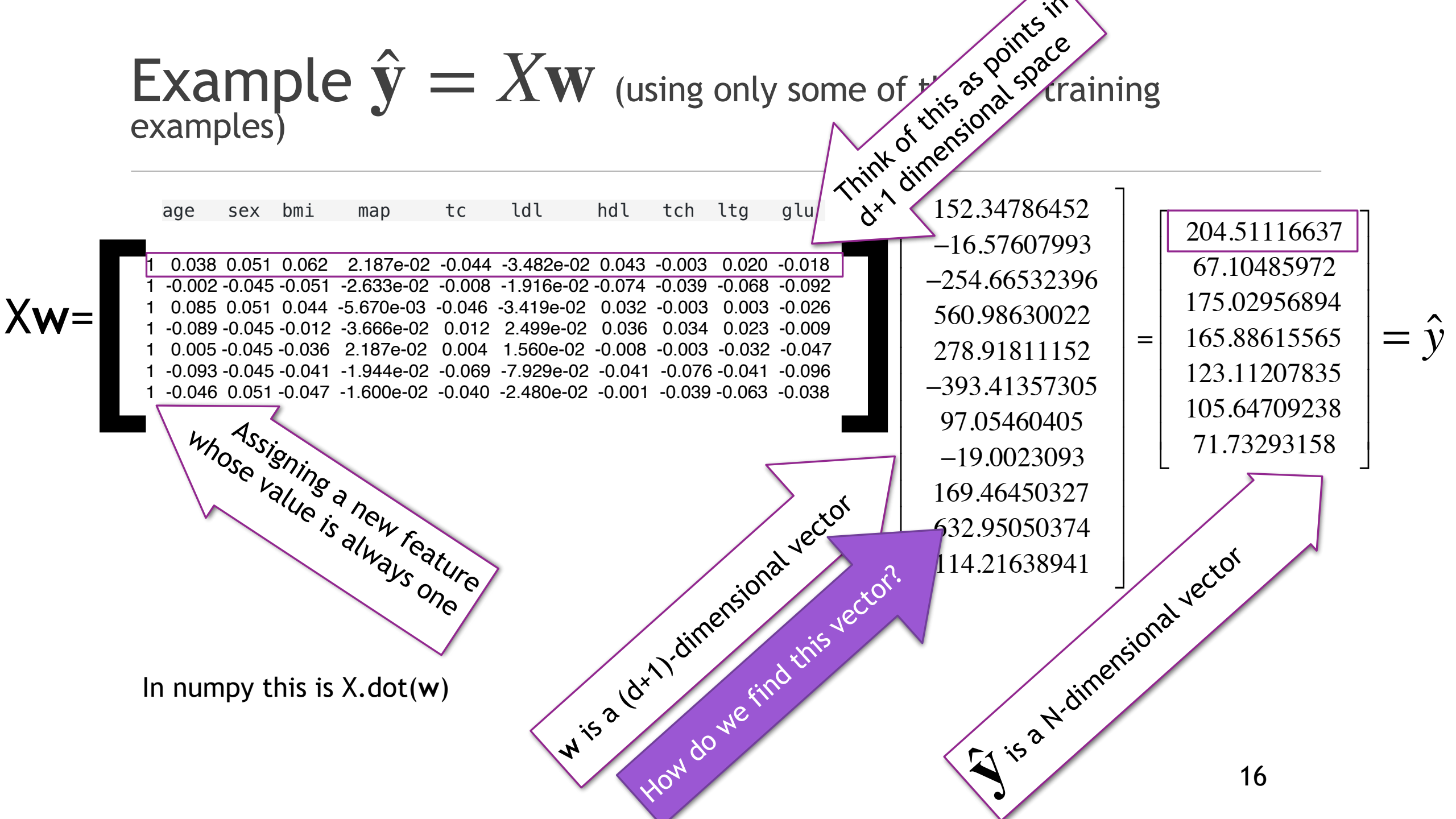
Weight vector: $\mathbf{w} \in \mathbb{R}^{d+1}$

Feature vector: $\mathbf{x} \in \mathbb{R}^{d+1}$

$$\hat{y}^{(i)} = w_0 + w_1 \cdot x_1^{(i)} + w_2 \cdot x_2^{(i)} + \cdots + w_d \cdot x_d^{(i)} = \begin{bmatrix} w_0 & w_1 & w_2 & \dots & w_d \end{bmatrix} \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix} = \mathbf{w}^T \mathbf{x}^{(i)}$$

$$\hat{\mathbf{y}} = X \cdot \mathbf{w}$$

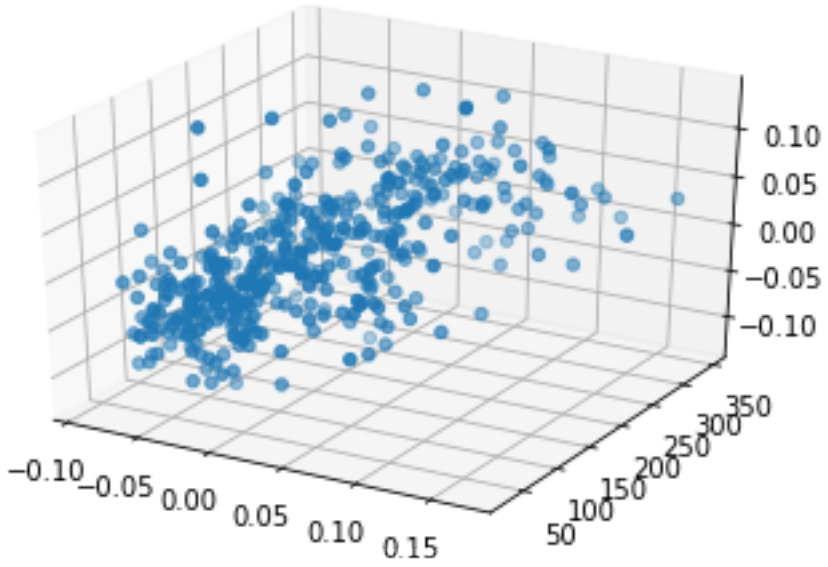
Example $\hat{\mathbf{y}} = X\mathbf{w}$ (using only some of t training examples)

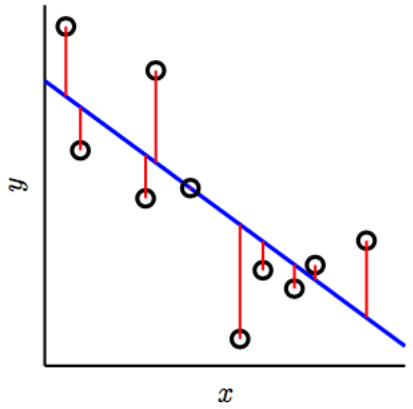


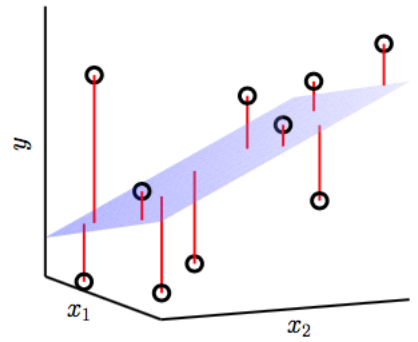
In numpy this is X.dot(**w**)

Least mean squares when $\mathbf{x}$ is d-dimensional

Finding the best line/hyperplane with the smallest squared residuals







$y = f(\mathbf{x}) + \epsilon$ ⟵ noisy target $P(y|\mathbf{x})$

$$y^{(i)} \approx \hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_d x_d^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$$

$$RSS(\mathbf{w}) = \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2 = \sum_{i=1}^{N} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2$$

We imagine that
$y = w_0 + w_1 x_1 + w_2 x_2 + \epsilon$

The pictures on the right are from the Malik Magdon-Ismail and Hsuan-Tien Lin's lecture slides

"Remember that all models are wrong; the practical question is how wrong do they have to be to not be useful." George Box

Linear regression is predicting a continuous value given some input

Outline

- Motivating Example: Understanding glucose levels in diabetes patients
- Multiple variable linear models
- Least squares solutions
  - Gradient descent (local optimization)
  - Normal Equations (global optimization)
  - Feature scaling
- Evaluating our hypothesis
- Computing the solutions in python
- Special case: Simple linear regression
- Extensions
- Removing features

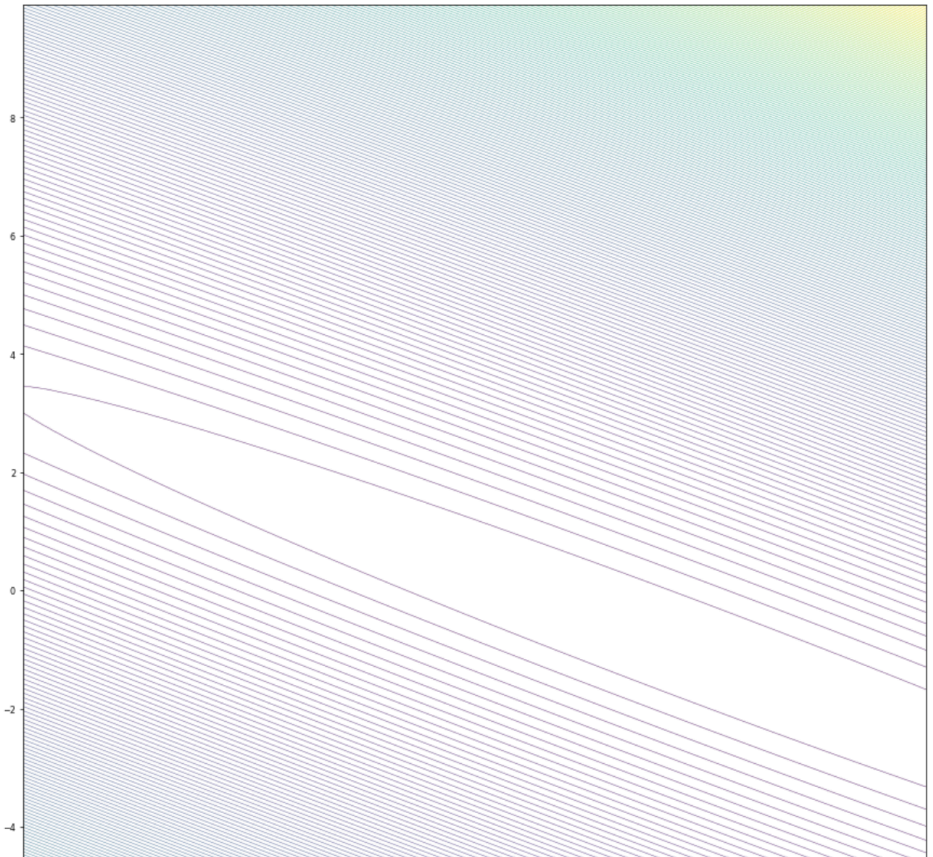
Local Optimization

Finding the optimal parameters **w**

If I gave you a bad guess at the optimal w, could you improve it a little bit?

