APS1070

Foundations of Data Analytics and Machine Learning

Winter 2022

Week 9:
- *Empirical Risk Minimization*
- *Maximum Likelihood Estimation*
- *Linear Regression*

Sinisa Colic and Samin Aref

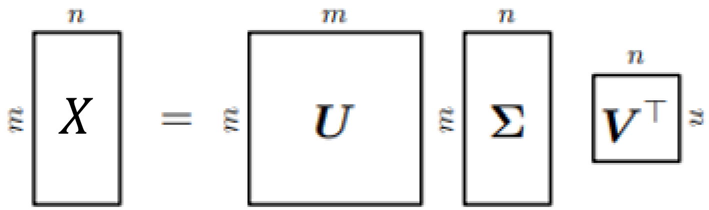
Slide Attribution

These slides contain materials from various sources. Special thanks to the following authors:

- Roger Grosse
- William Fleshman
- Lisa Zhang
- Andrew Ng
- Jason Riordon

Last Time

- Matrix decompositions, dimensionality reduction and interpretations.
  - SVD
  - PCA
  - Applications
  - Vector Calculus



- Today we will return to learning algorithms, this time we will focus on **model-based learning algorithms** starting with linear regression.

Vector Calculus Examples:

Example:

- Given: $f(x) = Ax$, A = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, x $= [x_1\ x_2]^T$

- Find df(x)/dx

Example:

- Given: $f(x) = x^T A x,$ A = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix},$ x $= [x_1\ x_2]^T$

