CS2109S: Introduction to AI and Machine Learning

Lecture 5: Linear Regression

11 February 2024

Midterm – Reminder

- Date & Time:
  - **Tuesday, 4 March 2025**, from **18:30** to **20:00**
- Venue:
  - **MPSH 2A & 2B**
- Format:
  - Digital Assessment (**Examplify**)
- Materials:
  - All topics covered **until and including Lecture 6**
- Cheatsheet:
  - **1 x A4 paper, both sides**
- Calculators:
  - **Standard** and **scientific calculators** are allowed.
  - No graphing/programmable calculators.

More details will be announced later.

Midterm – Examplify

All the info:

https://nus.atlassian.net/wiki/spaces/DAstudent/overview

Video

https://mediaweb.ap.panopto.com/Panopto/Pages/Viewer.aspx?id=48df9509-7daf-41f4-9ee8-ae22008a7383

Common briefing:

https://nus.atlassian.net/wiki/spaces/DAstudent/pages/22511675/Common+Briefing+Sessions

Outline

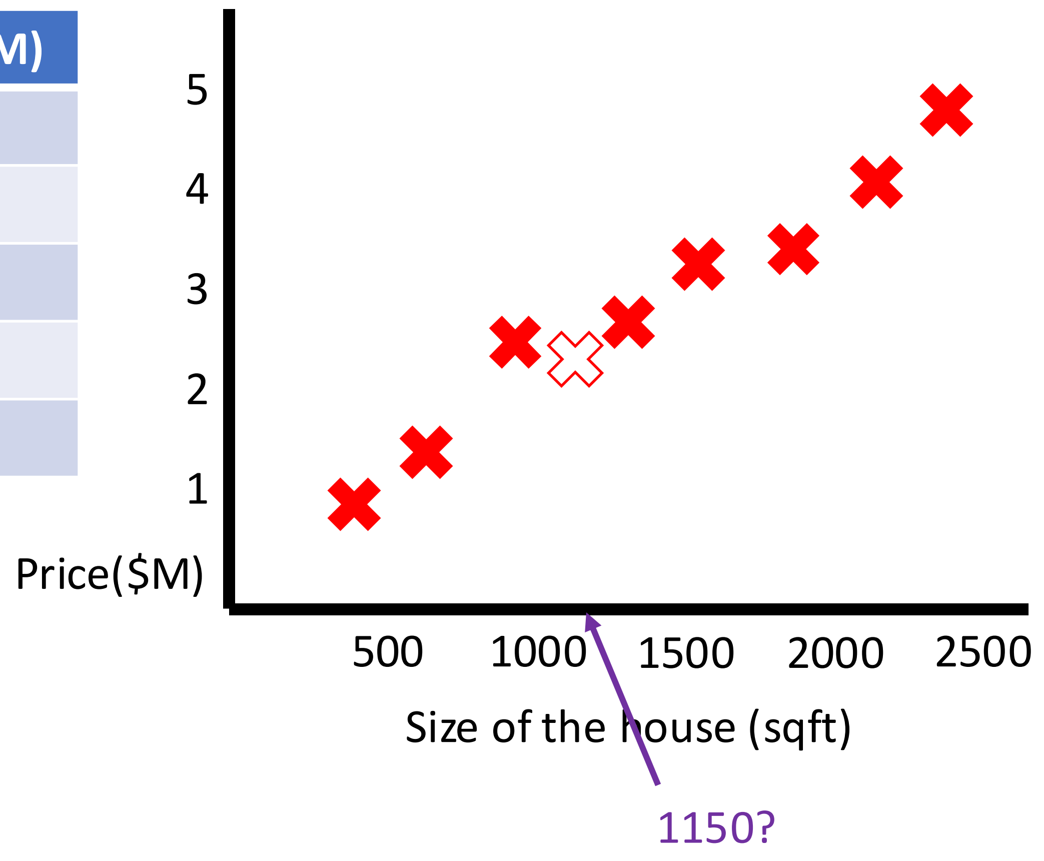
- Linear Regression
    - Data
    - Model
    - Loss
- Learning via Normal Equations
- Learning via Gradient Descent
    - Gradient Descent Algorithm
    - Variants: Mini-batch, stochastic
    - Problems and Solutions

Outline

- **Linear Regression**
  - Data
  - Model
  - Loss
- Learning via Normal Equations
- Learning via Gradient Descent
  - Gradient Descent Algorithm
  - Variants: Mini-batch, stochastic
  - Problems and Solutions

Example: Housing Price Prediction

Size (sqft)	Price ($M)
400	0.9
750	1
950	2.5
1200	2.8
…	…



Price of a house with 1150 sqft?

Data

Suppose:

- We are given N data points.
- Each data point consists of features and a target variable.
- The features are described by a vector of real numbers in dimension d.
- The target is also a real number.

Data – Math

Suppose

$$D = \{(x^{(1)}, y^{(1)}), (x^{(2)},\ y^{(2)}), \dots, (x^{(N)},\ y^{(N)})\},$$

where for all $i \in \{1, \dots, N\}$

$$\text{Features: } x^{(i)} \in \mathbb{R}^{d}$$

$$\text{Targets: } y^{(i)} \in \mathbb{R}$$

Task

Suppose we are given another data point $x \in \mathbb{R}^d$ and no target. Based on the dataset, find a function that predicts the target $y \in \mathbb{R}$ for that x.

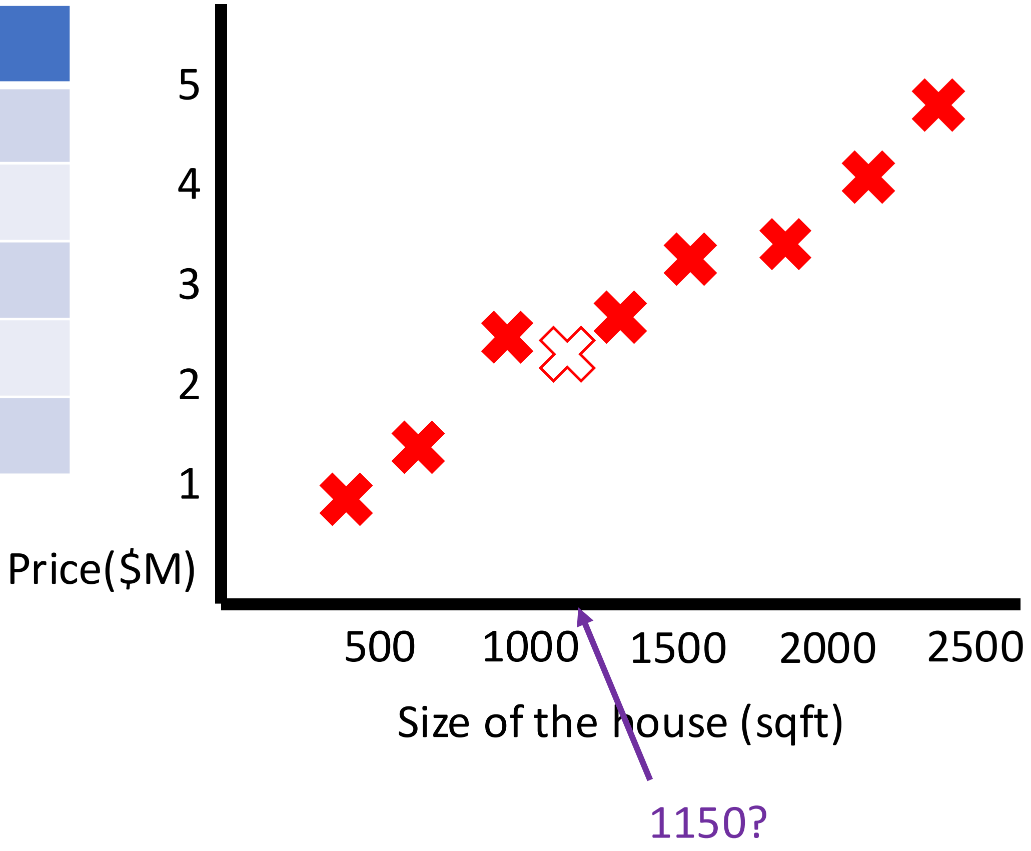
This task is called regression.

History: the word comes from “to regress”, as in “going back”, reverting to the mean, in the context of heredity (biology).

Example: Housing Price Prediction

Dataset D

i	$x^{(i)}$	$y^{(i)}$
1	400	0.9
2	750	1
3	950	2.5
4	1200	2.8
…	…	…



x = a house with 1150 sqft

Price of x?

What class of functions should we use?

Linear Model

By observing the data, from experience, or as a first guess, we may suppose that the hypothesis class is the set of linear functions.

What are linear functions that map as follows?

- **From** d-dimensional vectors of real numbers
- **To** 1-dimensional real numbers (scalars)

Background: Vectors and Dot Product

- Vectors: Let $w_1, w_2, \dots, w_d$ be real numbers.

  - Column vector $w = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_d \end{bmatrix}$ - Row vector $w^T = [w_1 \quad w_2 \quad \cdots \quad w_d]$

- Dot product: Let u, v be two vectors.

$$u^T v = [u_1 \quad u_2 \quad \cdots \quad u_d] \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_d \end{bmatrix} = \sum_{j=1}^{d} u_j v_j$$

Linear Model

Given an input vector x of dimension d, the hypothesis class of linear models is defined as the set of functions:

$$h_w(x) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

where $w_0, \ldots, w_d$ are **parameters/weights** and $x_0 = 1$ is a dummy variable.

We shorthand this function by using the dot product:

$$h_w(x) = w^T x$$

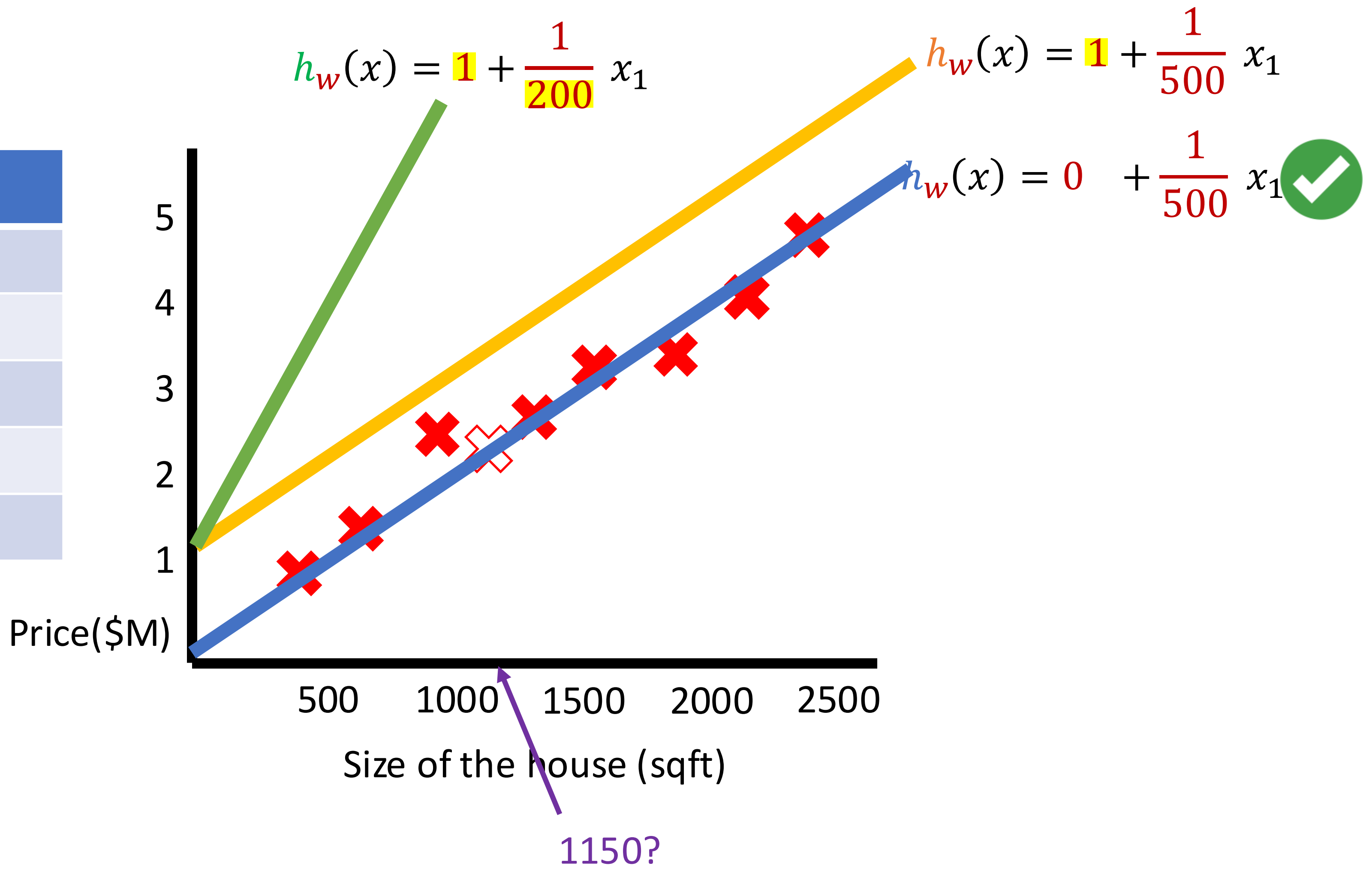
Example: Housing Price Prediction

$$\boxed{h_w(x) = w_0 x_0 + w_1 x_1}$$

$x_0 = 1$ (dummy variable)

Dataset D

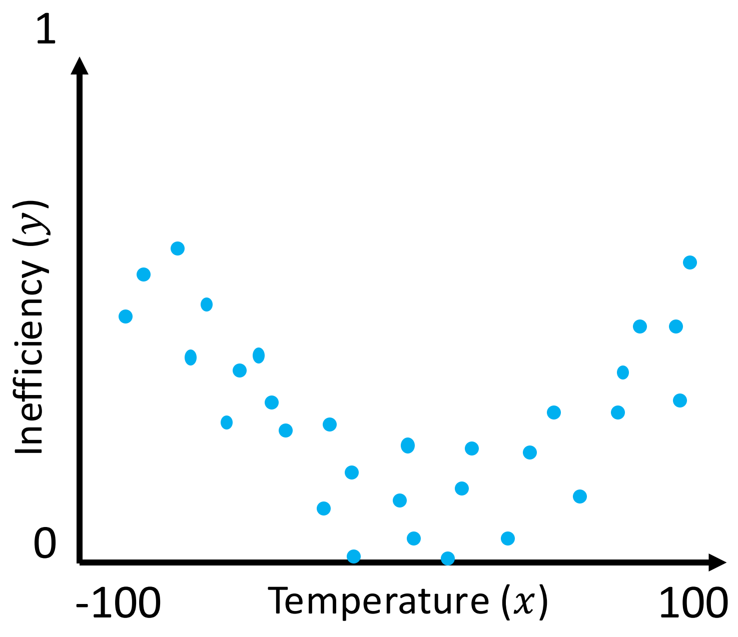
i	$x_1^{(i)}$	$y^{(i)}$
1	400	0.9
2	750	1
3	950	2.5
4	1200	2.8
…	…	…



Example: Engine Inefficiency Prediction

Dataset D

i	$x_1^{(i)}$	$y^{(i)}$
1	-100	1
2	-50	0.25
3	0	0.05
4	50	0.27
5	100	0.98
…	…	…



-100	-50	0	50	100

$$\boxed{h_w(x) = w_0 x_0 + w_1 x_1}$$

$x_0 = 1$ (dummy variable)

Feature Transformations

Feature transformations are techniques used to modify the original features of a dataset to make them more suitable for modeling.

- **Feature Engineering**: create new features based on existing features.
- **Feature Scaling**: scale features to be within a specific range.
- **Feature Encoding**: encoding features from one type to another.

Feature Engineering

Creating new features based on existing features.

- **Polynomial Features**: create new features $z = x^k$ where k is the polynomial degree.