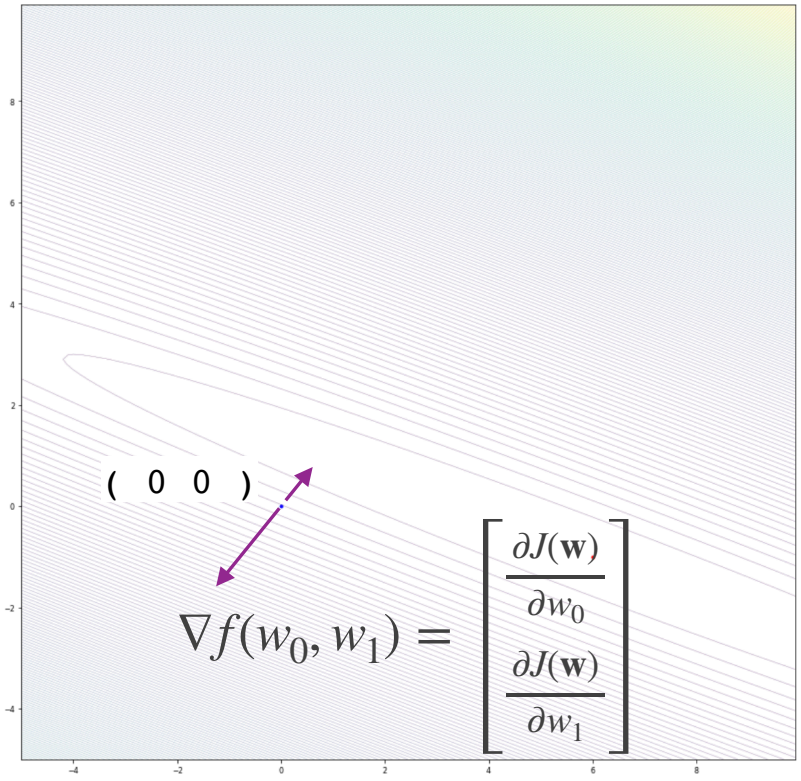
Lets look at minimizing a simpler function

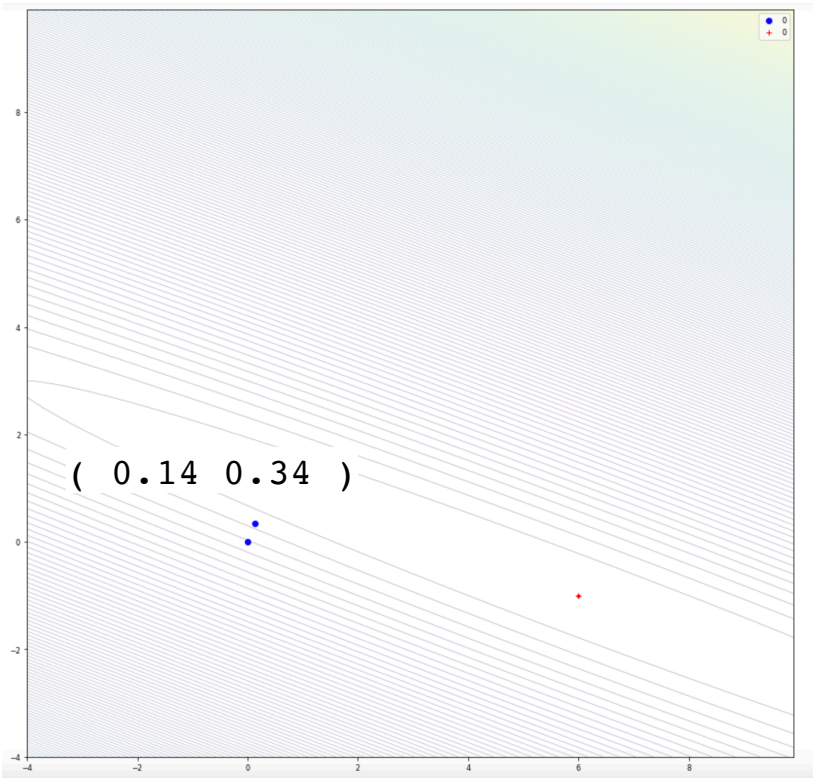
$$f(w_0, w_1) = (w_0 + 2 * w_1 - 4)^2 + (w_0 + 3 * w_1 - 3)^2$$

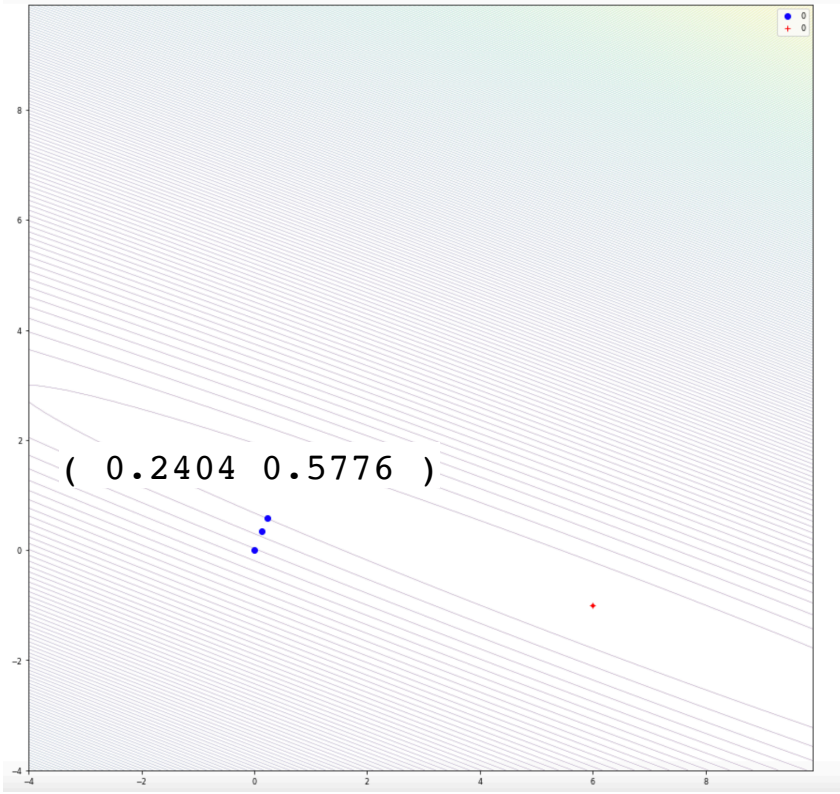


Local optimization to minimize a function

$$f(w_0, w_1) = (w_0 + 2 * w_1 - 4)^2 + (w_0 + 3 * w_1 - 3)^2$$





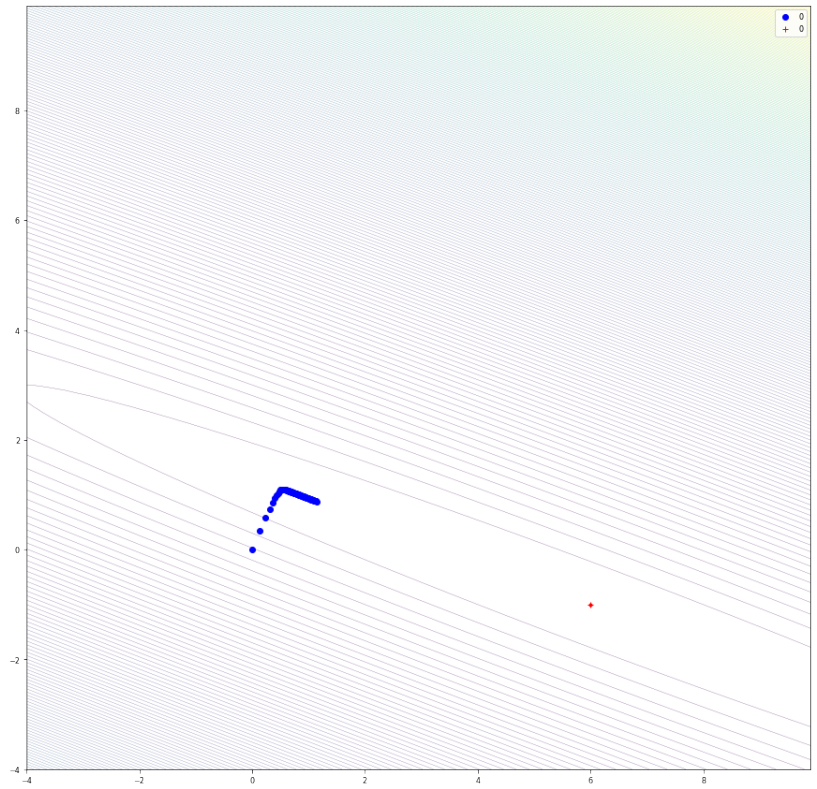


$$\text{temp0} = w_0 - \alpha \frac{\partial J(\mathbf{w})}{\partial w_0}$$

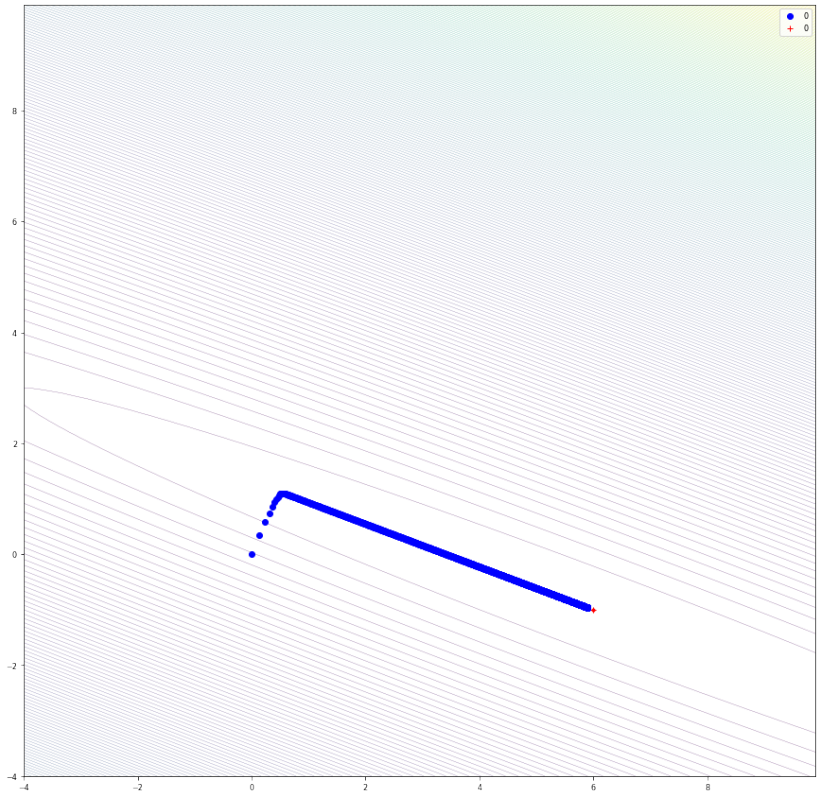
$$\text{temp1} = w_1 - \alpha \frac{\partial J(\mathbf{w})}{\partial w_1}$$