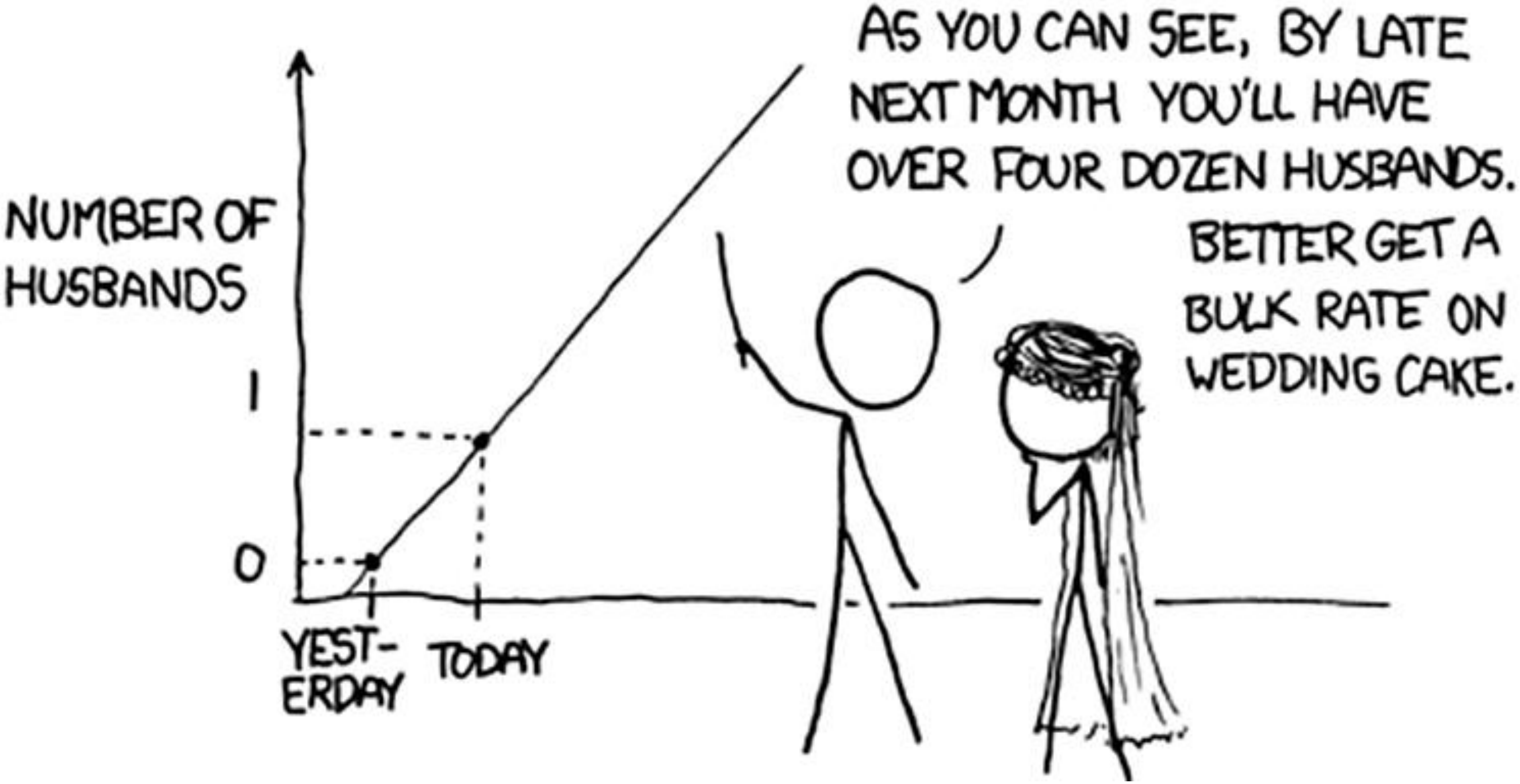
- Find df(x)/dx

Decision Making

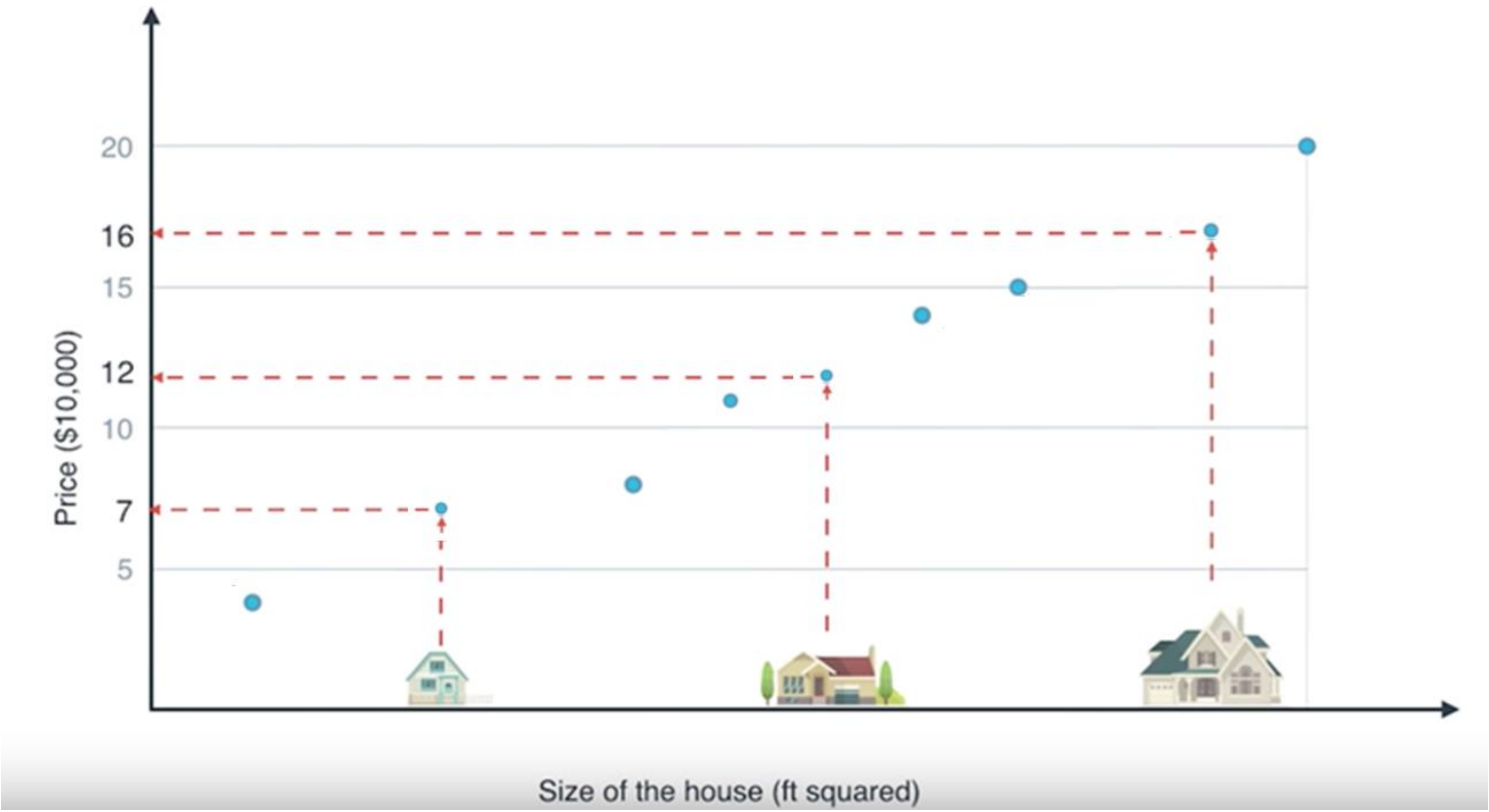
- We often need to make some decisions (or predictions) given some data.