  When used in conjunction with linear regression, this is called polynomial regression.
- **Log Features**: create new features $z = \log(x)$

  This transformation is useful to handle skewed data or to linearize exponential trends.
- **Exp Features**: create new features $z = \mathrm{e}^x$

  This transformation is useful to model exponential growth or decay patterns.
- …

Feature Scaling

Scaling features to be within a specific range.

Min-max scaling

$$z_i = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$$

$\min(x_i)$ and $\max(x_i)$ denote the minimum and the maximum value of a feature x_i in dataset D

Scales the features to be within [0,1].

It is also common to scale the features to be within [-1,1]

Standardization

$$z_i = \frac{x_i - \mu_i}{\sigma_i}$$

μ_i and σ_i is the mean and the standard deviation of feature x_i in dataset D

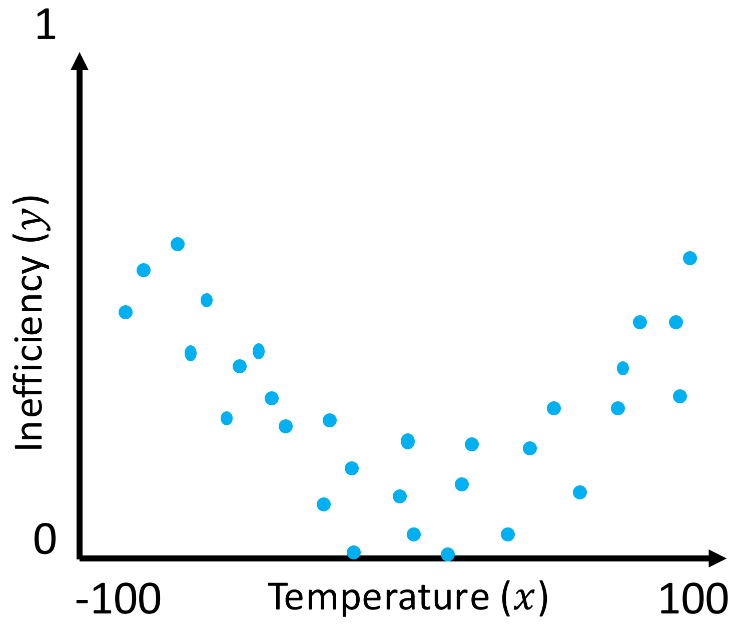
This method transforms features to have a mean of 0 and a standard deviation of 1

Robust Scaling, …

Example: Engine Inefficiency Prediction

Dataset D

i	$x_1^{(i)}$	$y^{(i)}$
1	-100	1
2	-50	0.25
3	0	0.05
4	50	0.27
5	100	0.98
…	…	…



x_1	-100	-50	0	50	100

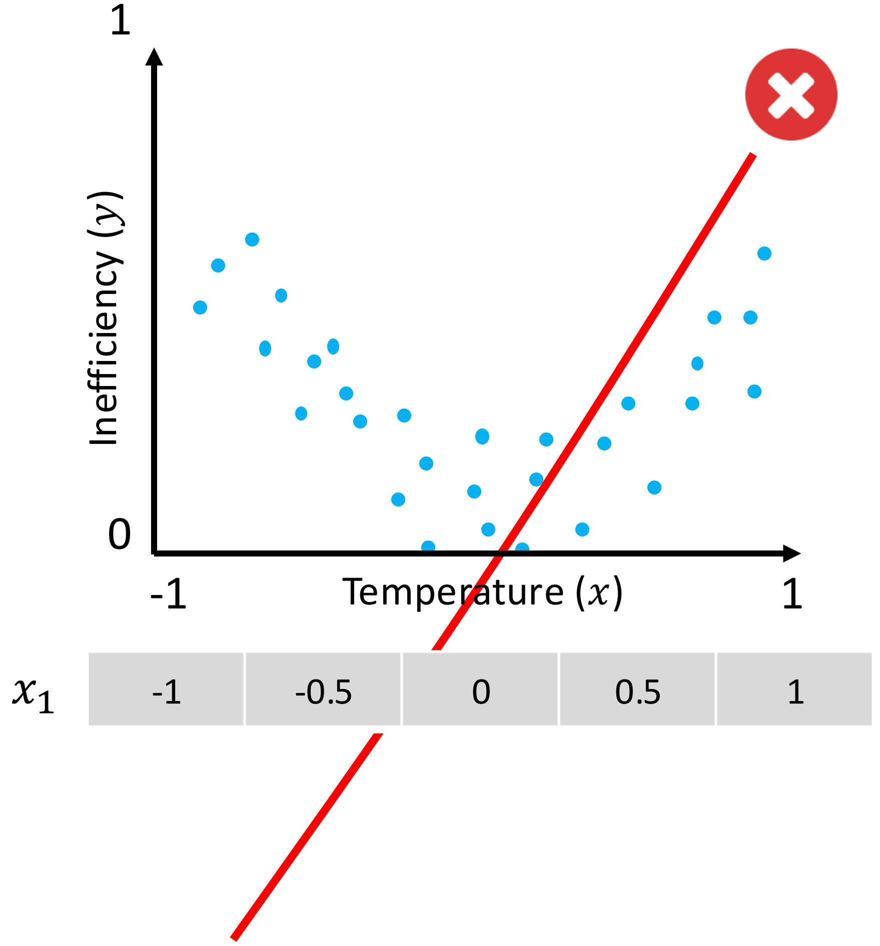
$$\boxed{h_w(x) = w_0 x_0 + w_1 x_1}$$

$x_0 = 1$ (dummy variable)

Example: Engine Inefficiency Prediction

Dataset D

i	$x_1^{(i)}$	$y^{(i)}$
1	-100	1
2	-50	0.25
3	0	0.05
4	50	0.27
5	100	0.98
…	…	…



$$\boxed{h_w(x) = w_0 x_0 + w_1 x_1}$$

$x_0 = 1$ (dummy variable)

Scale features to be between -1 and 1

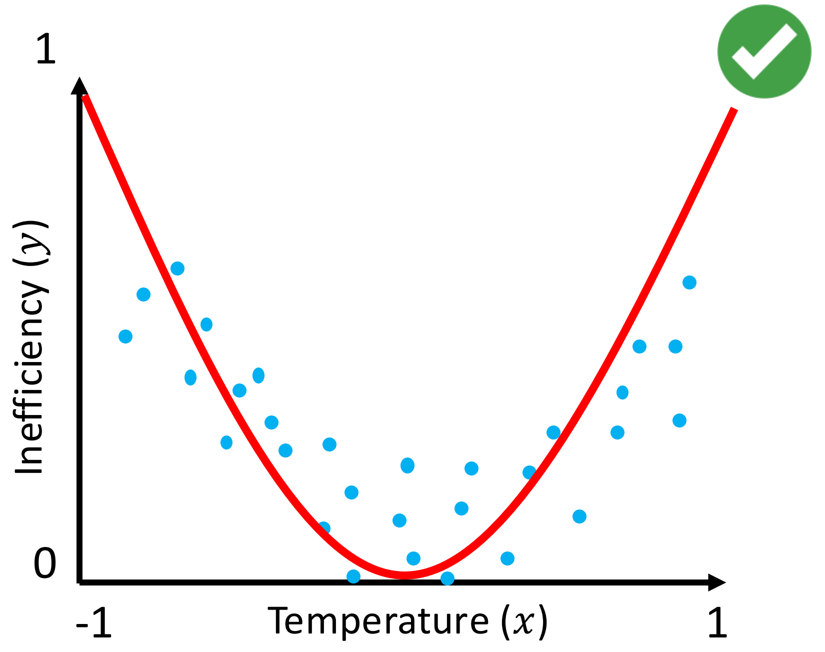
Linear model:

$$h_w(x) = 0 + 1x_1$$

Example: Engine Inefficiency Prediction

Dataset D

i	$x_1^{(i)}$	$y^{(i)}$
1	-100	1
2	-50	0.25
3	0	0.05
4	50	0.27
5	100	0.98
…	…	…



x_1	-1	-0.5	0	0.5	1
x_2	1	0.25	0	0.25	1

$$\boxed{h_w(x) = w_0 x_0 + w_1 x_1}$$

$x_0 = 1$ (dummy variable)

Scale features to be between -1 and 1

Linear model:

$$h_w(x) = 0 + 1x_1$$ ✗

Engineer feature $x_2 = x_1^2$

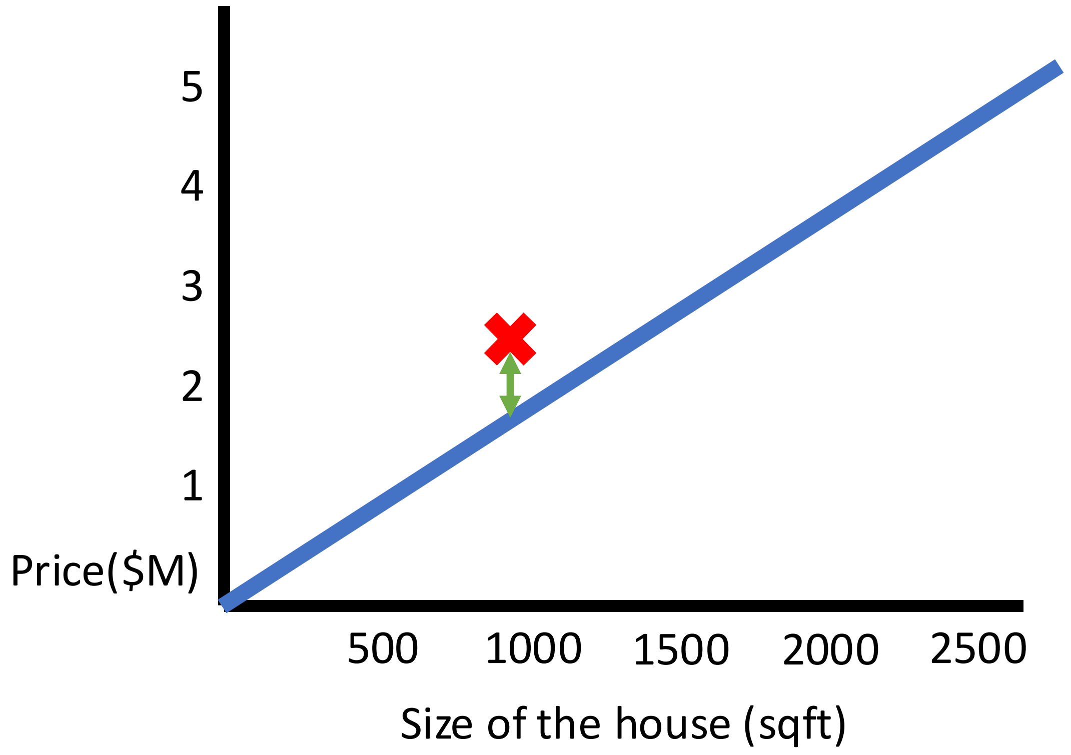
Linear model:

$$h_w(z) = 0 + 0x_1 + 1x_2$$ ✓

(Polynomial Regression)

Linear Regression: Measuring Fit

Given a hypothesis, we want to measure how good it fits the data.



We can use squared error

$$(h_w(x^{(i)}) - y^{(i)})^2$$

Linear Regression: Measuring Fit

For N examples $\{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$, define the **mean squared error (MSE)**:

$$J_{MSE}(w) = \frac{1}{2N} \sum_{i=1}^{N} \left(h_w(x^{(i)}) - y^{(i)}\right)^2$$

- Also called the **loss function**. Notice that it is a function of w.
- The factor ½ is only for mathematical convenience, i.e., because we take derivatives later.
- We want to find w that minimize this loss function!

Outline

- Linear Regression
  - Data
  - Model
  - Loss
- **Learning via Normal Equations**
- Learning via Gradient Descent
  - Gradient Descent Algorithm
  - Variants: Mini-batch, stochastic
  - Problems and Solutions

Background: Minimizing a Function

Minimizing a one-dimensional function of variable w, where x and y are scalars.

- Function $J(w) = \frac{1}{2}(wx - y)^2$
- Take derivative $J'(w) = (wx - y)x$
- First-order condition $J'(w) = 0$

Hence

$$(wx - y)x = 0$$
$$w = \frac{y}{x}$$

Background: Partial Derivative

- Suppose we are given a scalar function $f(w)$ with d-dimensional input.

- Partial derivative $\dfrac{\partial f(w)}{\partial w_i}$

Example: $f(w) = w_0^2 + w_1^2 \Longrightarrow \frac{\partial f(w)}{\partial w_1} = 2w_1$

Linear Regression

- Let’s take the partial derivative of the linear model

- Let’s take partial derivative for each term in MSE

- Hence,

Linear Regression

- Let's take the partial derivative of the linear model

$$\frac{\partial}{\partial w_j} h_w\left(x^{(i)}\right) = \frac{\partial}{\partial w_j}\left(w^T x^{(i)}\right) = x_j^{(i)}$$

- Let's take partial derivative for each term in MSE

$$\frac{\partial}{\partial w_j}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)^2 = 2\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

- Hence,

$$\begin{aligned}\frac{\partial J_{MSE}(w)}{\partial w_j} &= \frac{1}{2N}\frac{\partial}{\partial w_j}\sum_{i=1}^{N}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)^2 \\ &= \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial}{\partial w_j}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\end{aligned}$$

Minimum: $\frac{\partial J_{MSE}(w)}{\partial w_j} = 0$

Background: Matrices

Let $x_{11}, x_{12}, \dots, x_{1d}, x_{21}, \dots, x_{2d}, \dots, x_{Nd}$ be real numbers.

Matrix

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \cdots & \cdots & \cdots \\ x_{N1} & \cdots & x_{Nd} \end{bmatrix} \in \mathbb{R}^{N \times d}$$

Transpose of Matrix

$$X^T = \begin{bmatrix} x_{11} & \cdots & x_{N1} \\ \cdots & \cdots & \cdots \\ x_{1d} & \cdots & x_{Nd} \end{bmatrix} \in \mathbb{R}^{d \times N}$$

Background: Matrices

Matrix-vector multiplication: Let X be a matrix and v be a vector.

$$Xv = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \cdots & \cdots & \cdots \\ x_{N1} & \cdots & x_{Nd} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \cdots \\ v_d \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{d} x_{1j} v_j \\ \sum_{j=1}^{d} x_{2j} v_j \\ \cdots \\ \sum_{j=1}^{d} x_{Nj} v_j \end{bmatrix} \in \mathbb{R}^N$$

Matrix multiplication: Let X, A be two matrices (with suitable dimension)

$$XA = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \cdots & \cdots & \cdots \\ x_{N1} & \cdots & x_{Nd} \end{bmatrix} \begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \cdots & \cdots & \cdots \\ a_{d1} & \cdots & a_{dN} \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{d} x_{1j} a_{j1} & \cdots & \sum_{j=1}^{d} x_{1j} a_{jN} \\ \cdots & \cdots & \cdots \\ \sum_{j=1}^{d} x_{Nj} a_{j1} & \cdots & \sum_{j=1}^{d} x_{Nj} a_{jN} \end{bmatrix} \in \mathbb{R}^{N \times N}$$

Normal Equation

Goal: find w that minimizes J_{MSE}

$$\frac{\partial J_{MSE}(w)}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}\left(w^T x^{(i)} - y^{(i)}\right)x_j^{(i)} = 0 \quad \xrightarrow{} \quad X^T(Xw - Y) = 0$$

Express with **vectors and matrices**

$$X = \begin{bmatrix} 1 & x_1^{(1)} & & x_d^{(1)} \\ 1 & x_1^{(2)} & \dots & x_d^{(2)} \\ 1 & \vdots & & \vdots \\ 1 & x_1^{(N)} & & x_d^{(N)} \end{bmatrix} \qquad w = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_d \end{bmatrix} \qquad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(N)} \end{bmatrix}$$

$$\xrightarrow{} \quad \boxed{w = (X^T X)^{-1} X^T Y}$$

Assume **invertible**

The Problems with Normal Equation

For linear regression (linear model + MSE), the normal equation solves the problem of finding the best parameters (assuming invertibility).