$$w_j = w_j - \alpha \frac{\partial f(\mathbf{w})}{\partial w_j}$$

$$f(w_0, w_1) = (w_0 + 2 * w_1 - 4)^2 + (w_0 + 3 * w_1 - 3)^2$$



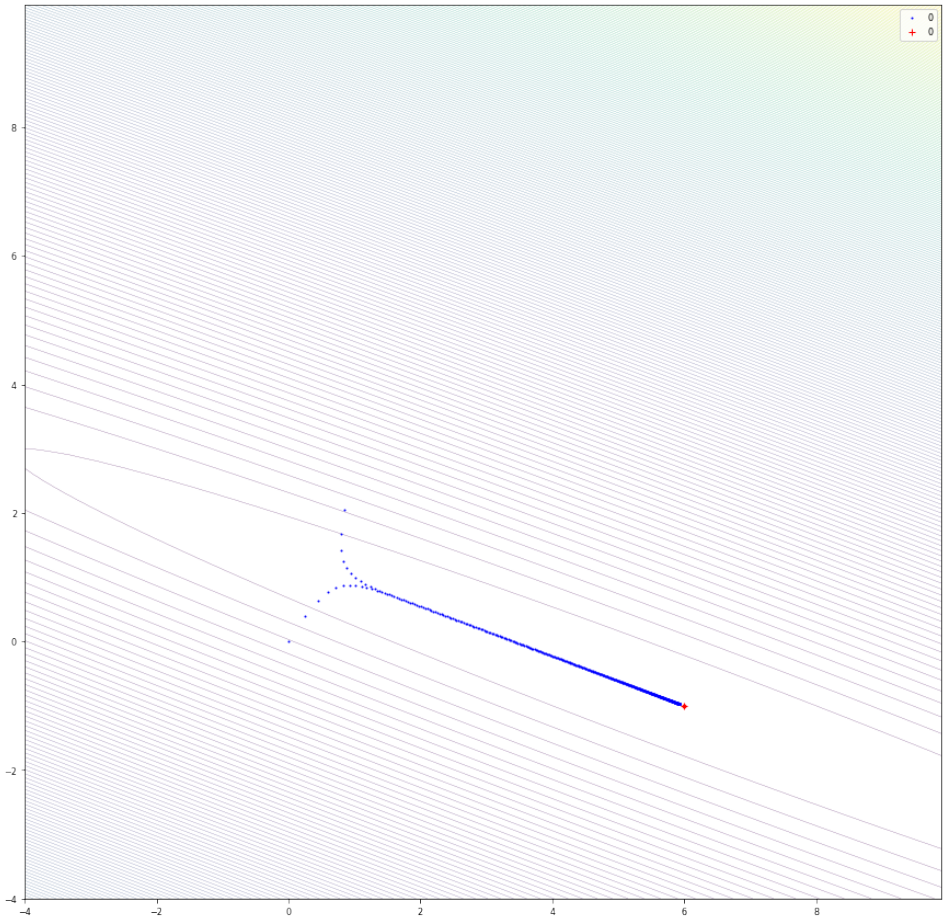
After 100 iterations



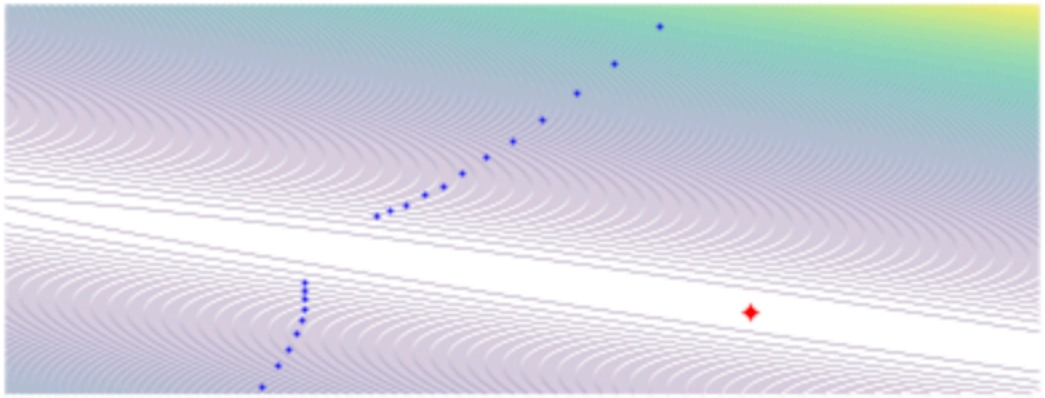
After 3000 iterations

What if we used a larger learning rate?

$$\begin{bmatrix}0\\0\end{bmatrix} \rightarrow \begin{bmatrix}0.84\\2.04\end{bmatrix} \rightarrow \begin{bmatrix}0.25\\0.39\end{bmatrix} \rightarrow \begin{bmatrix}0.780\\1.67\end{bmatrix} \rightarrow \begin{bmatrix}0.45\\0.623\end{bmatrix} \rightarrow \cdots$$



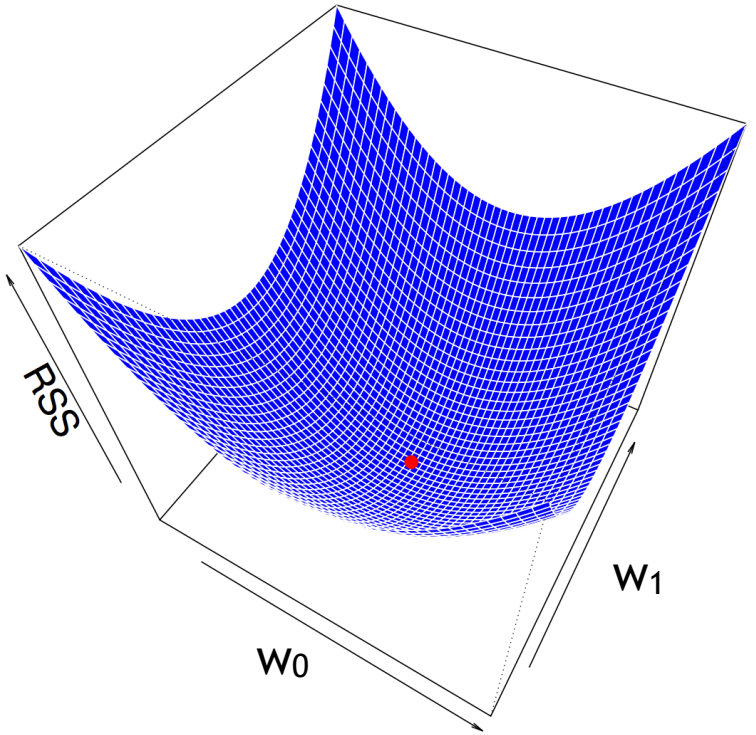
$$\begin{bmatrix}0\\0\end{bmatrix} \rightarrow \begin{bmatrix}0.98\\2.38\end{bmatrix} \rightarrow \begin{bmatrix}0.02\\-0.26\end{bmatrix} \rightarrow \begin{bmatrix}1.17\\2.58\end{bmatrix} \rightarrow \begin{bmatrix}0.02\\-0.56\end{bmatrix} \rightarrow \cdots$$



Which cost function should we use?

$$RSS(\mathbf{w}) = \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2 = \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)})^2 = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

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



"Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani "

Updated loss function

General ML problem

- ☑ Get data
- ☑ Pick a model with parameters
- ☑ Pick a loss function
  - Measures goodness of fit model to data
  - Function of the parameters
- ☐ Find parameters that minimizes loss

Multiple linear regression

Data: $(\mathbf{x}^{(i)}, y^{(i)}), i = 1,2,\ldots,N$

Linear model:

$$\hat{y}^{(j)} = w_0 + w_1 x_1^{(j)} + w_2 x_2^{(j)} + \cdots + w_d x_d^{(j)}$$

Loss function: $\frac{\text{RSS}(\mathbf{w})}{2N} = \frac{1}{2N} \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2$

$$= \frac{1}{2N} \left[(y^{(1)} - \hat{y}^{(1)})^2 + (y^{(2)} - \hat{y}^{(2)})^2 + \cdots + (y^{(N)} - \hat{y}^{(N)})^2 \right]$$

Select $\mathbf{w} = [w_0, w_1, w_2, \ldots, w_d]^T$ to minimize $\frac{\text{RSS}(\mathbf{w})}{2N}$

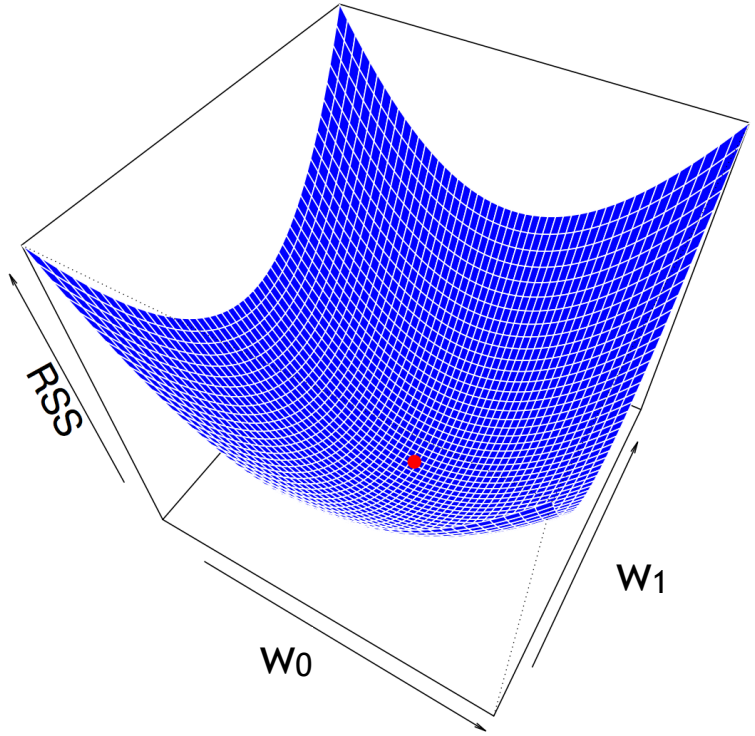
This $\mathbf{w}$ also minimizes $\text{RSS}(\mathbf{w})$

Partial Derivative

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

From calculus we know that the direction for the maximum rate of change for a function J(w) is