- Two common types of decisions that we make are:
  - Classification
    - Discrete number of possibilities
  - Regression
    - Continuous number of real-valued possibilities

Linear Regression



Example: House Price Prediction



Recap: Learning Algorithms/Models

- We've seen earlier that there are several approaches to solving regression problems.
- **Instance-Based**
  - The first couple lectures focused on algorithms require long-term storage of data in memory in order to make predictions/decisions.
- **Model-Based**
  - Now we will introduce learning algorithms that make replace samples with model parameters for making predictions.

Agenda

- Data
- Empirical Risk Minimization
- Gradient Descent
- Maximum Likelihood Estimation
- Negative Log-Likelihood
- Application of Linear Regression

Theme:
Linear Regression

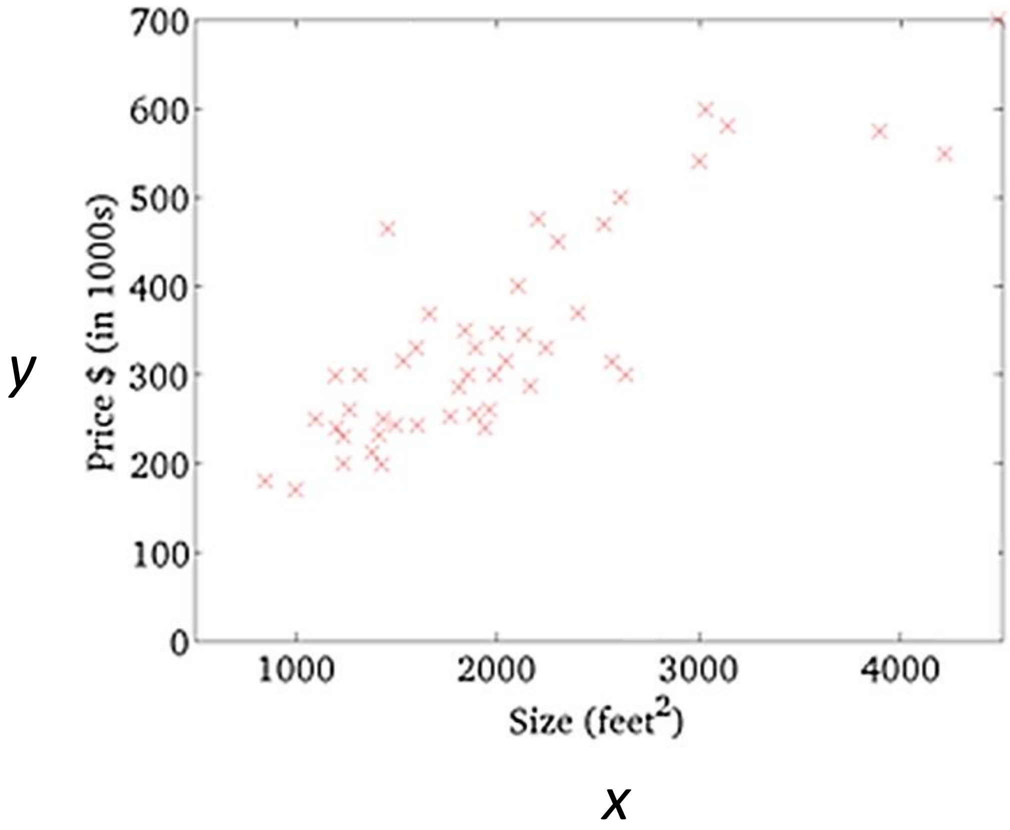
Linear Regression
(Empirical Risk Minimization)