However:

- The cost of normal equation is d^3 (for inverting matrix) .
- It will not work for non-linear models (which we will introduce in the future lectures).

Is there an alternative to finding a minimum of a function?

Outline

- Linear Regression
    - Data
    - Model
    - Loss
- Learning via Normal Equations
- **Learning via Gradient Descent**
    - Gradient Descent Algorithm
    - Variants: Mini-batch, stochastic
    - Problems and Solutions

Background: Gradient

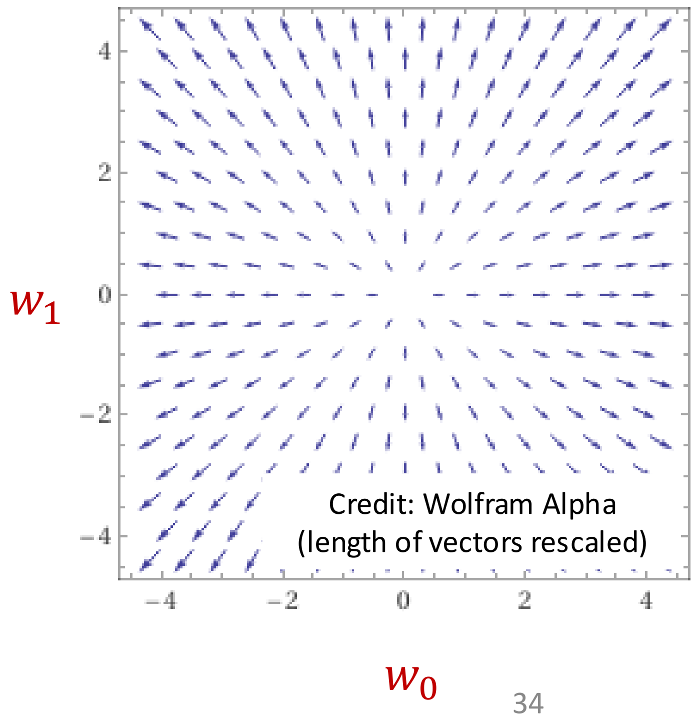
- Suppose we are given a scalar function $f(w)$ with $d + 1$-dimensional input.

- Partial derivative $\frac{\partial f(w)}{\partial w_i}$

  Example: $f(w) = w_0^2 + w_1^2 \Longrightarrow \frac{\partial f(w)}{\partial w_1} = 2w_1$

- Gradient $\begin{bmatrix} \frac{\partial f(w)}{\partial w_0} \\ \frac{\partial f(w)}{\partial w_1} \\ \cdots \\ \frac{\partial f(w)}{\partial w_d} \end{bmatrix}$ Example: $\begin{bmatrix} 2w_0 \\ 2w_1 \end{bmatrix}$

$f(w) = w_0^2 + w_1^2$



Credit: Wolfram Alpha
(length of vectors rescaled)

Gradient Descent

Remember local search? Hill-climbing?

- Start at some w (e.g., randomly initialized).
- Update w with a step in the opposite direction of the gradient (i.e., towards lower loss)

$$w_j \leftarrow w_j - \underbrace{\gamma}_{\text{Learning Rate}} \frac{\partial J(w_0, w_1, \ldots)}{\partial w_j}.$$

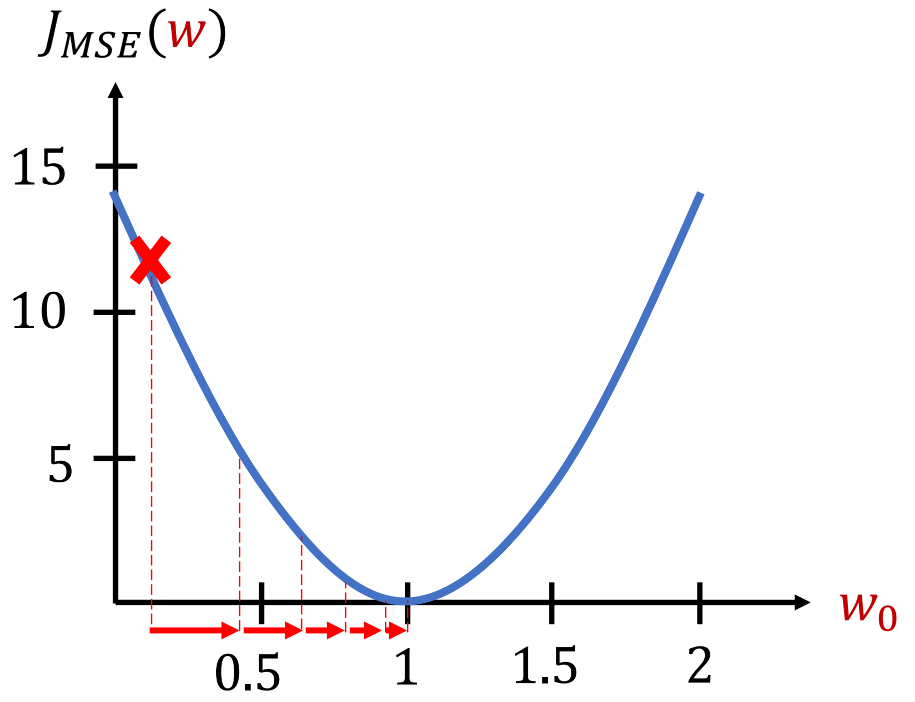
- Learning rate $\gamma > 0$ is a hyperparameter that determines the step size.
- Repeat until termination criterion is satisfied.
  - E.g., change between steps is small, maximum number of steps is reached, etc.

Gradient Descent: 1 Parameter

- Start at some w_0.
- Update w_0 with

$$w_0 \leftarrow w_0 - \underbrace{\gamma}_{\text{Learning Rate}} \boxed{\frac{\partial J(w_0)}{\partial w_0}}$$

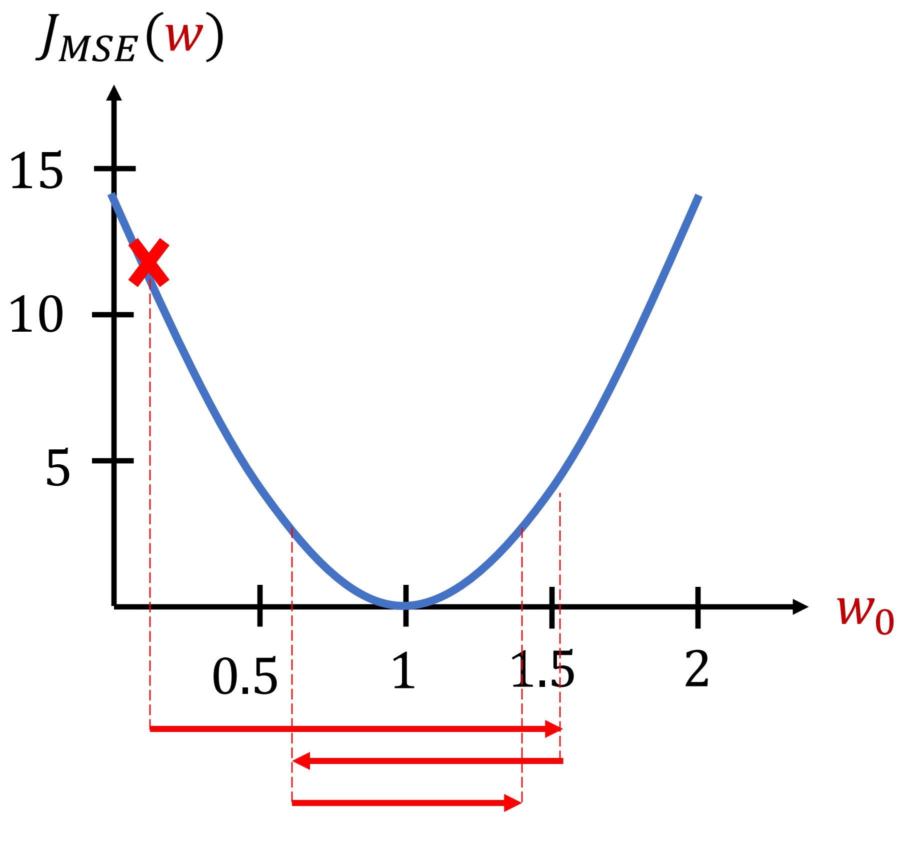
- Repeat until termination criterion is satisfied.



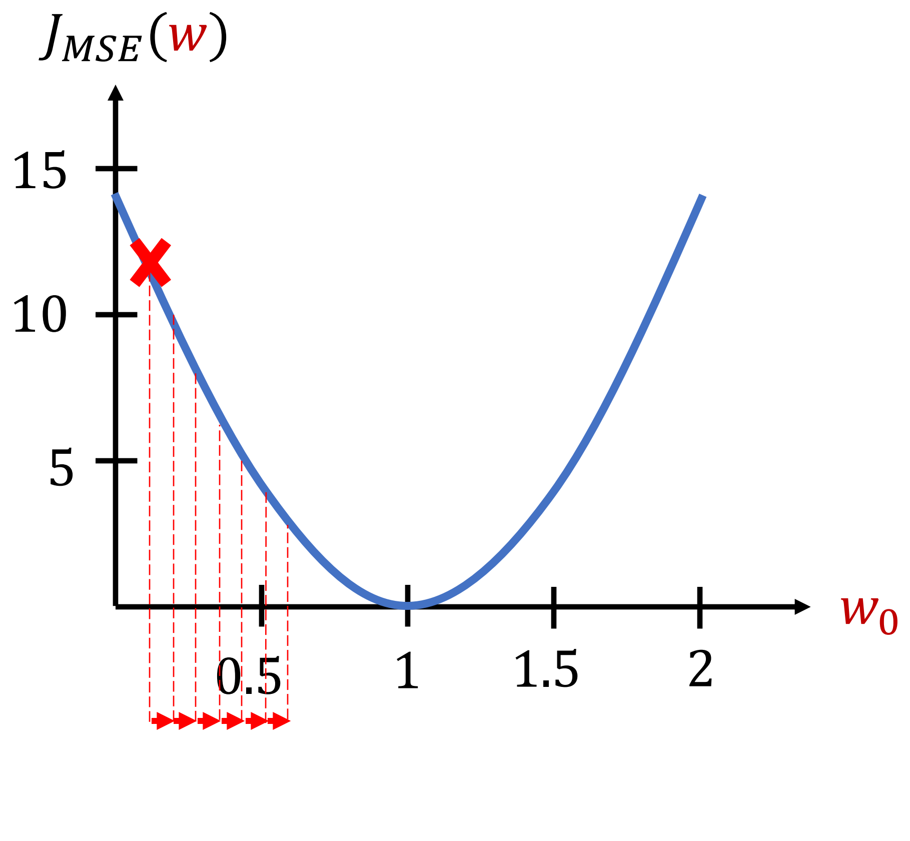
As it gets closer to a minimum,

- The magnitude of the **slope** becomes smaller
- The **step size** become smaller

Gradient Descent: Setting the Learning Rate



γ too large



γ too small

Gradient Descent: 2 Parameters

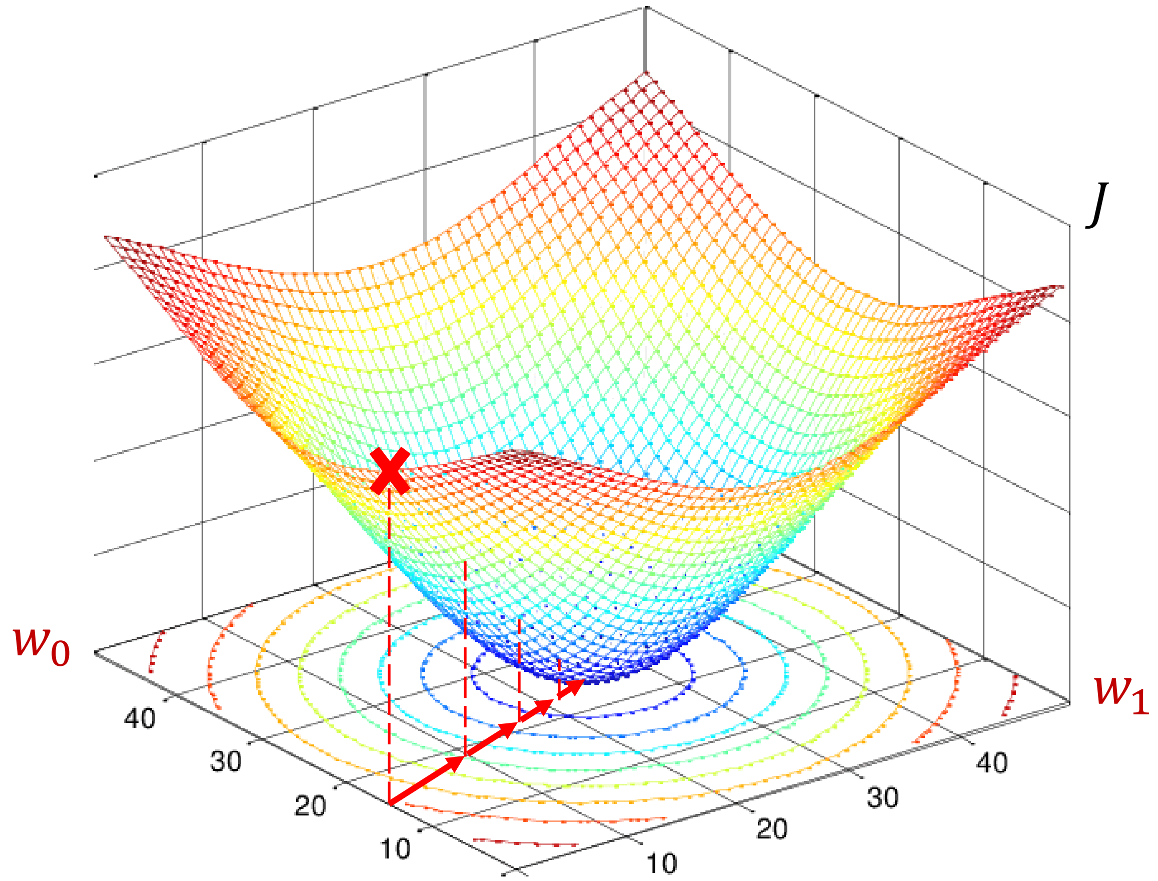
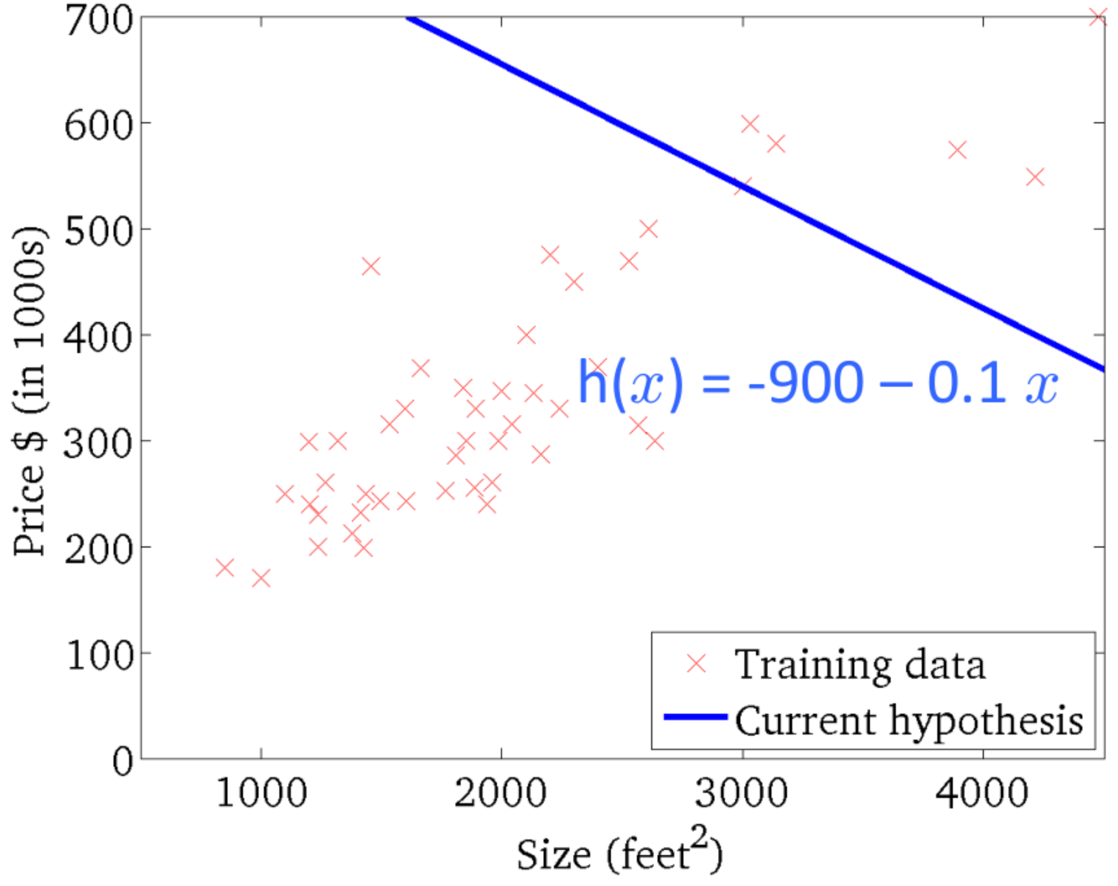