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{bmatrix}$$



$$w_0 - \alpha \frac{\partial J(\mathbf{w})}{\partial w_0}$$

$$w_1 - \alpha \frac{\partial J(\mathbf{w})}{\partial w_1}$$

$$\vdots$$

$$w_d - \alpha \frac{\partial J(\mathbf{w})}{\partial w_d}$$

The partial derivative of J(w)

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}((\mathbf{w}^T x^{(i)}) - y^{(i)})^2 = \frac{1}{2N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots + w_d x_d^{(i)} - y_i)^2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N} \cdot \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$

Verify at home 😊

$$= \frac{1}{N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots + w_d x_d^{(i)} - y^{(i)}) x_j^{(i)}$$

Slide not presented in class

$$\mathrm{J}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}((\mathbf{w}^T x^{(i)}) - y^{(i)})^2 = \frac{1}{2N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots + w_d x_d^{(i)} - y_i)^2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{2N}\cdot\frac{\partial}{\partial w_j}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2 = \frac{1}{2N}\cdot\sum_{i=1}^{N}\frac{\partial}{\partial w_j}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2$$

$$= \frac{1}{2N}\cdot\sum_{i=1}^{N}2(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})\frac{\partial}{\partial w_j}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}) = \frac{1}{N}\cdot\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)}$$

$$= \frac{1}{N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots + w_d x_d^{(i)} - y^{(i)})x_j^{(i)}$$

Cost function

- Minimizing $RSS(\mathbf{w}) = \sum_{i=1}^{N}((\mathbf{w}^T x^{(i)}) - y^{(i)})^2$ (our cost function for linear regression) is the same as minimizing:

$$\mathrm{J}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}((\mathbf{w}^T x^{(i)}) - y^{(i)})^2 = \frac{1}{2N}\mathrm{np.sum}(\mathrm{np.square}(X\mathbf{w} - y)) \quad = \frac{1}{2N} RSS(\mathbf{w})$$

- Both RSS and J are convex function (as was x^2), so we can use the gradient to find the minimum by taking a sequence of steps. Here is the derivative for the J function wrt the parameters:

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \frac{1}{N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots + w_d x_d^{(i)} - y_i)x_{i0} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T x^{(i)} - y^{(i)})x_0^{(i)}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots + w_d x_d^{(i)} - y^{(i)})x_j^{(i)} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T x^{(i)} - y^{(i)})x_j^{(i)}$$

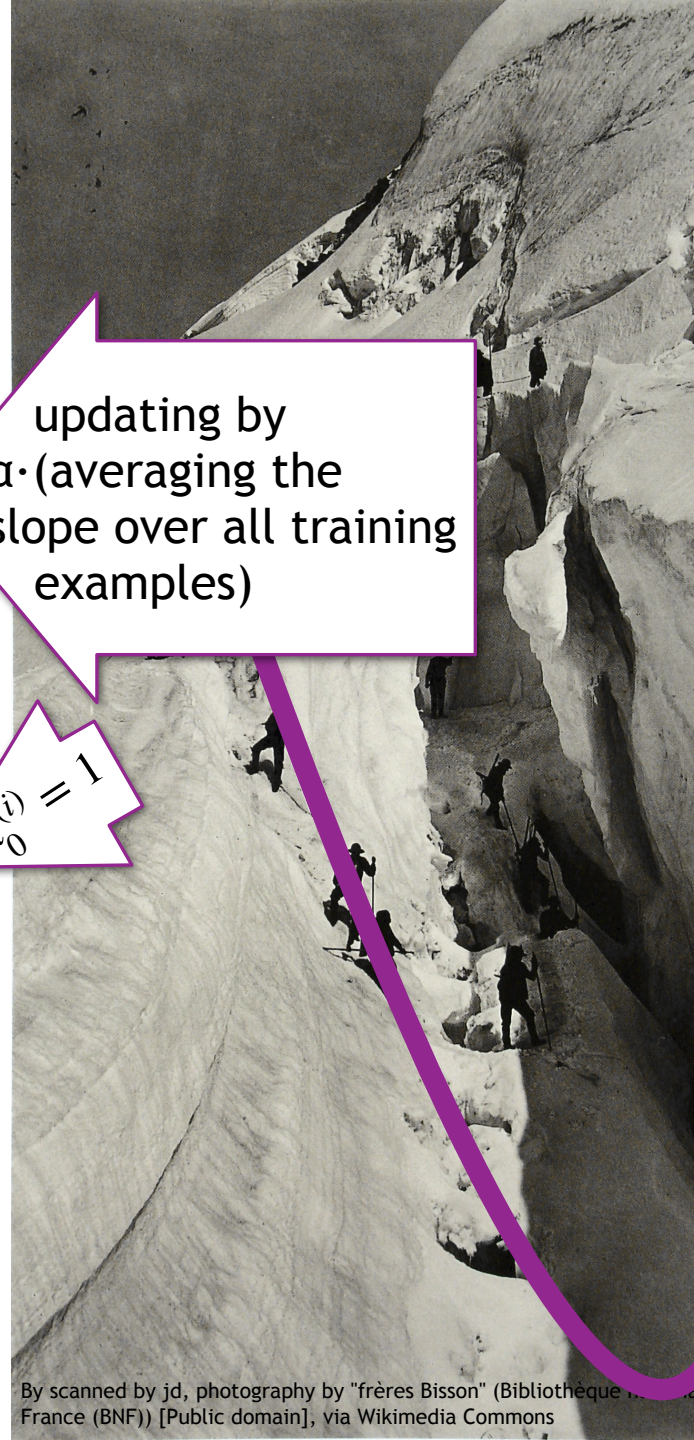
individual slope

average slope

A very nice explaination:
http://theory.stanford.edu/~tim/s16/l/l5.pdf
http://theory.stanford.edu/~tim/s16/l/l6.pdf

updating by α·(averaging the slope over all training examples)

$x_0^{(i)} = 1$



By scanned by jd, photography by "frères Bisson" (Bibliothèque [illegible] France (BNF)) [Public domain], via Wikimedia Commons

Gradient Descent Optimization

- The gradient is: $\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^i$
- To decrease the cost, we update the parameters, $\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$, using the update rule:

for $i = 1$ to num_iter

$$temp0 = w_0 - \alpha\frac{\partial J(\mathbf{w})}{\partial w_0} = w_0 - \frac{\alpha}{N}\sum_{i=1}^{N}(w^T\mathbf{x}^{(i)} - y^{(i)})x_0^{(i)}$$

$x_0^{(i)} = 1$

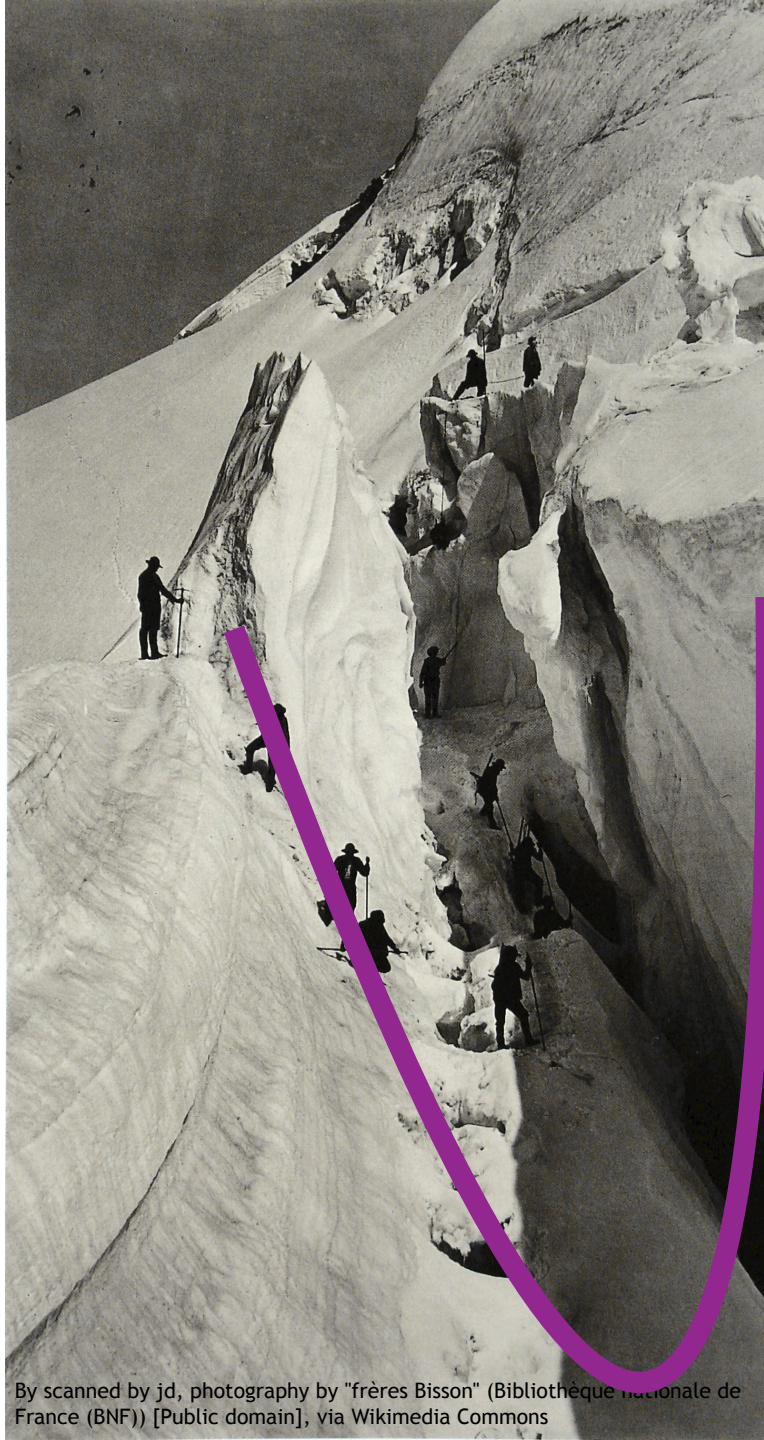
$$temp1 = w_1 - \alpha\frac{\partial J(\mathbf{w})}{\partial w_1} = w_1 - \frac{\alpha}{N}\sum_{i=1}^{N}(w^T\mathbf{x}^{(i)} - y^{(i)})x_1^{(i)}$$

$$\vdots$$

$$tempd = w_d - \alpha\frac{\partial J(\mathbf{w})}{\partial w_d} = w_d - \frac{\alpha}{N}\sum_{i=1}^{N}(\mathbf{w}^T x^{(i)} - y^{(i)})x_d^{(i)}$$

$$w_0 = temp0$$
$$w_1 = temp1$$
$$\vdots$$
$$w_d = tempd$$

Simultaneous update



By scanned by jd, photography by "frères Bisson" (Bibliothèque nationale de France (BNF)) [Public domain], via Wikimedia Commons

Vectorized Implementation

The next three slides were not presented during the class

Small example to show $\nabla J(\mathbf{w}) = \frac{1}{N} X^T(X\mathbf{w} - \mathbf{y})$ if we have one feature: $\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$.