Readings:
- **Chapter 7 MML Textbook**
- **Chapter 8.1-2 MML Textbook**

Problem Setup

Size in feet^2 (x)	Price in 1000's (y)
320	148
450	210
845	362
1043	440
1160	550
…	…

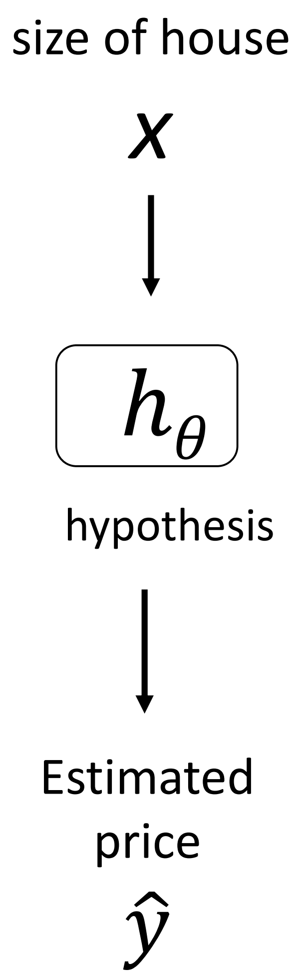
$$\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$$



Linear Model

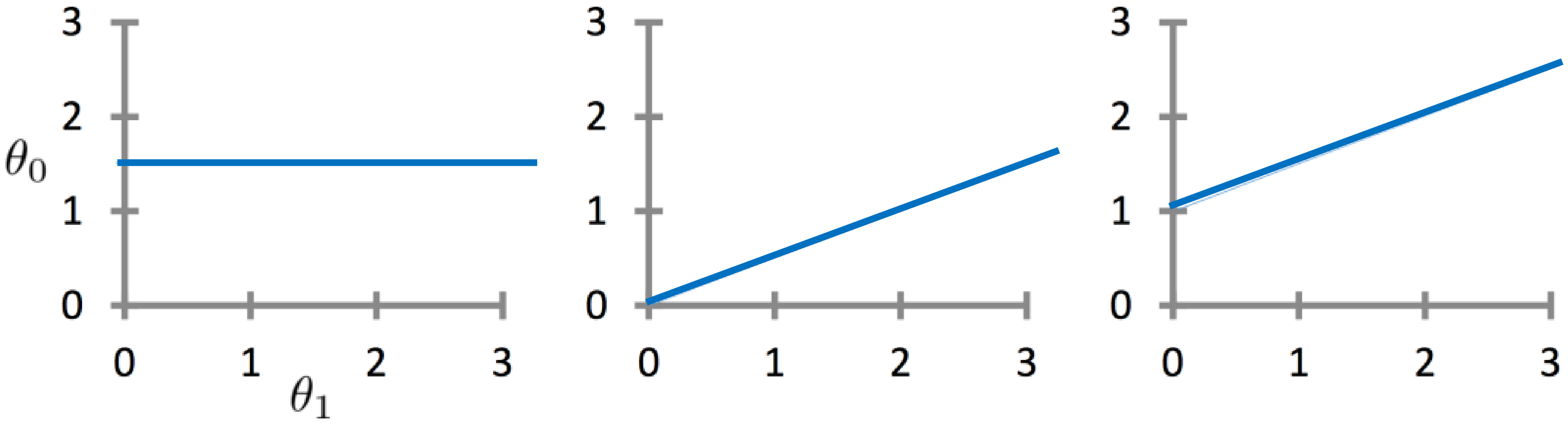
- Parameters $\theta = (\theta_0, \theta_1)$ can be used to estimate a hypothesis about the data
- If the data is correctly predicted according to the hypothesis h_θ, then $y \sim h_\theta(x) = \theta_0 + \theta_1 x$
- We can then estimate y for new values of x using our h_θ

- If $h_\theta\ (x)$ is a linear function of a real number x, this procedure is called linear regression.