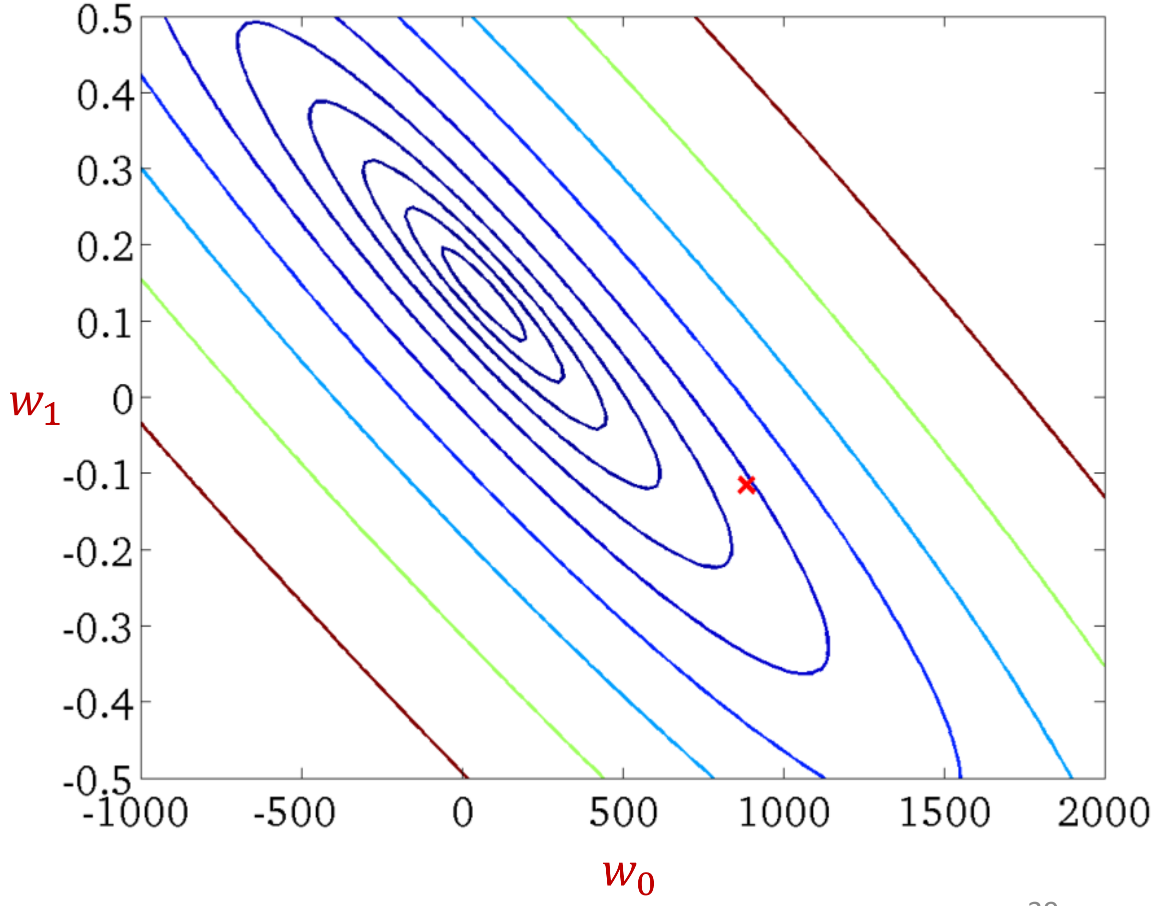
Image Credit: https://www.researchgate.net/figure/The-2D-Caussian-function-of-Example-2-Example-1-Consider-the-following-strongly-convex_fig2_230787652