The partial derivative of our cost function is: $\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$

If we have one feature and three examples ($N = 3$) the partial derivatives are:

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \frac{1}{3}\Big(\underbrace{(\mathbf{w}^T \mathbf{x}^{(1)})}_{\hat{y}^{(1)}} - y^{(1)} + \underbrace{(\mathbf{w}^T \mathbf{x}^{(2)})}_{\hat{y}^{(2)}} - y^{(2)} + \underbrace{(\mathbf{w}^T \mathbf{x}^{(3)})}_{\hat{y}^{(3)}} - y^{(3)} \Big) = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)} \end{bmatrix}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \frac{1}{3}\Big(\underbrace{(\mathbf{w}^T \mathbf{x}^{(1)})}_{\hat{y}^{(1)}} - y^{(1)} x_1^{(1)} + \underbrace{(\mathbf{w}^T \mathbf{x}^{(2)})}_{\hat{y}^{(2)}} - y^{(2)} x_1^{(2)} + \underbrace{(\mathbf{w}^T \mathbf{x}^{(3)})}_{\hat{y}^{(3)}} - y^{(3)} x_1^{(3)} \Big) = \frac{1}{3} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \end{bmatrix} \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)} \end{bmatrix}$$

Thus we can write the gradient for these three examples as:

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \end{bmatrix} \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)} \end{bmatrix}$$

Calculations continued on the next slide

Small example continued:

Using the fact that $X\mathbf{w} = \hat{\mathbf{y}}$, $d = 1$, and $N = 3$ we can show $\nabla J(\mathbf{w}) = \frac{1}{N} X^T (X\mathbf{w} - \mathbf{y})$

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \end{bmatrix} \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)} \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \end{bmatrix} \left(\begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ 1 & x_1^{(3)} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \end{bmatrix} \right) = \frac{1}{N} X^T (X\mathbf{w} - \mathbf{y})$$

2nd small example:

The partial derivative of our objective function $\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)}$

If we have two features and three examples ($N = 3$) the partial derivatives are:

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \frac{1}{3}\left((\mathbf{w}^T\mathbf{x}^{(1)} - y^{(1)}) + (\mathbf{w}^T\mathbf{x}^{(2)} - y^{(2)}) + (\mathbf{w}^T\mathbf{x}^{(3)} - y^{(3)}) \right) = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \frac{1}{3}\left((\mathbf{w}^T\mathbf{x}^{(1)} - y^{(1)})x_1^{(1)} + (\mathbf{w}^T\mathbf{x}^{(2)} - y^{(2)})x_1^{(2)} + (\mathbf{w}^T\mathbf{x}^{(3)} - y^{(3)})x_1^{(3)} \right) = \frac{1}{3}\begin{bmatrix}x_1^{(1)} & x_1^{(2)} & x_1^{(3)}\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_2} = \frac{1}{3}\left((\mathbf{w}^T\mathbf{x}^{(1)} - y^{(1)})x_2^{(1)} + (\mathbf{w}^T\mathbf{x}^{(2)} - y^{(2)})x_2^{(2)} + (\mathbf{w}^T\mathbf{x}^{(3)} - y^{(3)})x_2^{(3)} \right) = \frac{1}{3}\begin{bmatrix}x_2^{(1)} & x_2^{(2)} & x_2^{(3)}\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix}$$

Thus we can write the gradient for these three examples as:

$$\nabla J(\mathbf{w}) = \begin{bmatrix}\frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \frac{\partial J(\mathbf{w})}{\partial w_2}\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1 & 1 & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)}\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix} = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y})$$

The next two slides can be skipped - the new slides are clearer

If we have one feature:

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)} = \frac{1}{N}\sum_{i=1}^{N}(-\epsilon^{(i)})x_j^{(i)}$$

$$\nabla J(\mathbf{w}) = \frac{1}{N}X^T(X\mathbf{w}-\mathbf{y}) = \frac{1}{N}\begin{bmatrix}1 & 1 & \dots & 1\\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(N)}\end{bmatrix}\left[\begin{bmatrix}1 & x_1^{(1)}\\ 1 & x_1^{(2)}\\ \vdots & \\ 1 & x_1^{(N)}\end{bmatrix}\begin{bmatrix}w_0\\ w_1\end{bmatrix} - \begin{bmatrix}y^{(1)}\\ y^{(2)}\\ \vdots\\ y^{(N)}\end{bmatrix}\right] = \frac{1}{N}X^T(\hat{\mathbf{y}}-\mathbf{y})$$

$$= \frac{1}{N}\begin{bmatrix}1 & 1 & \dots & 1\\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(N)}\end{bmatrix}\left[\begin{bmatrix}\hat{y}^{(1)}\\ \hat{y}^{(2)}\\ \vdots\\ \hat{y}^{(N)}\end{bmatrix} - \begin{bmatrix}y^{(1)}\\ y^{(2)}\\ \vdots\\ y^{(N)}\end{bmatrix}\right] = \frac{1}{N}\begin{bmatrix}1 & 1 & \dots & 1\\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(N)}\end{bmatrix}\left[\begin{bmatrix}\hat{y}^{(1)}-y^{(1)}\\ \hat{y}^{(2)}-y^{(2)}\\ \vdots\\ \hat{y}^{(N)}-y^{(N)}\end{bmatrix}\right]$$

$$= \frac{1}{N}\begin{bmatrix}\sum_{i=1}^{N}\left(\hat{y}^{(i)}-y^{(i)}\right)\\ \sum_{i=1}^{N}\left(\hat{y}^{(i)}-y^{(i)}\right)x_1^{(i)}\end{bmatrix} = \frac{1}{N}\begin{bmatrix}\sum_{i=1}^{N}\left(w_0+w_1x_1^{(i)}-y^{(i)}\right)\\ \sum_{i=1}^{N}\left(w_0+w_1x_1^{(i)}-y^{(i)}\right)x_1^{(i)}\end{bmatrix} = \begin{bmatrix}\frac{\partial J(\mathbf{w})}{\partial w_0}\\ \frac{\partial J(\mathbf{w})}{\partial w_1}\end{bmatrix}$$

This slide can be skipped - the new slides are clearer

If we have two features:

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)} = \frac{1}{N}\sum_{i=1}^{N}(-\epsilon^{(i)})x_j^{(i)}$$

$$\nabla J(\mathbf{w}) = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y}) = \frac{1}{N}X^T(\hat{\mathbf{y}} - \mathbf{y})$$

$$\nabla J(\mathbf{w}) = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y}) = \frac{1}{N}\begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(N)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(N)} \end{bmatrix}\left[\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & & \\ 1 & x_1^{(N)} & x_2^{(3)} \end{bmatrix}\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}\right]$$

$$= \frac{1}{N}\begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(N)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(N)} \end{bmatrix}\left[\begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(N)} \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}\right] = \frac{1}{N}\begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(N)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(N)} \end{bmatrix}\left[\begin{bmatrix} \hat{y}^{(1)} - y^{(i)} \\ \hat{y}^{(2)} - y^{(i)} \\ \vdots \\ \hat{y}^{(N)} - y^{(i)} \end{bmatrix}\right]$$

$$= \frac{1}{N}\begin{bmatrix} \sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right) \\ \sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)x_1^{(i)} \\ \sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)x_2^{(i)} \end{bmatrix} = \frac{1}{N}\begin{bmatrix} \sum_{i=1}^{N}\left(w_0 + w_1x_1^{(i)}w_1 + w_2x_2^{(i)} - y^{(i)}\right) \\ \sum_{i=1}^{N}\left(w_0 + w_1x_1^{(i)} + w_2x_2^{(i)} - y^{(i)}\right)x_1^{(i)} \\ \sum_{i=1}^{N}\left(w_0 + w_1x_1^{(i)} + w_2x_2^{(i)} - y^{(i)}\right)x_2^{(i)} \end{bmatrix} = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \frac{\partial J(\mathbf{w})}{\partial w_2} \end{bmatrix}$$

This slide can be skipped - the new slides are clearer

Example

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)} = \frac{1}{N}\sum_{i=1}^{N}(-\epsilon^{(i)})x_j^{(i)}$$

$$\nabla J(\mathbf{w}) = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y}) = \frac{1}{N}X^T(\hat{\mathbf{y}} - \mathbf{y})$$

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{bmatrix} = \frac{1}{N}$$

1	1	1	1	1	1	1	1
0.038	−0.002	0.085	−0.089	0.005	−0.093	−0.045	⋮
0.051	−0.045	0.051	−0.045	−0.045	−0.045	0.051	
0.062	−0.051	0.044	−0.012	−0.036	−0.041	−0.047	
0.022	−0.026	−0.006	−0.037	0.022	−0.019	−0.016	
−0.044	−0.008	−0.046	0.012	0.004	−0.069	−0.04	
−0.035	−0.019	−0.034	0.025	0.016	−0.079	−0.025	
−0.043	0.074	−0.032	−0.036	0.008	0.041	0.001	
−0.003	−0.039	−0.003	0.034	−0.003	−0.076	−0.039	
0.02	−0.068	0.003	0.023	−0.032	−0.041	−0.063	
−0.018	−0.092	−0.026	−0.009	−0.047	−0.096	−0.038	

	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
1	0.038	0.051	0.062	$2.187e-02$	−0.044	$-3.482e-02$	0.043	−0.003	0.020	−0.018
1	−0.002	−0.045	−0.051	$-2.633e-02$	−0.008	$-1.916e-02$	−0.074	−0.039	−0.068	−0.092
1	0.085	0.051	0.044	$-5.670e-03$	−0.046	$-3.419e-02$	0.032	−0.003	0.003	−0.026
1	−0.089	−0.045	−0.012	$-3.666e-02$	0.012	$2.499e-02$	0.036	0.034	0.023	−0.009
1	0.005	−0.045	−0.036	$2.187e-02$	0.004	$1.560e-02$	−0.008	−0.003	−0.032	−0.047
1	−0.093	−0.045	−0.041	$-1.944e-02$	−0.069	$-7.929e-02$	−0.041	−0.076	−0.041	−0.096
1	−0.046	0.051	−0.047	$-1.600e-02$	−0.040	$-2.480e-02$	−0.001	−0.039	−0.063	−0.038
⋮										