Many Hypotheses to Choose From

$$h_\theta(x) = \theta_0 + \theta_1 x$$

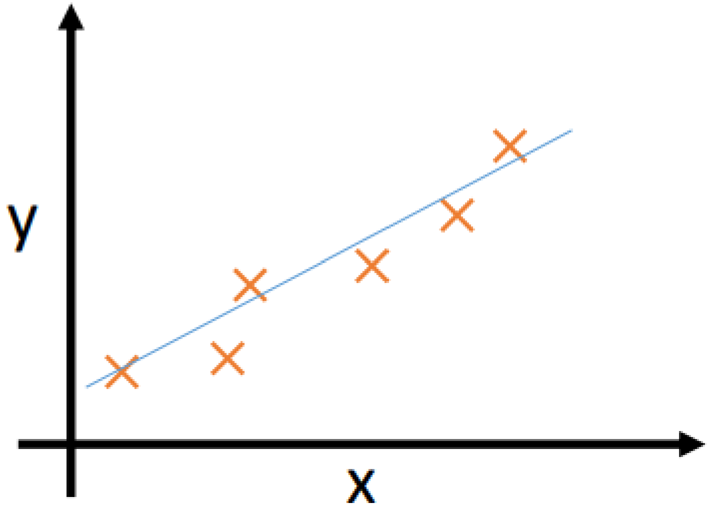


$\theta_0 = 1.5$
$\theta_1 = 0$

$\theta_0 = 0$
$\theta_1 = 0.5$

$\theta_0 = 1$
$\theta_1 = 0.5$

What is the best Model?



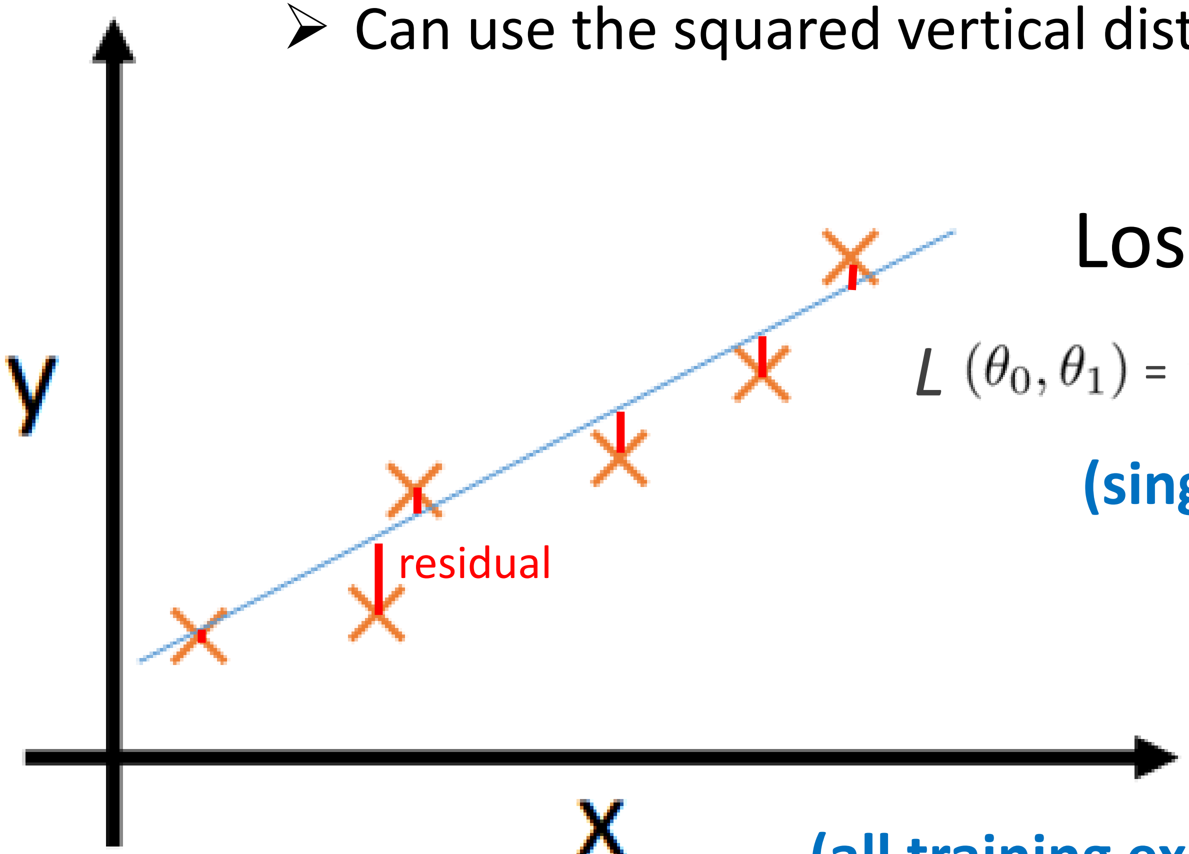
Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples

$$\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$$

But what does "close" mean?