Linear Regression with Gradient Descent

$h_w(x)$



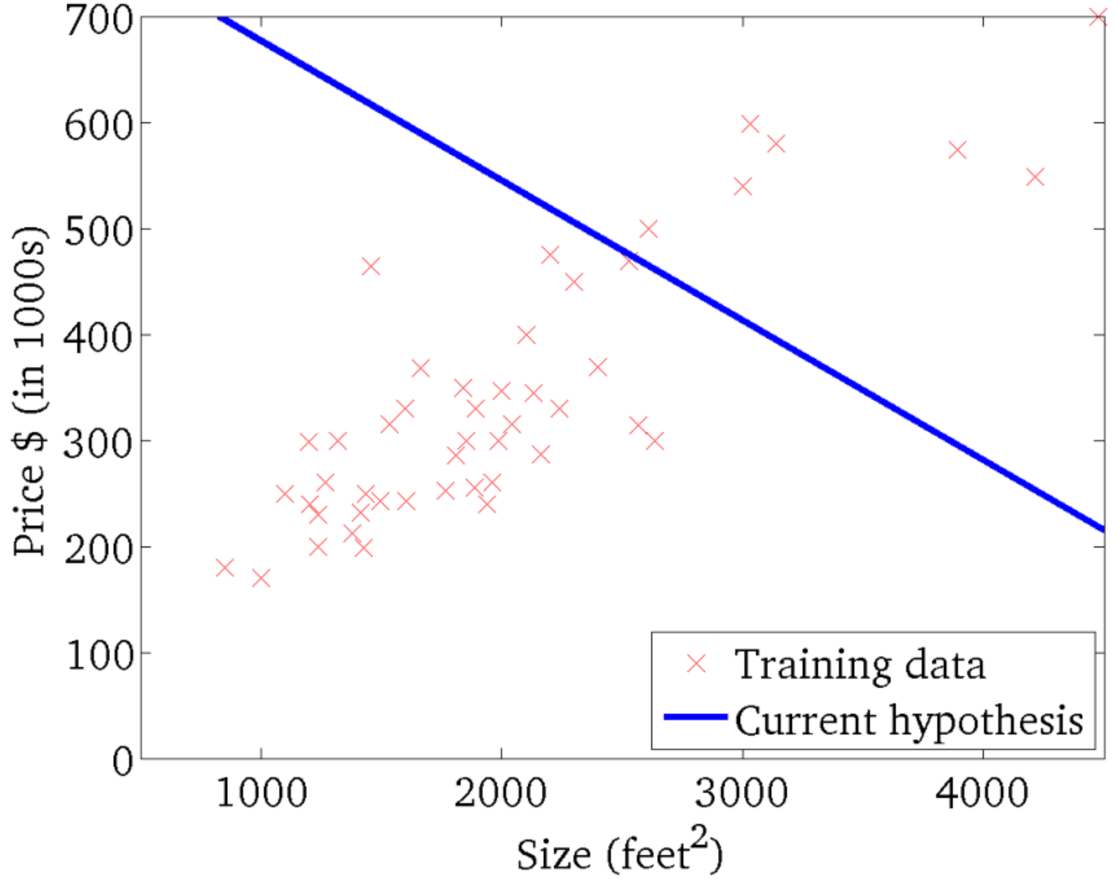
$J_{MSE}(w_0, w_1)$



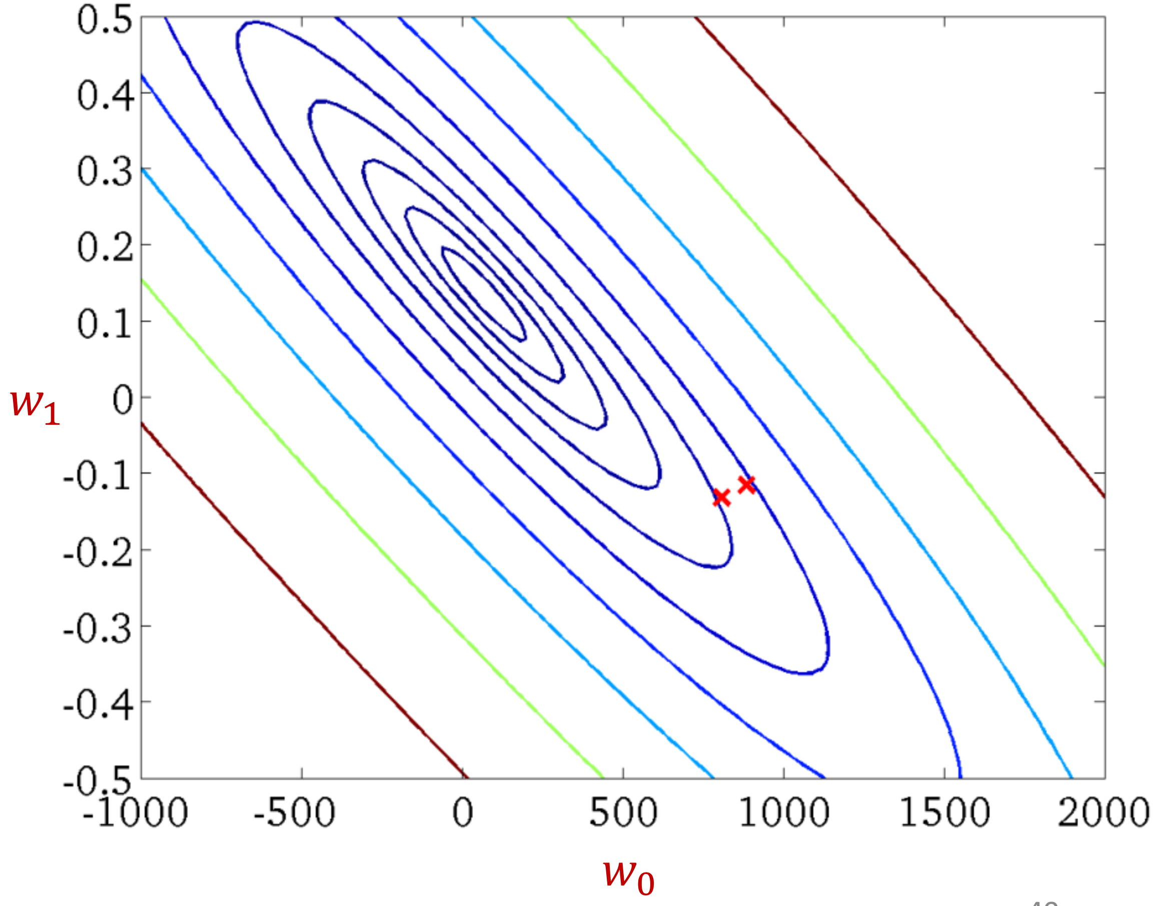
Credit: Andrew Ng

Linear Regression with Gradient Descent

$h_w(x)$



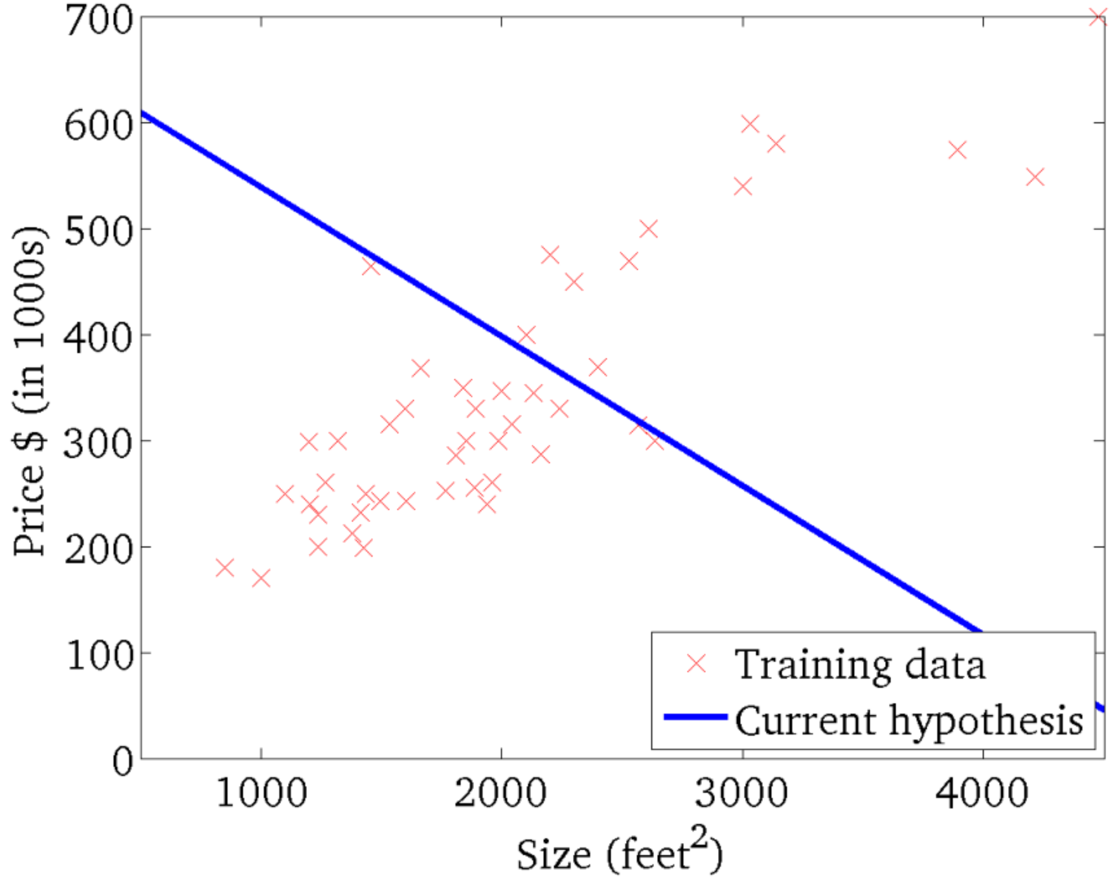
$J_{MSE}(w_0, w_1)$



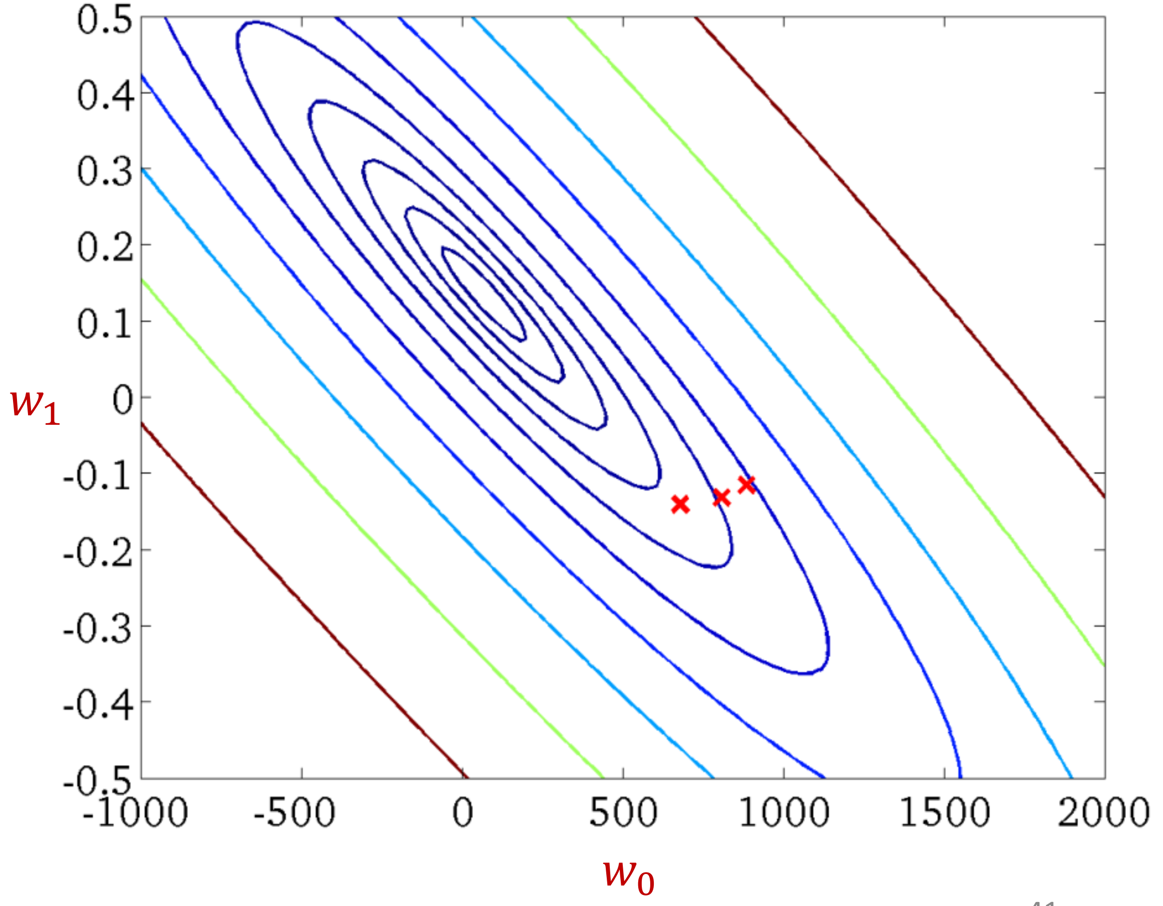
Credit: Andrew Ng

Linear Regression with Gradient Descent

$h_w(x)$



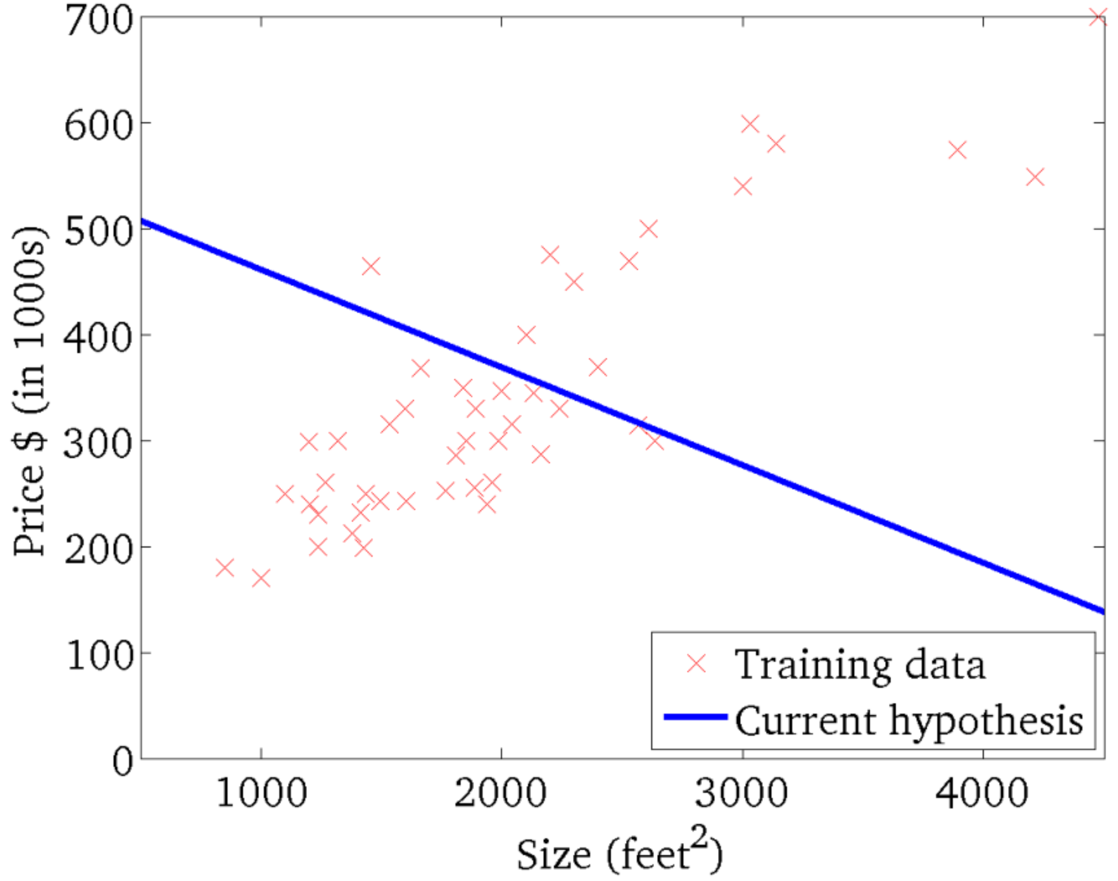
$J_{MSE}(w_0, w_1)$



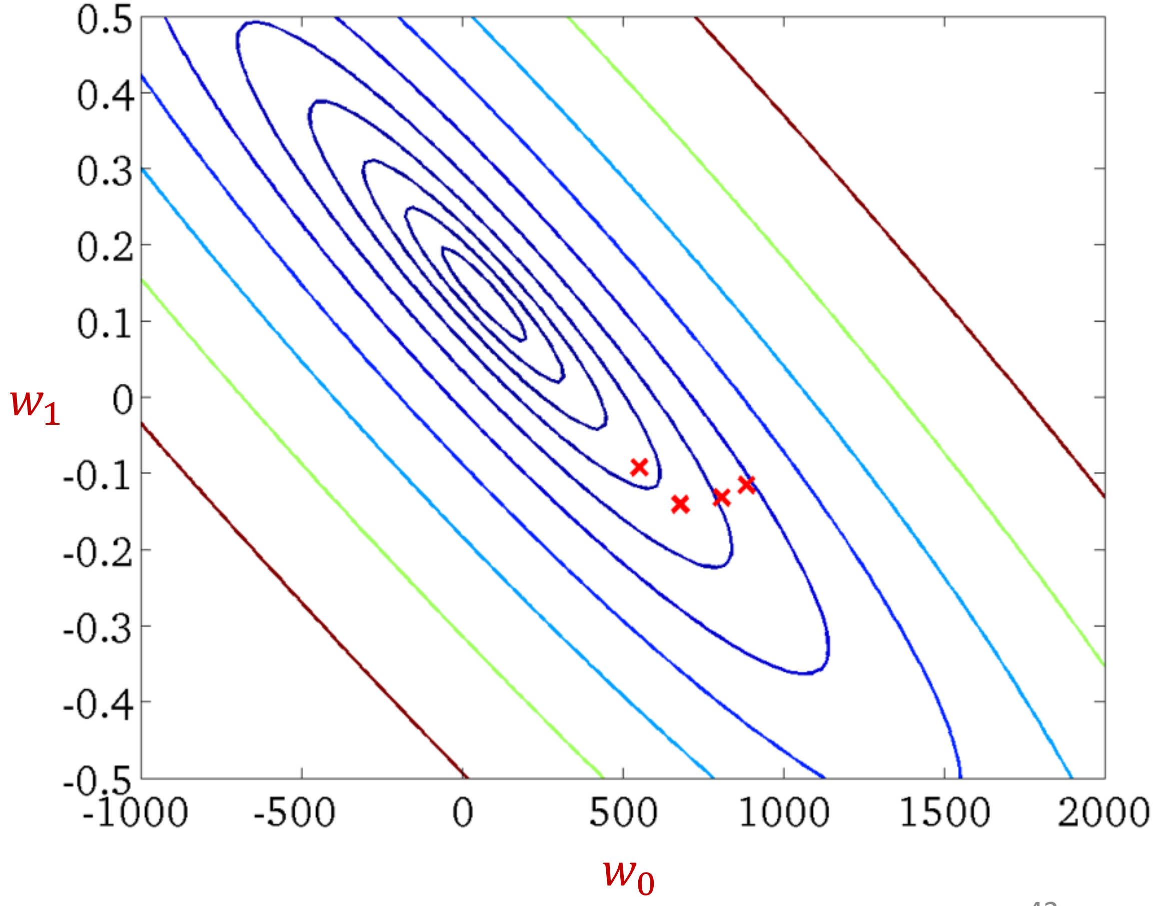
Credit: Andrew Ng

Linear Regression with Gradient Descent

$h_w(x)$



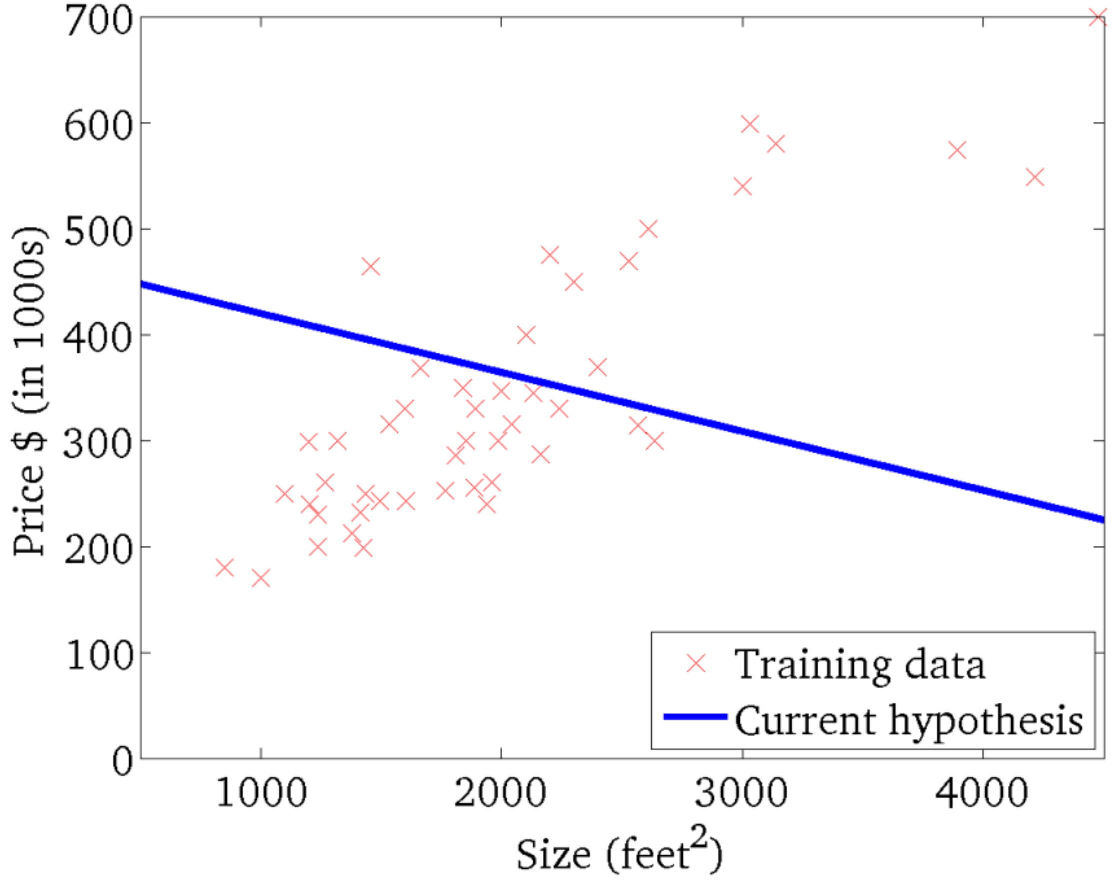
$J_{MSE}(w_0, w_1)$



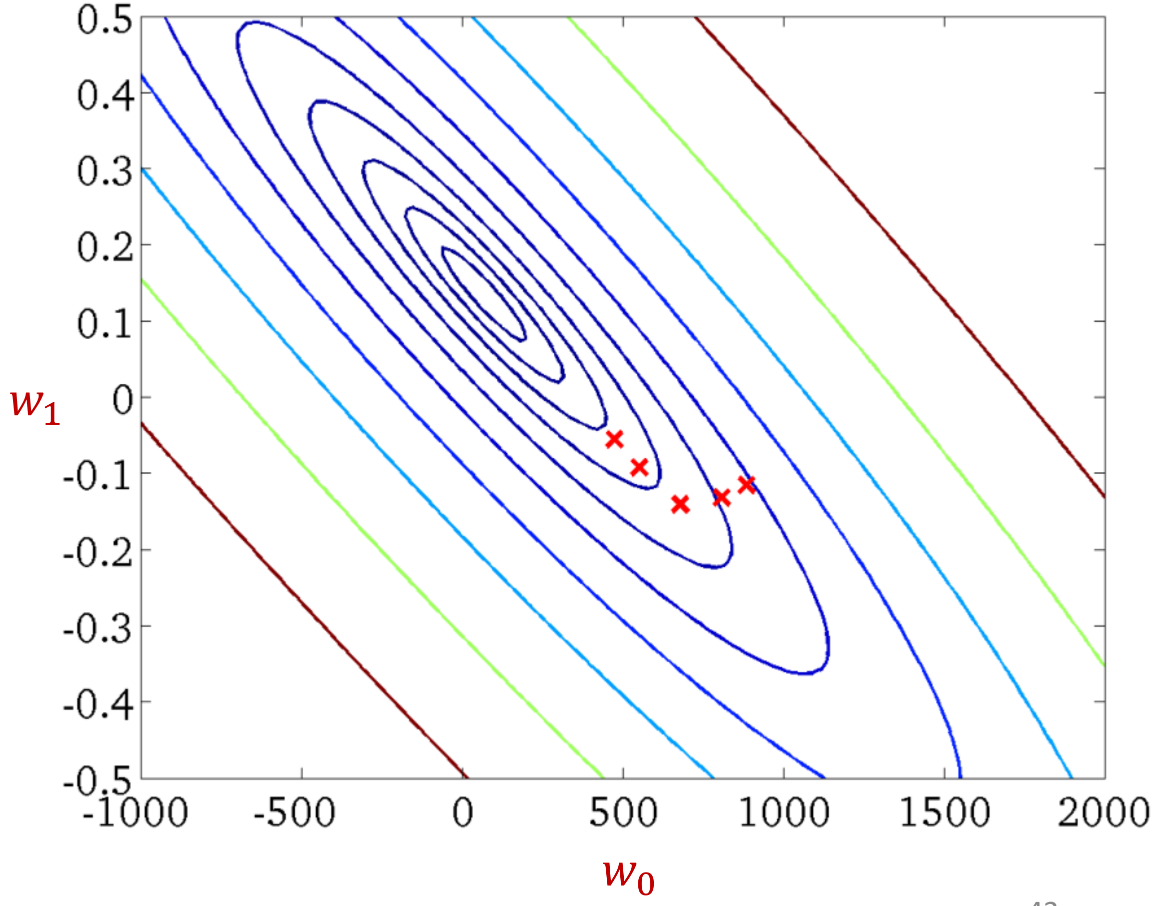
Credit: Andrew Ng

Linear Regression with Gradient Descent

$h_{w}(x)$



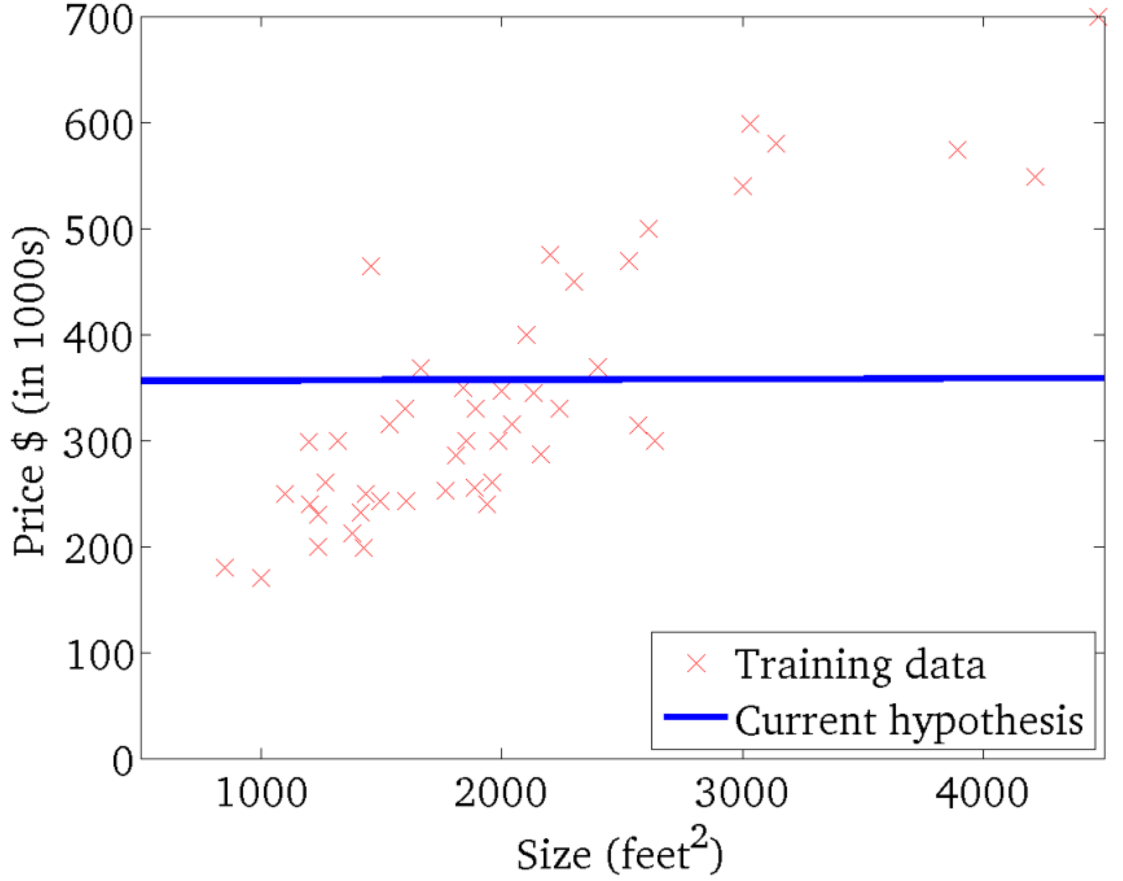
$J_{MSE}(w_0, w_1)$



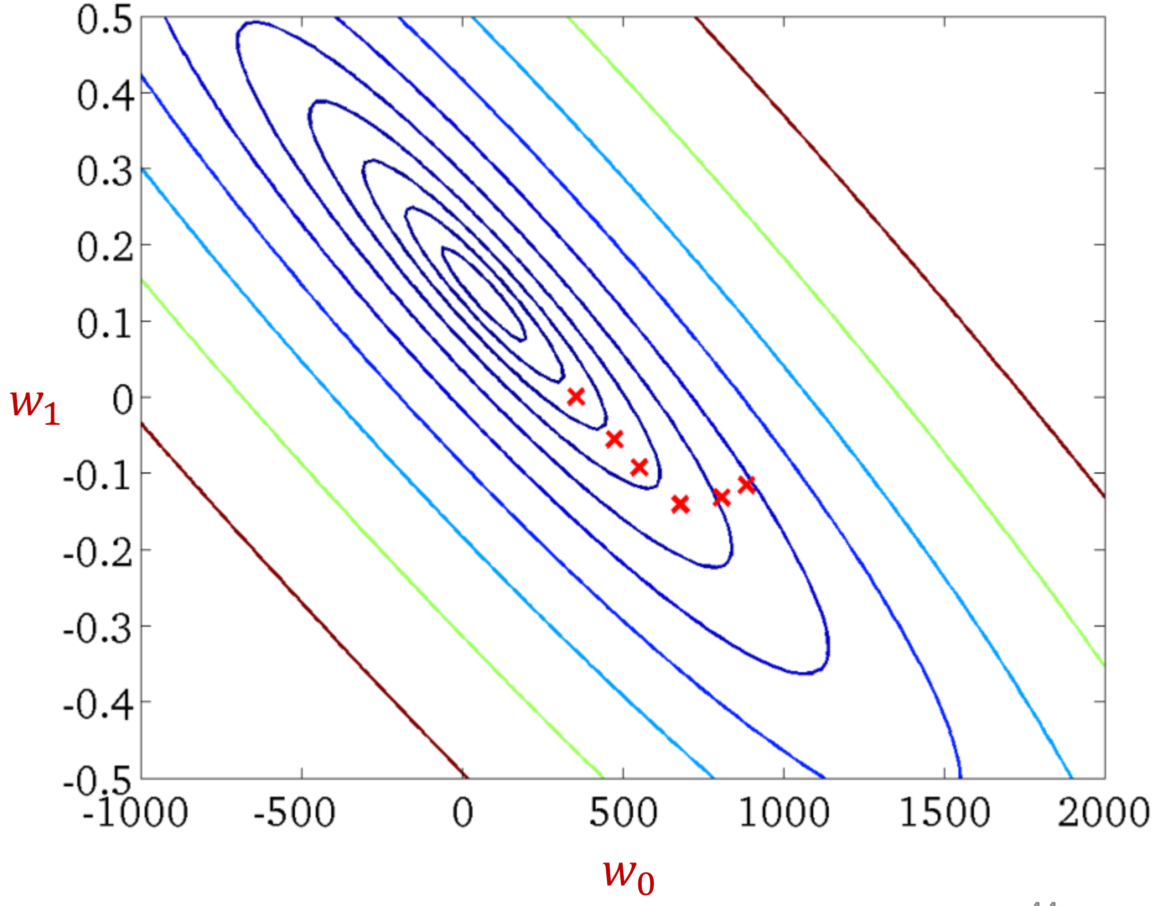
Credit: Andrew Ng

Linear Regression with Gradient Descent

$h_w(x)$



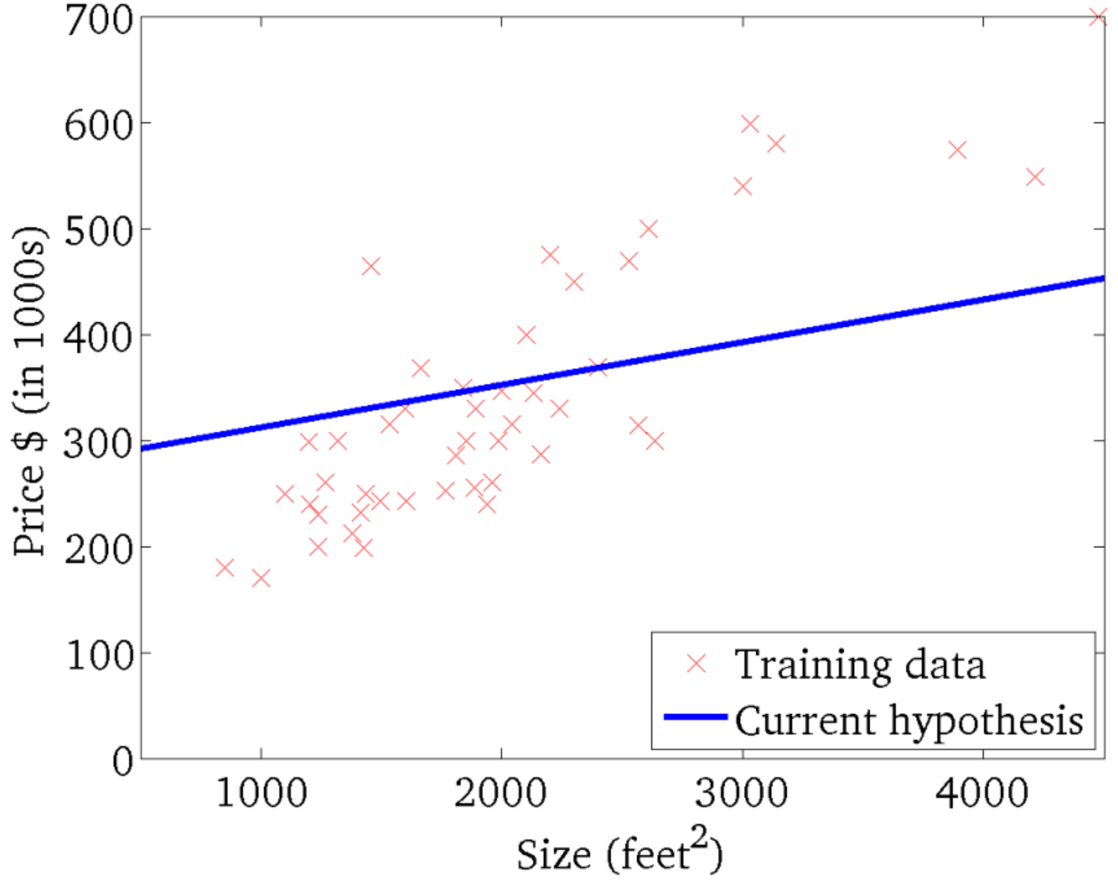
$J_{MSE}(w_0, w_1)$



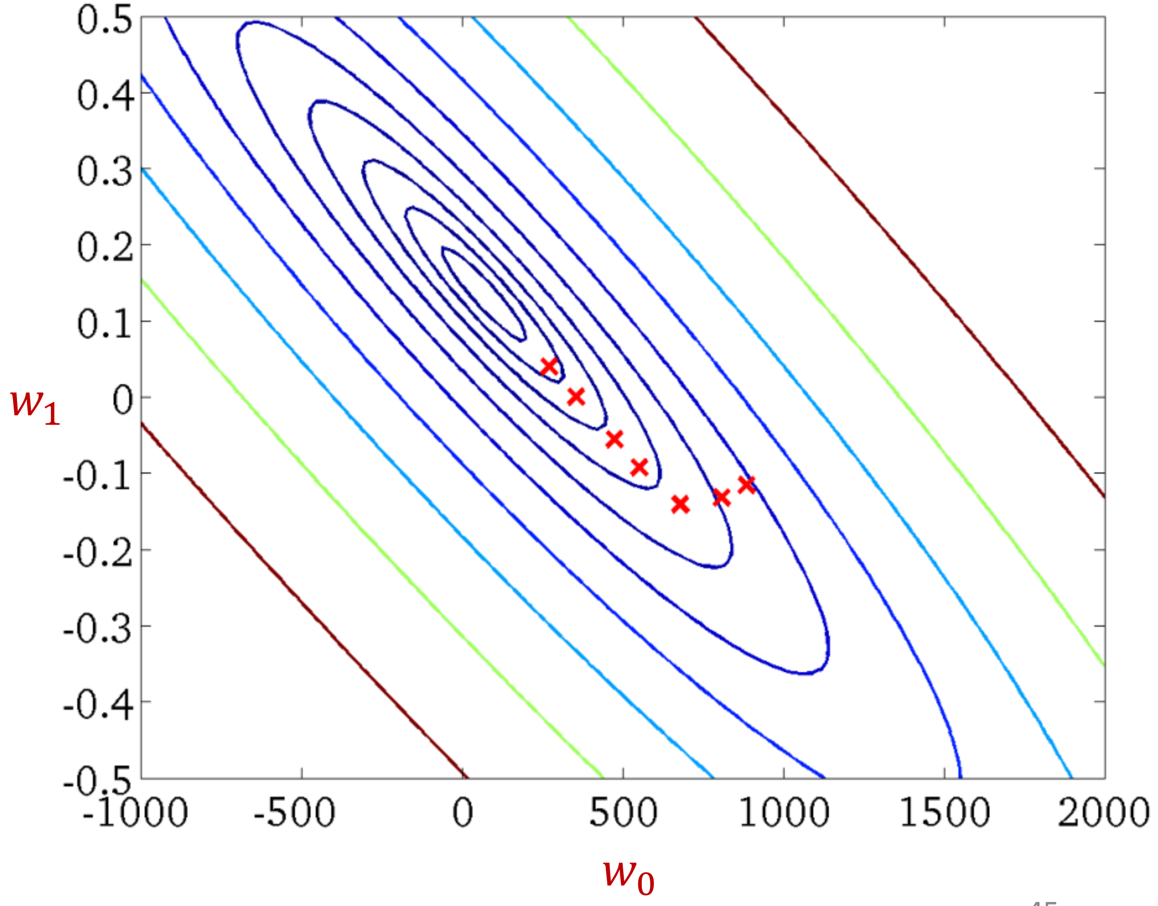
Credit: Andrew Ng

Linear Regression with Gradient Descent

$h_w(x)$



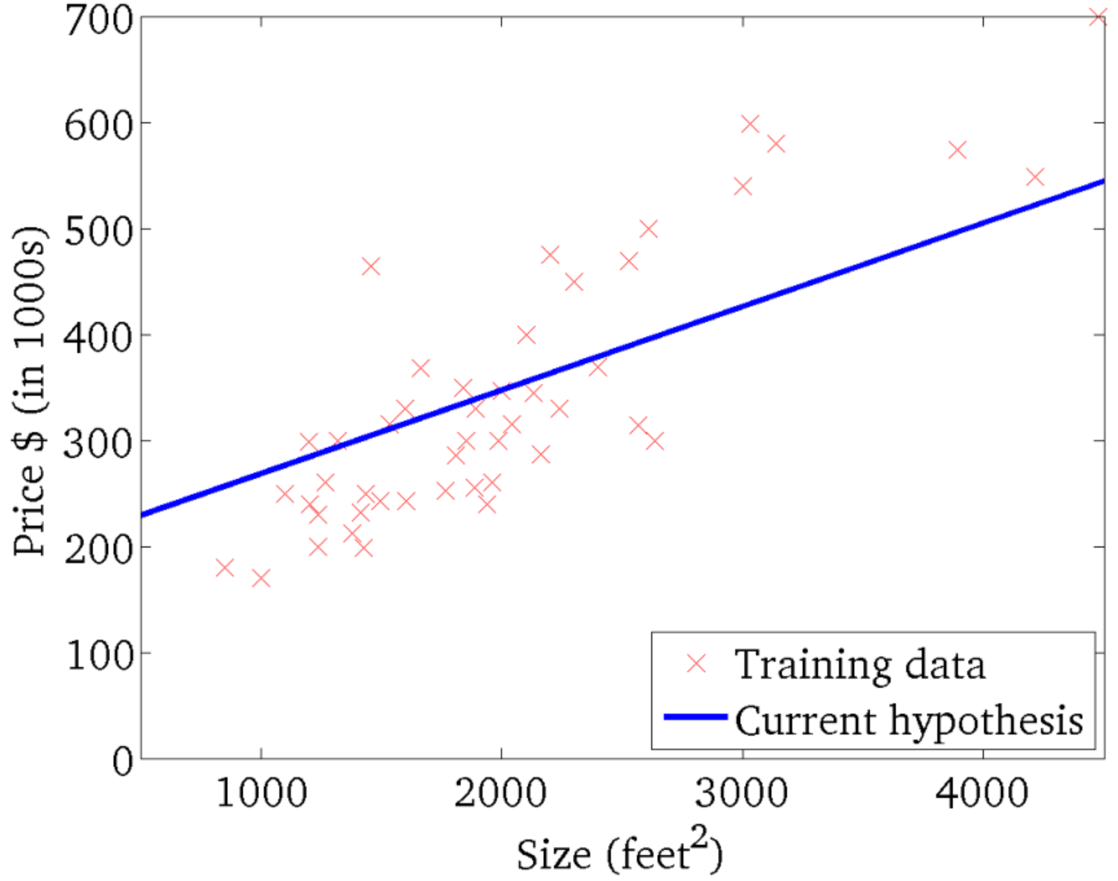