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \\ w_{10} \end{bmatrix} - \begin{bmatrix} 204.51116637 \\ 67.10485972 \\ 175.02956894 \\ 165.88615565 \\ 123.11207835 \\ 105.64709238 \\ 71.73293158 \end{bmatrix}$$

$$= \frac{1}{N}$$

1	1	1	1	1	1	1	1
0.038	−0.002	0.085	−0.089	0.005	−0.093	−0.045	⋮
0.051	−0.045	0.051	−0.045	−0.045	−0.045	0.051	
0.062	−0.051	0.044	−0.012	−0.036	−0.041	−0.047	
0.022	−0.026	−0.006	−0.037	0.022	−0.019	−0.016	
−0.044	−0.008	−0.046	0.012	0.004	−0.069	−0.04	
−0.035	−0.019	−0.034	0.025	0.016	−0.079	−0.025	
−0.043	0.074	−0.032	−0.036	0.008	0.041	0.001	
−0.003	−0.039	−0.003	0.034	−0.003	−0.076	−0.039	
0.02	−0.068	0.003	0.023	−0.032	−0.041	−0.063	
−0.018	−0.092	−0.026	−0.009	−0.047	−0.096	−0.038	

$$\begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \hat{y}^{(3)} \\ \hat{y}^{(4)} \\ \hat{y}^{(5)} \\ \hat{y}^{(6)} \\ \hat{y}^{(7)} \end{bmatrix} - \begin{bmatrix} 204.51116637 \\ 67.10485972 \\ 175.02956894 \\ 165.88615565 \\ 123.11207835 \\ 105.64709238 \\ 71.73293158 \end{bmatrix} = \frac{1}{N}$$

1	1	1	1	1	1	1	1
0.038	−0.002	0.085	−0.089	0.005	−0.093	−0.045	⋮
0.051	−0.045	0.051	−0.045	−0.045	−0.045	0.051	
0.062	−0.051	0.044	−0.012	−0.036	−0.041	−0.047	
0.022	−0.026	−0.006	−0.037	0.022	−0.019	−0.016	
−0.044	−0.008	−0.046	0.012	0.004	−0.069	−0.04	
−0.035	−0.019	−0.034	0.025	0.016	−0.079	−0.025	
−0.043	0.074	−0.032	−0.036	0.008	0.041	0.001	
−0.003	−0.039	−0.003	0.034	−0.003	−0.076	−0.039	
0.02	−0.068	0.003	0.023	−0.032	−0.041	−0.063	
−0.018	−0.092	−0.026	−0.009	−0.047	−0.096	−0.038	

$$\begin{bmatrix} -\epsilon^{(1)} \\ -\epsilon^{(2)} \\ -\epsilon^{(3)} \\ -\epsilon^{(4)} \\ -\epsilon^{(5)} \\ -\epsilon^{(6)} \\ -\epsilon^{(7)} \end{bmatrix} = \nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{bmatrix}$$

Gradient Descent Optimization Using Vectorization

$$\nabla J(\mathbf{w}) = \frac{1}{N} X^T (X\mathbf{w} - \mathbf{y})$$

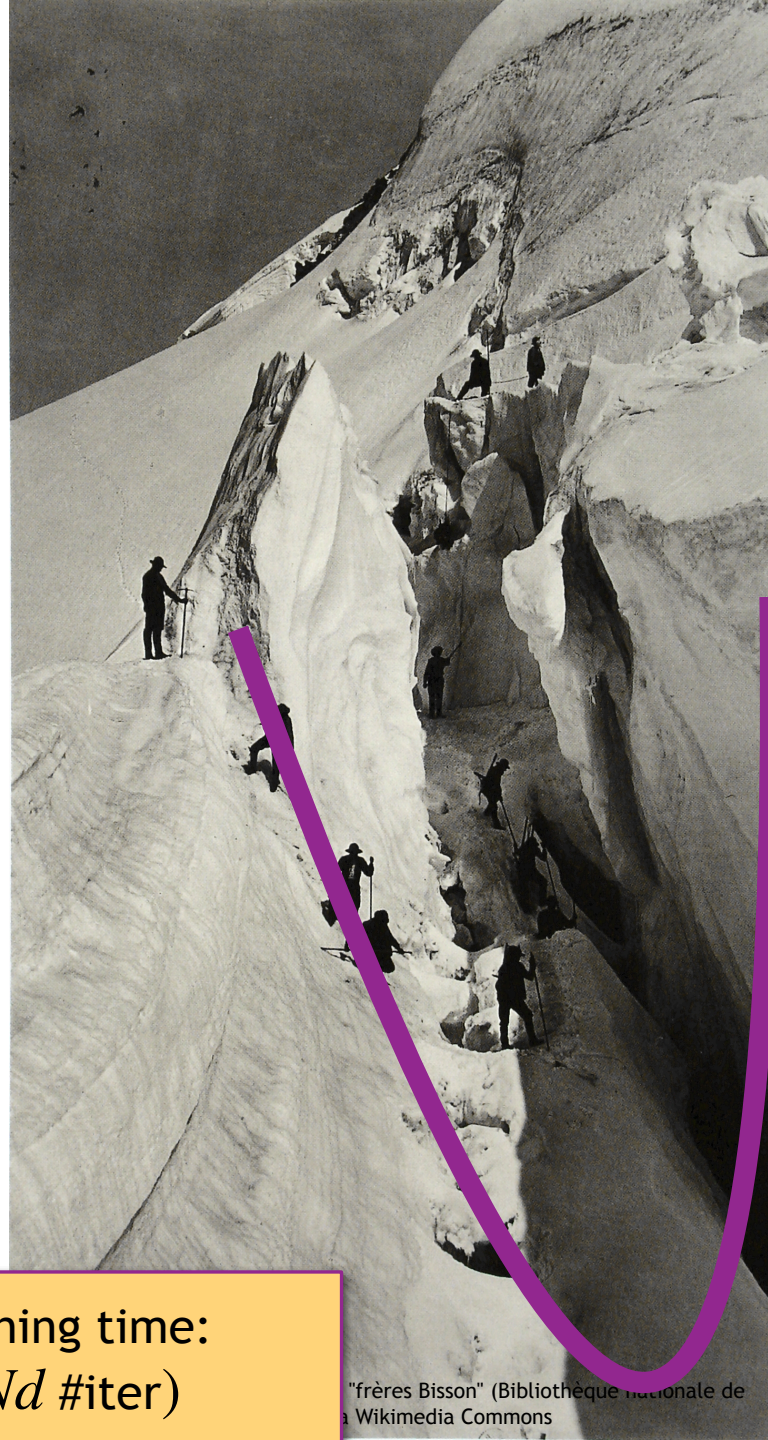
- We can perform gradient update for all parameters $\mathbf{w} = [w_0, w_1, \ldots, w_d]$ in a single line of vectorized code

for $i = 1$ to num_iter

$$\mathbf{w} = \mathbf{w} - \alpha \nabla J(\mathbf{w}) = \mathbf{w} - \alpha \frac{1}{N} X^T (X\mathbf{w} - \mathbf{y})$$

Simultaneous update

- Note that:
  - $\mathbf{w}$ and $\nabla J(\mathbf{w})$ are $(d+1) \times 1$ column vectors
  - all $\mathbf{w}$ are getting updated simultaneously and we do not need to store them in temporary variable

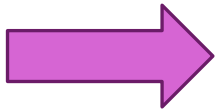


Running time: $O(Nd \#\text{iter})$

"frères Bisson" (Bibliothèque nationale de
Wikimedia Commons

Outline

- Motivating Example: Understanding glucose levels in diabetes patients
- Multiple variable linear models
- Least squares solutions
  - Gradient descent
  - Normal Equations
  - Feature scaling
- Evaluating our hypothesis
- Computing the solutions in python
- Special case: Simple linear regression
- Extensions
- Removing features



Global Optimization

Our second method to finding the parameters $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$

Updated loss function

General ML problem

- ❑ Get data
- ❑ Pick a model with parameters
- ❑ Pick a loss function
  - Measures goodness of fit model to data
  - Function of the parameters
- ❑ Find parameters that minimizes loss

Multiple linear regression

Data: $(\mathbf{x}^{(i)}, y^{(i)}), i = 1,2,\ldots,N$

Linear model:

$$\hat{y}^{(j)} = w_0 + w_1 x_1^{(j)} + w_2 x_2^{(j)} + \cdots + w_d x_d^{(j)}$$

Loss function: $\frac{\text{RSS}(\mathbf{w})}{2N} = \frac{1}{2N} \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2$

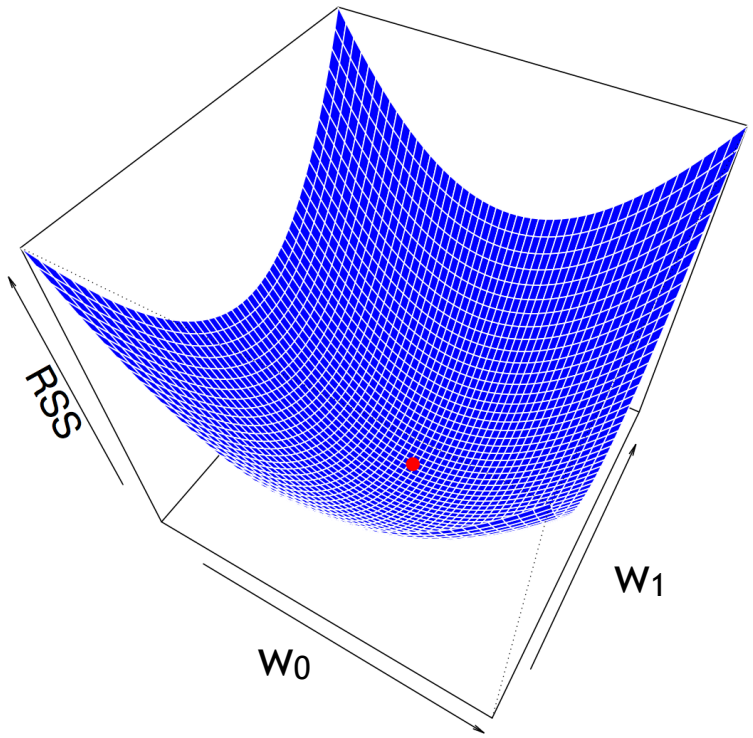
$$= \frac{1}{2N} \left[(y^{(1)} - \hat{y}^{(1)})^2 + (y^{(2)} - \hat{y}^{(2)})^2 + \cdots + (y^{(N)} - \hat{y}^{(N)})^2 \right]$$

Select $\mathbf{w} = [w_0, w_1, w_2, \ldots, w_d]^T$ to minimize $\frac{\text{RSS}(\mathbf{w})}{2N}$

This $\mathbf{w}$ also minimizes $\text{RSS}(\mathbf{w})$

Global optimization for finding **w** to minimize

$$\frac{1}{2}E_{in} = J(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$



$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{bmatrix} = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Goal find $\mathbf{w}$ such that $\nabla E_{in}(\mathbf{w}) = \mathbf{0}$

"Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani "

Finding **w** to minimize E_{in}

Goal find $\mathbf{w}_{\text{lin}}$ such that $\nabla E_{\text{in}}(\mathbf{w}) = \mathbf{0}$

$$\nabla J(\mathbf{w}) = \frac{1}{N} X^T (X\mathbf{w} - \mathbf{y}) = \frac{1}{N} (X^T X\mathbf{w} - X^T \mathbf{y})$$

Setting $\nabla J(\mathbf{w}) = \frac{1}{N} (X^T X\mathbf{w} - X^T \mathbf{y}) = \mathbf{0}$

Results in: $X^T X\mathbf{w} = X^T \mathbf{y}$

Thus $\mathbf{w}_{\text{lin}} = \underbrace{\left(X^T X\right)^{-1} X^T}_{\text{pseudoinverse}} \mathbf{y}$

pseudoinverse
left inverse

If the columns of the mat
X are linearly independen
then $X^T X$ is invertible

$$X^+ = (X^T X)^{-1} X^T$$

is the pseudo inverse, i.e.
the left inverse of X

$$X^+ X = (X^T X)^{-1} X^T X = I$$

Running time:
$O(Nd^2)$ for $X^T X$
$O(d^3)$ for inverse $(X^T X)^{-1}$
$O(dN)$ for $X^T \mathbf{y}$
$O(d^2)$ for $(X^T X)^{-1}$ times $X^T \mathbf{y}$
————————————————————
Total $O(Nd^2) + O(d^3)$

*finding the inverse (or pseudoinverse) of a $d \times d$ matrix can be found in $O(d^{2.373})$ time

We added 'lin' to **w** to specify it was linear regression

$X^T X$ is a $(d+1) \times (d+1)$ matrix
$(X^T X)^{-1}$ is a $(d+1) \times (d+1)$ matrix
$(X^T X)^{-1} X^T$ is a $(d+1) \times N$ matrix
$(X^T X)^{-1} X^T \mathbf{y}$ is $(d+1) \times 1$

http://sci.utah.edu/~gerig/CS6640-F2012/Materials/pseudoinverse-cis61009sl10.pdf

Finding **w** to minimize E_{in} (and RSS)

Linear Regression Algorithm:

1. Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$, where each $\mathbf{x}$ includes the $x_0 = 1$ coordinate,

$$\underbrace{\mathrm{X} = \begin{bmatrix} -\mathbf{x}_1- \\ -\mathbf{x}_2- \\ \vdots \\ -\mathbf{x}_N- \end{bmatrix}}_{\text{data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

2. Compute the pseudo inverse $\mathrm{X}^\dagger$ of the matrix X. If $\mathrm{X}^\mathrm{T}\mathrm{X}$ is invertible,

$$\mathrm{X}^\dagger = (\mathrm{X}^\mathrm{T}\mathrm{X})^{-1}\mathrm{X}^\mathrm{T}$$

3. Return $\mathbf{w}_{\mathrm{lin}} = \mathrm{X}^\dagger\mathbf{y}$.

Algorithm from book Learning form Data

Example

$$\mathbf{w}_{\text{lin}} = \left(X^T X\right)^{-1} X^T \mathbf{y}$$

$$\mathbf{w}_{\text{lin}} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \\ w_{10} \end{bmatrix} = \left[\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ 0.038 & -0.002 & 0.085 & -0.089 & 0.005 & -0.093 & -0.045 & \\ 0.051 & -0.045 & 0.051 & -0.045 & -0.045 & -0.045 & 0.051 & \\ 0.062 & -0.051 & 0.044 & -0.012 & -0.036 & -0.041 & -0.047 & \\ 0.022 & -0.026 & -0.006 & -0.037 & 0.022 & -0.019 & -0.016 & \\ -0.044 & -0.008 & -0.046 & 0.012 & 0.004 & -0.069 & -0.04 & \\ -0.035 & -0.019 & -0.034 & 0.025 & 0.016 & -0.079 & -0.025 & \\ -0.043 & 0.074 & -0.032 & -0.036 & 0.008 & 0.041 & 0.001 & \\ -0.003 & -0.039 & -0.003 & 0.034 & -0.003 & -0.076 & -0.039 & \\ 0.02 & -0.068 & 0.003 & 0.023 & -0.032 & -0.041 & -0.063 & \\ -0.018 & -0.092 & -0.026 & -0.009 & -0.047 & -0.096 & -0.038 & \end{bmatrix} \begin{bmatrix} \text{age} & \text{sex} & \text{bmi} & \text{map} & \text{tc} & \text{ldl} & \text{hdl} & \text{tch} & \text{ltg} & \text{glu} \\ 1 & 0.038 & 0.051 & 0.062 & 2.187e-02 & -0.044 & -3.482e-02 & 0.043 & -0.003 & 0.020 & -0.018 \\ 1 & -0.002 & -0.045 & -0.051 & -2.633e-02 & -0.008 & -1.916e-02 & -0.074 & -0.039 & -0.068 & -0.092 \\ 1 & 0.085 & 0.051 & 0.044 & -5.670e-03 & -0.046 & -3.419e-02 & 0.032 & -0.003 & 0.003 & -0.026 \\ 1 & -0.089 & -0.045 & -0.012 & -3.666e-02 & 0.012 & 2.499e-02 & 0.036 & 0.034 & 0.023 & -0.009 \\ 1 & 0.005 & -0.045 & -0.036 & 2.187e-02 & 0.004 & 1.560e-02 & -0.008 & -0.003 & -0.032 & -0.047 \\ 1 & -0.093 & -0.045 & -0.041 & -1.944e-02 & -0.069 & -7.929e-02 & -0.041 & -0.076 & -0.041 & -0.096 \\ 1 & -0.046 & 0.051 & -0.047 & -1.600e-02 & -0.040 & -2.480e-02 & -0.001 & -0.039 & -0.063 & -0.038 \\ \vdots \end{bmatrix} \right]^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ 0.038 & -0.002 & 0.085 & -0.089 & 0.005 & -0.093 & -0.045 & \\ 0.051 & -0.045 & 0.051 & -0.045 & -0.045 & -0.045 & 0.051 & \\ 0.062 & -0.051 & 0.044 & -0.012 & -0.036 & -0.041 & -0.047 & \\ 0.022 & -0.026 & -0.006 & -0.037 & 0.022 & -0.019 & -0.016 & \\ -0.044 & -0.008 & -0.046 & 0.012 & 0.004 & -0.069 & -0.04 & \\ -0.035 & -0.019 & -0.034 & 0.025 & 0.016 & -0.079 & -0.025 & \\ -0.043 & 0.074 & -0.032 & -0.036 & 0.008 & 0.041 & 0.001 & \\ -0.003 & -0.039 & -0.003 & 0.034 & -0.003 & -0.076 & -0.039 & \\ 0.02 & -0.068 & 0.003 & 0.023 & -0.032 & -0.041 & -0.063 & \\ -0.018 & -0.092 & -0.026 & -0.009 & -0.047 & -0.096 & -0.038 & \end{bmatrix} \begin{bmatrix} 204.51116637 \\ 67.10485972 \\ 175.02956894 \\ 165.88615565 \\ 123.11207835 \\ 105.64709238 \\ 71.73293158 \end{bmatrix}$$

$$\mathbf{w}_{\text{lin}} = \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \\ w_{10} \end{bmatrix} = \begin{bmatrix} 152.34786452 \\ -16.57607993 \\ -254.66532396 \\ 560.98630022 \\ 278.91811152 \\ -393.41357305 \\ 97.05460405 \\ -19.0023093 \\ 169.46450327 \\ 632.95050374 \\ 114.21638941 \end{bmatrix}$$

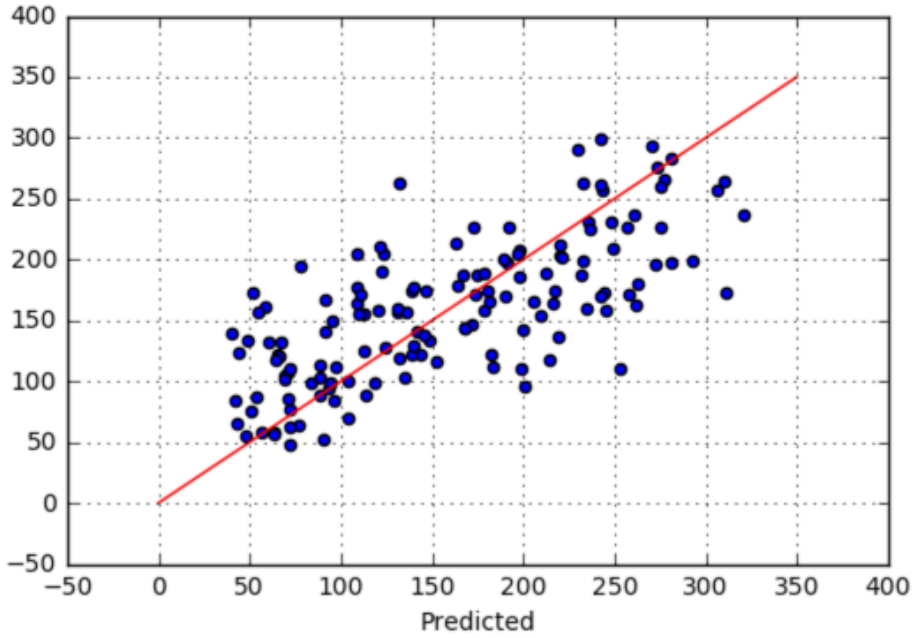
Python code: `w_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)`

Python code using the pseudoinverse: `w_best = np.linalg.pinv(X_b).dot(y)`

https://numpy.org/doc/stable/reference/generated/numpy.linalg.pinv.html

Best fitting coefficients/weights

- The difference between the predicted and the true values



w=

$$\begin{bmatrix} 152.34786452 \\ -16.57607993 \\ -254.66532396 \\ 560.98630022 \\ 278.91811152 \\ -393.41357305 \\ 97.05460405 \\ -19.0023093 \\ 169.46450327 \\ 632.95050374 \\ 114.21638941 \end{bmatrix}$$