$J_{MSE}(w_0, w_1)$



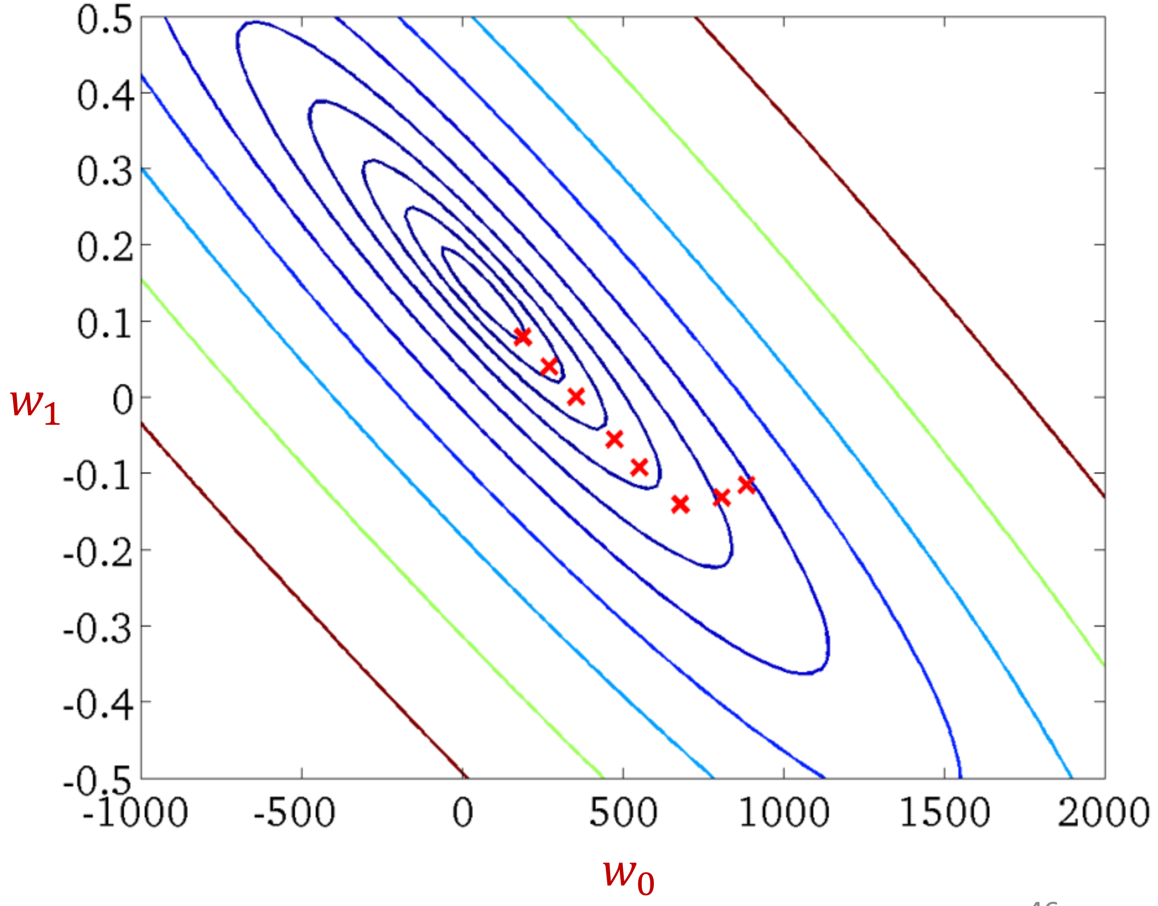
Credit: Andrew Ng

Linear Regression with Gradient Descent

$h_w(x)$



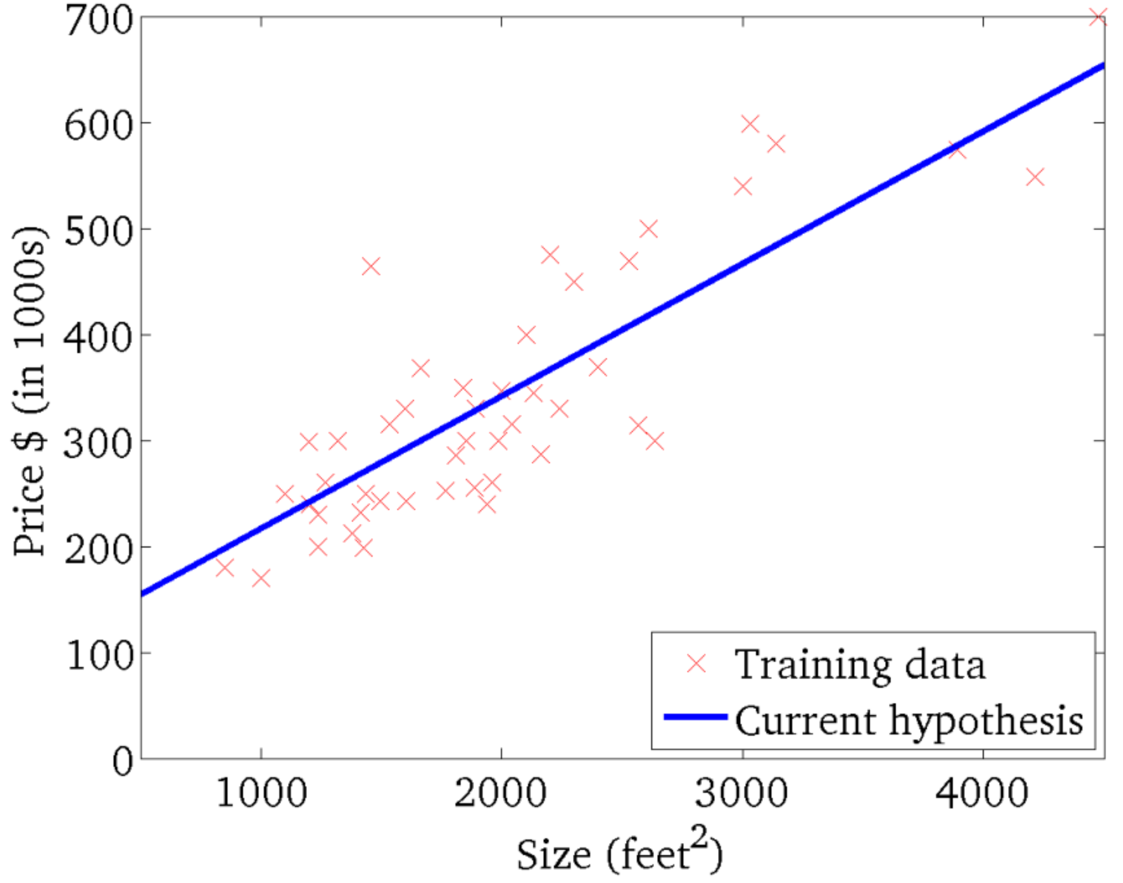
$J_{MSE}(w_0, w_1)$



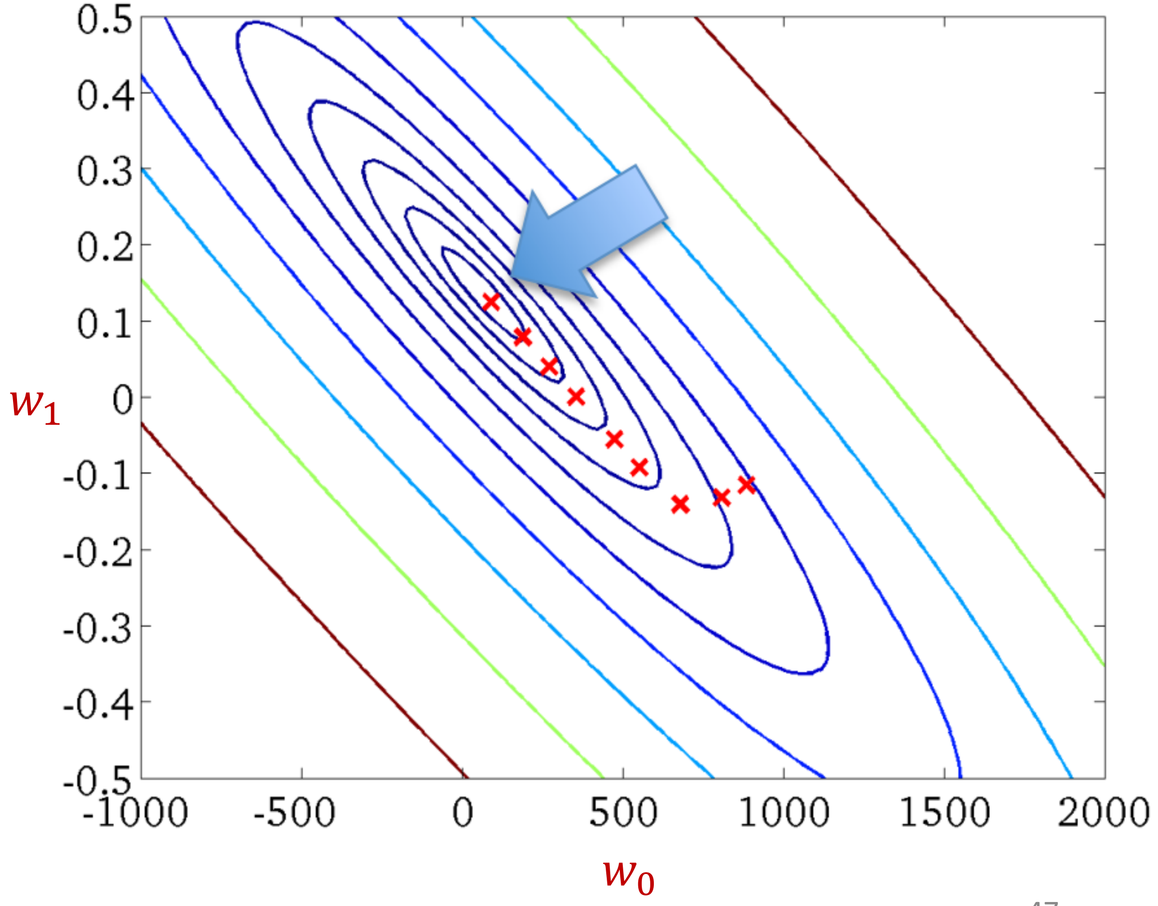
Credit: Andrew Ng

Linear Regression with Gradient Descent

$h_w(x)$



$J_{MSE}(w_0, w_1)$



Credit: Andrew Ng

Gradient Descent: Common Mistake

w_0 changed!

$$w_0 = w_0 - \gamma \frac{\partial J(w_0, w_1)}{\partial w_0}$$
$$w_1 = w_1 - \gamma \frac{\partial J(w_0, w_1)}{\partial w_1}$$

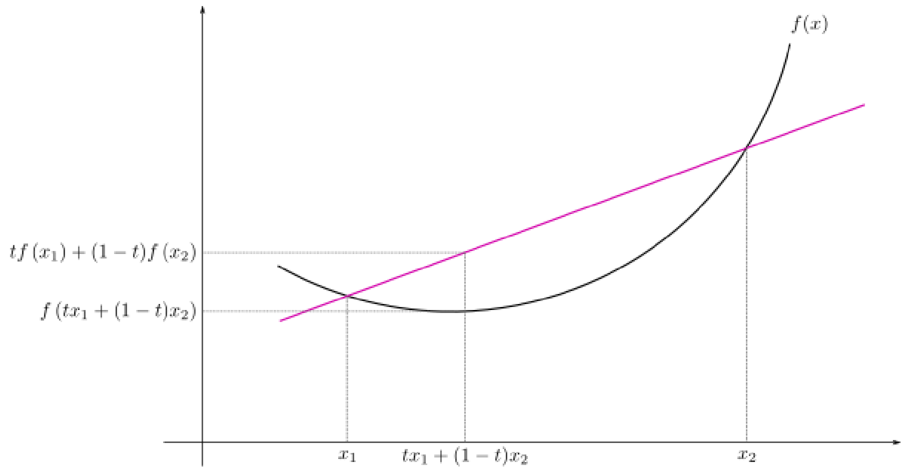
✗

$$a = \frac{\partial J(w_0, w_1)}{\partial w_0}$$
$$b = \frac{\partial J(w_0, w_1)}{\partial w_1}$$
$$w_0 = w_0 - \gamma a$$
$$w_1 = w_1 - \gamma b$$

✓

Background: Convexity

- A real-valued one-dimensional function is called convex if the line segment between any two distinct points on the graph of the function lies above or on the graph between the two points.



- For multi-dimensional function, think of bowl-shaped landscape.

Linear Regression with Gradient Descent

- **Theorem:** A convex function has a single global minimum (informal).
- **Theorem: MSE loss function is convex for linear regression.**

Poll Everywhere

Is the MSE loss function convex for polynomial regression?

a. Yes

b. No

Linear Regression with Gradient Descent

- **Theorem:** A convex function has a single global minimum (informal).
- **Theorem: MSE loss function is convex for linear regression.**

- **MSE loss function is convex for polynomial regression.**
  - After feature transformations, the model still remains a linear model, thus feature transformations do not affect convexity of the MSE (in w) and the number of minima.

Problem: Features of Different Scales

x_1 Size of kitchen counter (m²)	x_2 Size (m²)	y Price ($1K)
0.4	113	560
0.3	102	739
0.7	100	430
1.3	84	698
0.3	112	688
0.5	68	390
0.6	53	250
1.5	122	788
3.0	150	680
1.2	90	828

$$w_j \leftarrow w_j - \gamma \frac{\partial J(w_0, w_1, \dots)}{\partial w_j}.$$

$$\frac{\partial J_{MSE}(w)}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} \left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

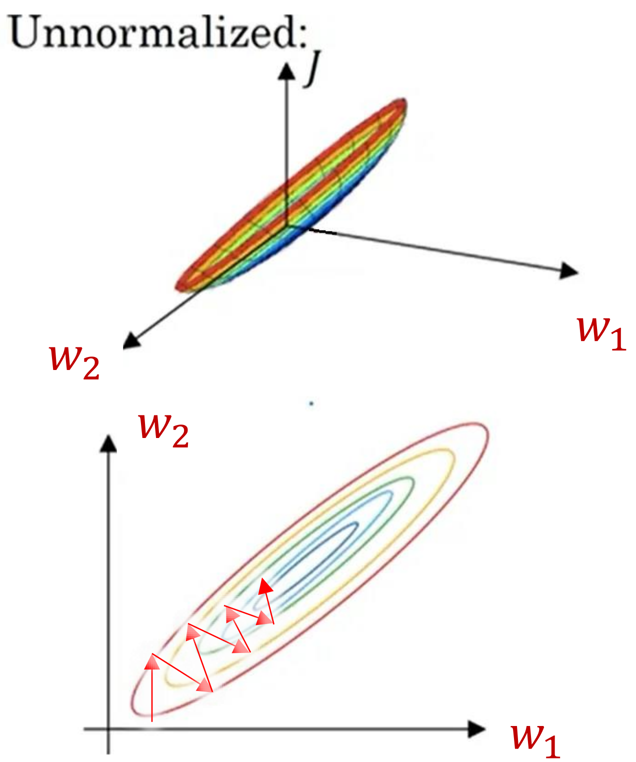
Features of different scales lead to an optimization landscape that is very asymmetric, e.g., the bowl shape becomes a skewed ellipsoid.