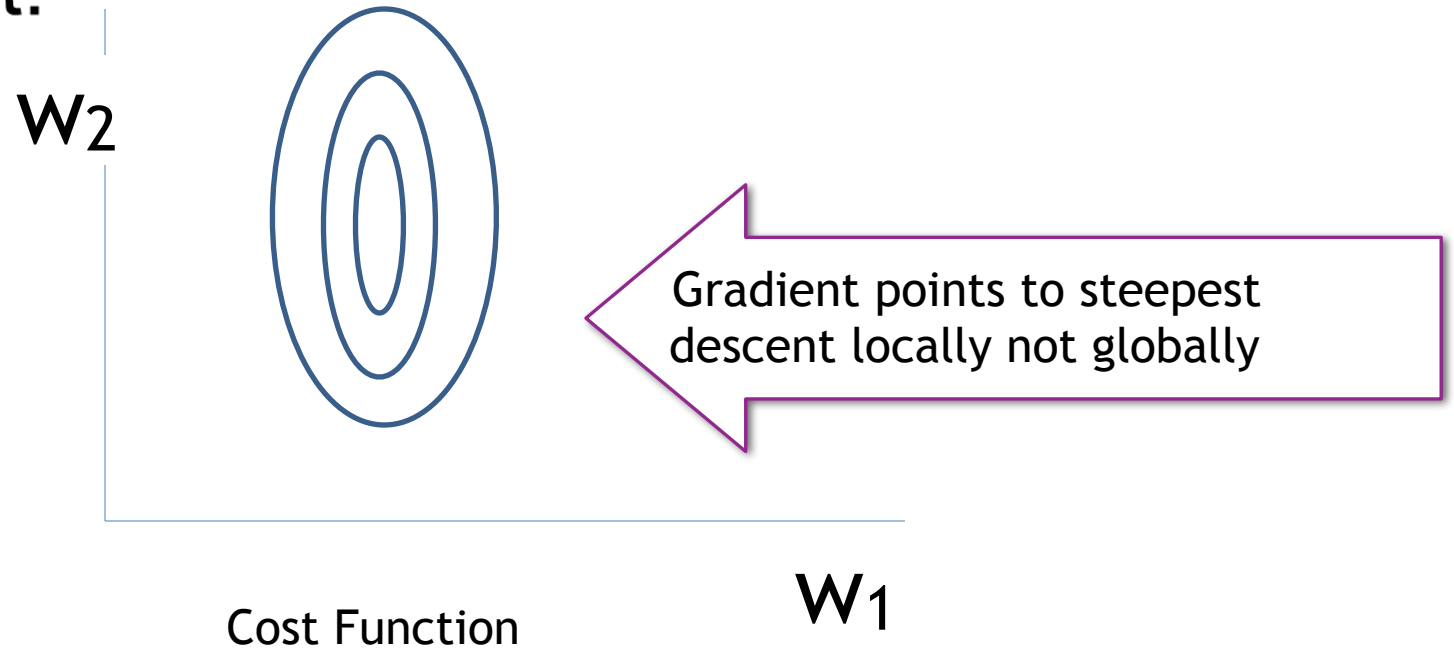
$R^2 = 0.514719$

Outline

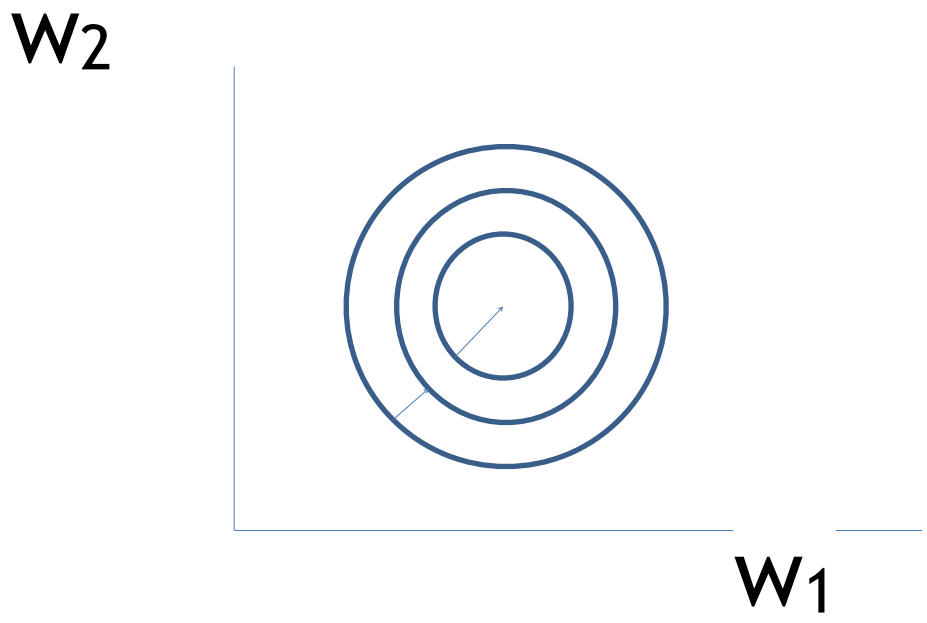
- Motivating Example: Understanding glucose levels in diabetes patients
- Multiple variable linear models
- Least squares solutions
  - Gradient descent
  - Normal Equations
  - Feature scaling
- Evaluating our hypothesis
- Computing the solutions in python
- Special case: Simple linear regression
- Extensions
- Removing features

Cost Function

- Visualizing cost function using a contour plot.



After Scaling



Feature Scaling
AKA Data Normalization

- Before applying many machine learning algorithms make sure that the features that are on a similar scale to prevent one feature from overly influencing the algorithm.
- Feature scaling is typically done before performing gradient descent to improve the rate the algorithm converges
- Eg: Say you are using 2 features for predicting housing price problem:
  - X_1 = size (0 – 4000 sq ft)
  - X_2 = No. of bedrooms (1 – 5)

Types of Feature Scaling
AKA Data Normalization

- Min/Max Normalization
- Standardization
- For other methods see https://en.wikipedia.org/wiki/Feature_scaling

Min/Max Normalization

- Scale the range of features to become [0,1] (or [a,b])
- Steps:

For each feature, j, in the data (except x_0)

- Find the range of values $[\min(x_j), \max(x_j)]$ for the jth feature
- Update every example's jth feature:

$$x_j^{(i)} = \frac{x_j^{(i)} - \min(x_j)}{\max(x_j) - \min(x_j)}$$

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
>>> scaler = MinMaxScaler()
>>> print(scaler.fit(data))
MinMaxScaler()
>>> print(scaler.data_max_)
[ 1. 18.]
>>> print(scaler.transform(data))
[[0.   0.  ]
 [0.25 0.25]
 [0.5  0.5 ]
 [1.   1.  ]]
>>> print(scaler.transform([[2, 2]]))
[[1.5 0. ]]
```

$$x_1^{(i)} = \frac{x_1^{(i)} - (-1)}{1 - (-1)} \qquad x_2^{(i)} = \frac{x_2^{(i)} - 2}{18 - 2}$$

Eg: If the range of feature 1 is [0, 4000]

update all training example such that $x^{(i)}_1 = (x^{(i)}_1 - 0)/4000$. Now the range of $x^{(i)}_1$ is [0,1]

If the range if the second feature is [1,5] update all training examples $x^{(i)}_2 = (x^{(i)}_2 - 1)/4$.
Now the range of the second feature is [0, 1]

Standardization (in social science this is call Z-score normalization)

```
>>> from sklearn.preprocessing import StandardScaler
>>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]
>>> scaler = StandardScaler()
>>> print(scaler.fit(data))
StandardScaler()
>>> print(scaler.mean_)
[0.5 0.5]
>>> print(scaler.transform(data))
[[-1. -1.]
 [-1. -1.]
 [ 1.  1.]
 [ 1.  1.]]
>>> print(scaler.transform([[2, 2]]))
[[3. 3.]]
```

$$x_1^{(i)} = \frac{x_1^{(i)} - 0.5}{0.5}$$

- Scale the range of features to become zero centered and have unit variance
- Steps:

  For each feature, j, in the data

  - Find the average for the jth feature: $\text{ave}(x_j) = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}$
  - Find the standard deviation for the jth feature: $\text{STD}(x_j) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_j^{(i)} - \text{ave}(x_j)\right)^2}$
  - Update the jth feature

$$x_j^{(i)} = \frac{x_j^{(i)} - \text{ave}(x_j)}{\text{STD}(x_j)}$$

Eg: If the average value of feature j is 70, and the standard deviation is 12 update all training example such that $x_j^{(i)} = \frac{x_j^{(i)} - 70}{12}$ Now the average value of feature j is 0, and the standard deviation of feature j for the training examples is 1

Diabetes dataset
Mean Centering and Scaling to Unit Length example

"these data are first standardized to have zero mean and unit L2 norm before they are used in the examples."

Original Data

age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
59	2	32.1	101	157	93.2	38	4	4.8598	87
48	1	21.6	87	183	103.2	70	3	3.8918	69
72	2	30.5	93	156	93.6	41	4	4.6728	85
24	1	25.3	84	198	131.4	40	5	4.8903	89
50	1	23	101	192	125.4	52	4	4.2905	80
23	1	22.6	89	139	64.8	61	2	4.1897	68
36	2	22	90	160	99.6	50	3	3.9512	82

The L2 norm is the standard deviations when the mean is zero

work area

$$\begin{bmatrix} \text{AGE} \\ 59 \\ 72 \\ 24 \\ 50 \\ 23 \\ 36 \end{bmatrix} - \begin{bmatrix} \text{mean} \\ 48.5 \\ 48.5 \\ 48.5 \\ 48.5 \\ 48.5 \\ 48.5 \end{bmatrix} = \begin{bmatrix} 10.5 \\ -0.5 \\ 23.5 \\ -24.5 \\ 1.5 \\ -25.5 \\ -12.5 \end{bmatrix}$$

l2 norm of the age feature is 275.3

$$\begin{bmatrix} 10.5/275.3 \\ -0.5/275.3 \\ 23.5/275.3 \\ -24.5/275.3 \\ 1.5/275.3 \\ -25.5/275.3 \\ -12.5/275.3 \end{bmatrix} = \begin{bmatrix} 0.038 \\ -0.002 \\ 0.085 \\ -0.089 \\ 0.0054 \\ -0.093 \\ -0.045 \end{bmatrix}$$

$X =$

	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
1	0.038	0.051	0.062	2.187e-02	-0.044	-3.482e-02	0.043	-0.003	0.020	-0.018
1	-0.002	-0.045	-0.051	-2.633e-02	-0.008	-1.916e-02	-0.074	-0.039	-0.068	-0.092
1	0.085	0.051	0.044	-5.670e-03	-0.046	-3.419e-02	0.032	-0.003	0.003	-0.026
1	-0.089	-0.045	-0.012	-3.666e-02	0.012	2.499e-02	0.036	0.034	0.023	-0.009
1	0.005	-0.045	-0.036	2.187e-02	0.004	1.560e-02	-0.008	-0.003	-0.032	-0.047
1	-0.093	-0.045	-0.041	-1.944e-02	-0.069	-7.929e-02	-0.041	-0.076	-0.041	-0.096
1	-0.046	0.051	-0.047	-1.600e-02	-0.040	-2.480e-02	-0.001	-0.039	-0.063	-0.038

Data from https://web.stanford.edu/~hastie/Papers/LARS/diabetes.data

Learn more at https://en.wikipedia.org/wiki/Feature_scaling

I used the full dataset when I computed the mean and the l2 norm