Intuition: Think of the slope of a curve
on a slightly downward sloping curve, a step of 1 meter will lower one's elevation by only a little.
on a strongly downward sloping curve, a step of 1 meter will lower one's elevation by a lot.

Solutions: Features of Different Scales

- Normalization/Standardization: $x_j \leftarrow \frac{x_j - \mu_j}{\sigma_j}$, where σ_j is the standard deviation of the feature j across the training data.
- Alternatives: Min-max scaling, robust scaling, etc.
- Other solution: Different learning rate γ_j for each weight.

Solutions: Features of Different Scales



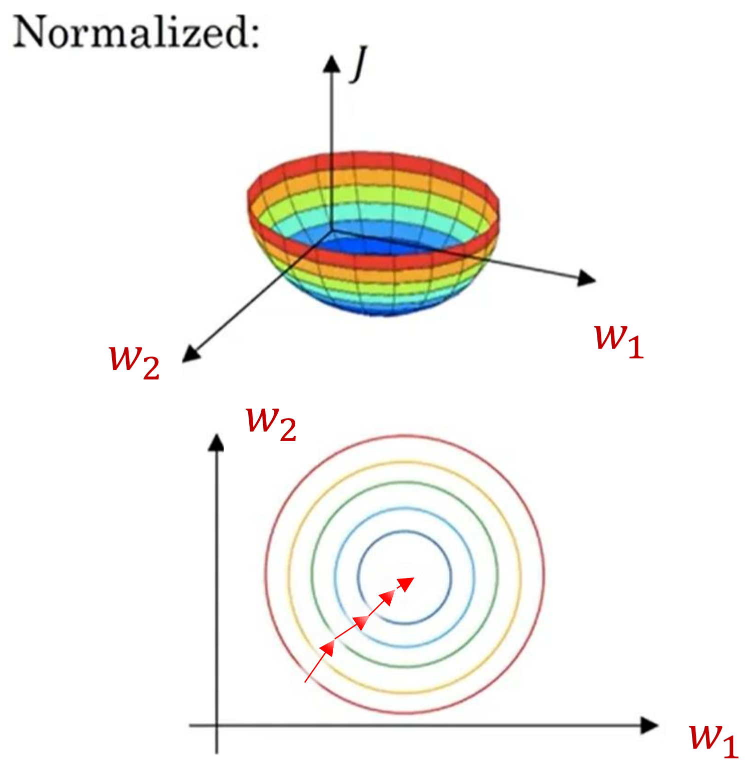


Image Credit: Andrew Ng

Variants of Gradient Descent

$$\boxed{w_j \leftarrow w_j - \gamma \frac{\partial J(w_0, w_1, \dots)}{\partial w_j} \qquad J_{MSE}(w) = \frac{1}{2N} \sum_{i=1}^{N} \left(h_w(x^{(i)}) - y^{(i)} \right)^2}$$

Note how Gradient Descent uses the complete data set for each update.

That can be inefficient for large data sets.

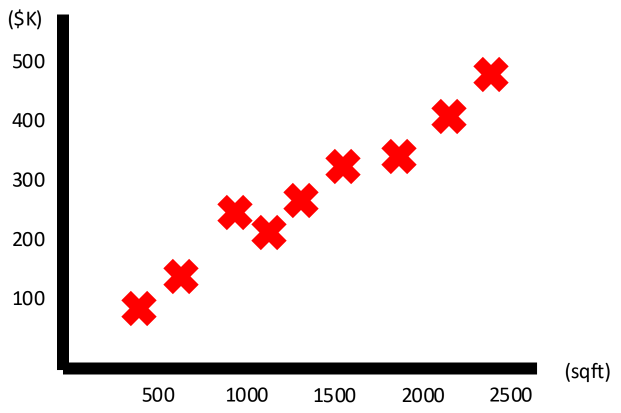
Idea: Let’s use a small set of data points for a single update, another small set of data points for the next update, and so on.

Key aspect: Randomness.

How about even using a single data point per iteration?

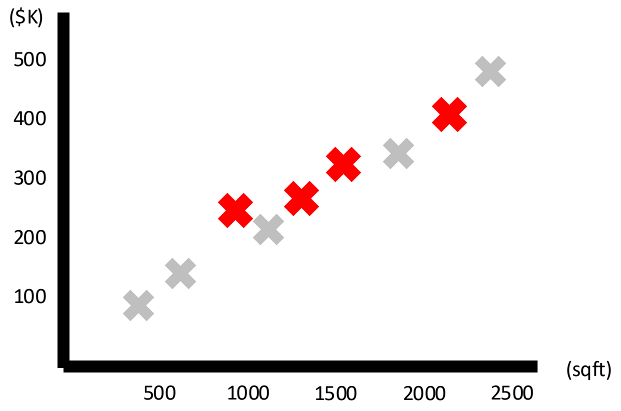
Variants of Gradient Descent

$$w_j \leftarrow w_j - \gamma \frac{\partial J(w_0, w_1, \dots)}{\partial w_j} \qquad J_{MSE}(w) = \frac{1}{2N} \sum_{i=1}^{N} \left(h_w(x^{(i)}) - y^{(i)}\right)^2$$



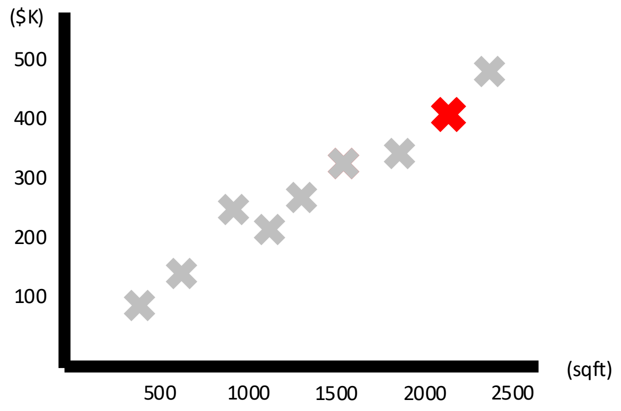
(Batch) Gradient Descent

- Consider all training examples



Mini-batch Gradient Descent

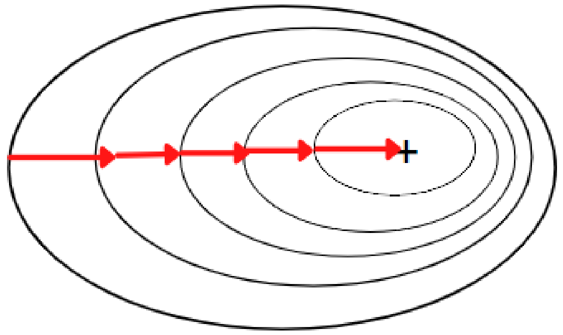
- Consider a subset of training examples at a time
- Cheaper (Faster) / iteration
- Randomness, may escape local minima

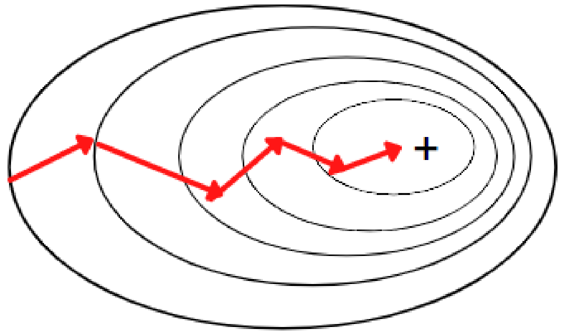


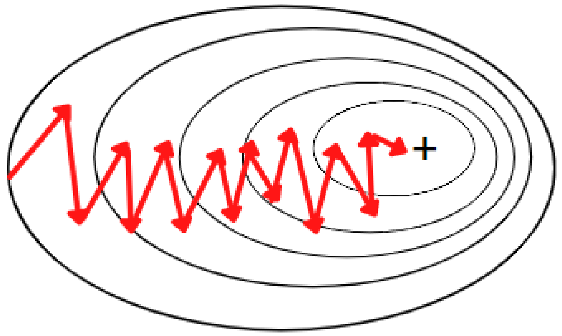
Stochastic Gradient Descent (SGD)

- Select one random data point at a time
- Cheapest (Fastest) / iteration
- More randomness, may escape local minima

Variants of Gradient Descent

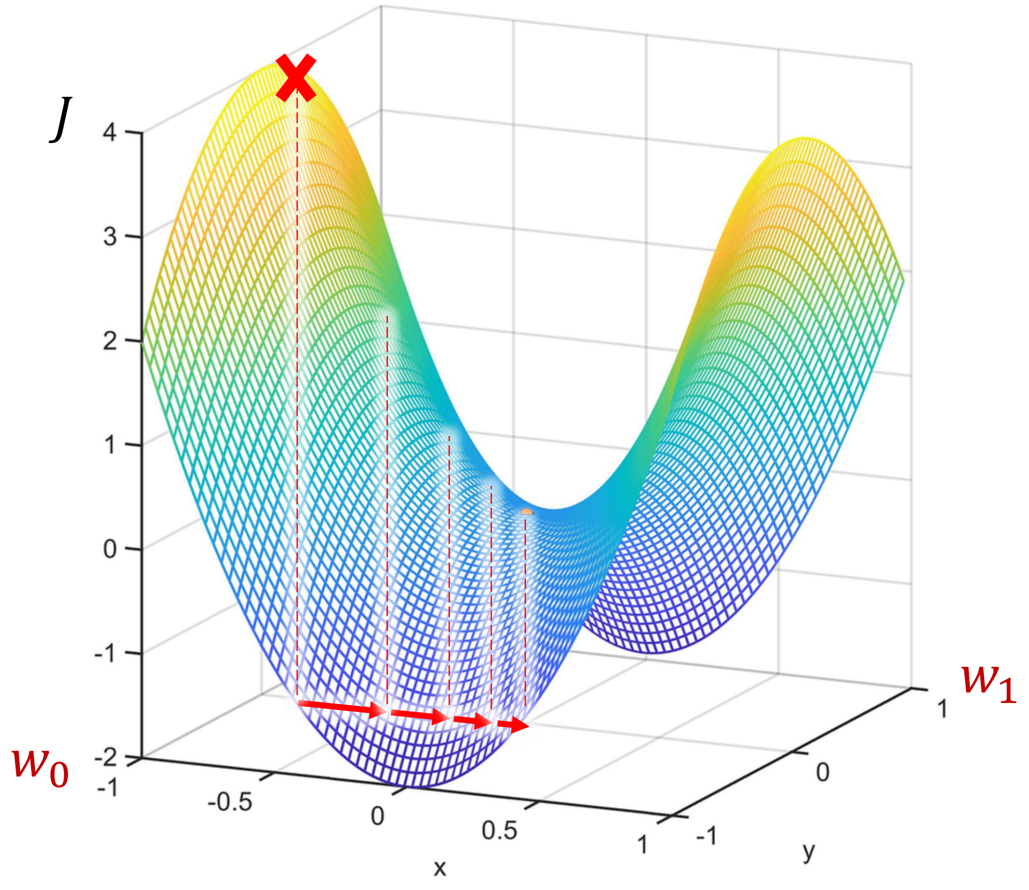

Batch Gradient Descent


Mini-Batch Gradient Descent

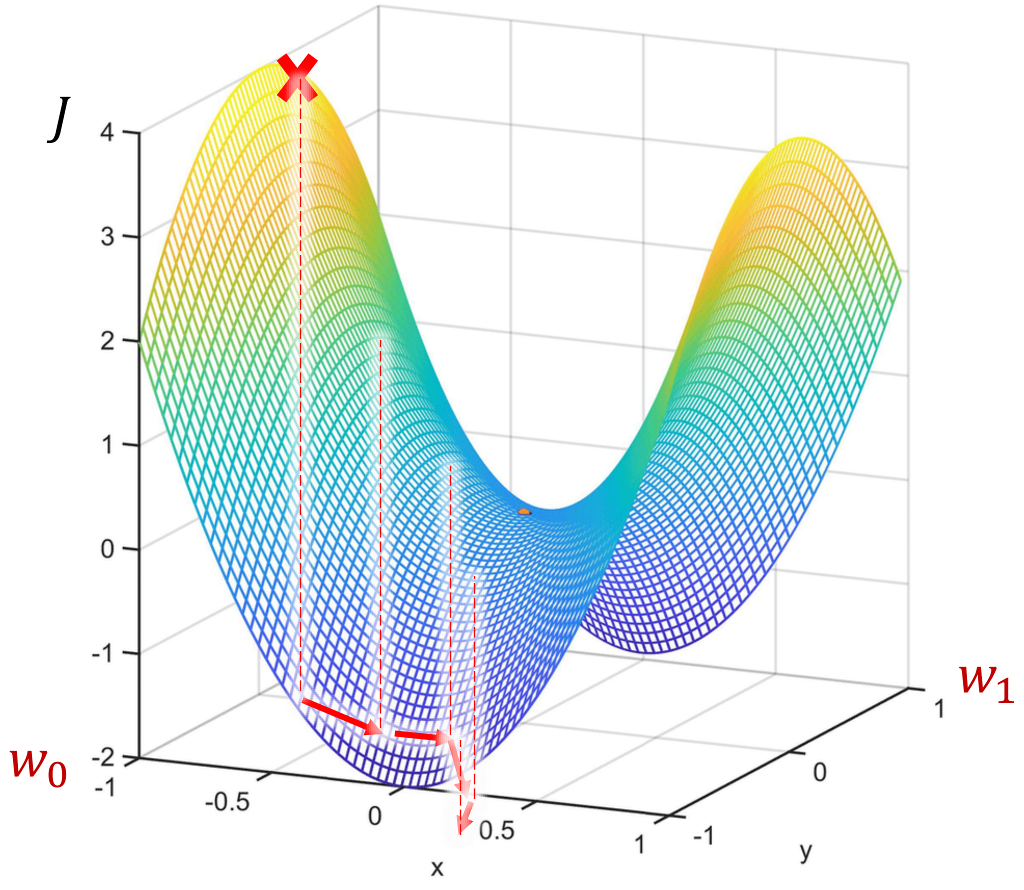

Stochastic Gradient Descent

Credit: analyticsvidhya.com

Escaping Local Minima / Plateaus on non-convex optimization



Batch Gradient Descent



Stochastic/Mini-batch Gradient Descent

Learning Algorithms: Comparison

$$w_j \leftarrow w_j - \gamma \frac{\partial J(w_0, w_1, \ldots)}{\partial w_j}$$

$$w = (X^T X)^{-1} X^T Y$$

	Gradient Descent	Normal Equation
Need to choose learning rate γ	Yes	No
Iteration(s)	Many	None
Large number of features d?	No problem	Slow, $(X^T X)^{-1} \rightarrow O(d^3)$
Feature scaling?	May be necessary	Not necessary
Constraints	-	$X^T X$ needs to be invertible

Further Reading (Optional)

- History of regression (The origins and uses of regression analysis, 1997)
- Feature encoding
- Robust scaling
- Normal equation derivation
- Complexity of inverting matrix
- Proof: A convex function has a single global minimum
- Proof: MSE loss function is convex for linear regression
- Different learning rate for each weight

Summary

- Linear Regression: **fitting a line** to data
- Linear Model
  - d dimensional input features: $h_w(x) = \sum_{j=0}^{d} w_j x_j = w^T x$
- Finding the best function, i.e., one that minimizes the loss
  - Normal Equation: **set derivative to 0, solve**
  - Gradient Descent
    - Gradient Descent Algorithm: **follow –gradient** to reduce error
    - Linear Regression with Gradient Descent: **convex** optimization, **one minimum**
    - Problem: Features of Different Scales: **normalize!**
    - Variants of Gradient Descent: batch, mini-batch, stochastic

Coming Up Next Week

- Logistic Regression

To Do

- **Lecture Training 5**
  - +250 EXP
  - +100 Early bird bonus