Loss Function

- Can use the squared vertical distance



Loss Function

$$L(\theta_0, \theta_1) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

(single example)

(all training example)

Cost Function $$J(\theta_0, \theta_1) = \frac{1}{2N}\sum_{i=1}^{N}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Learning a Hypothesis

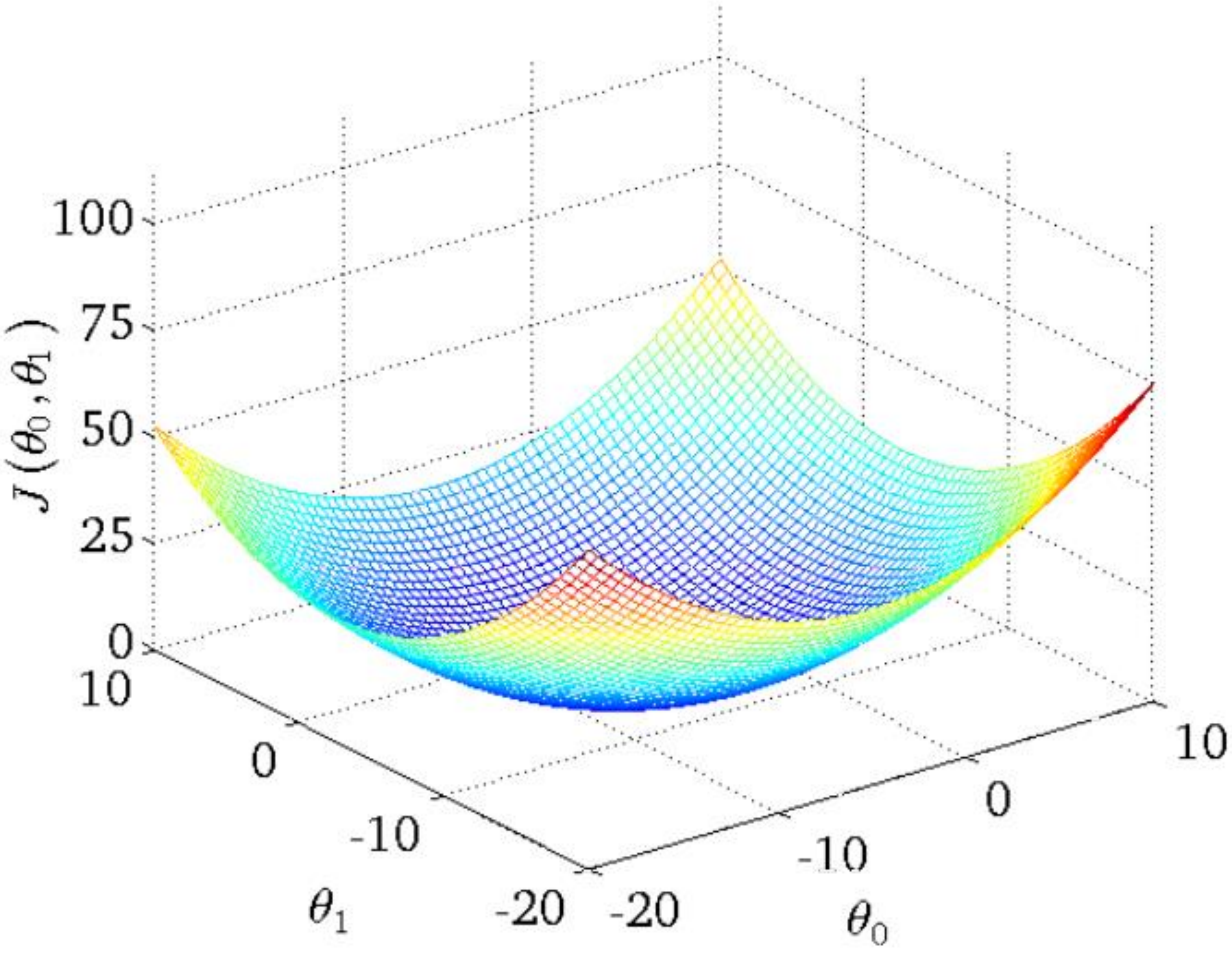
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2N} \sum_{i=1}^{N} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Cost Function Surface Plot

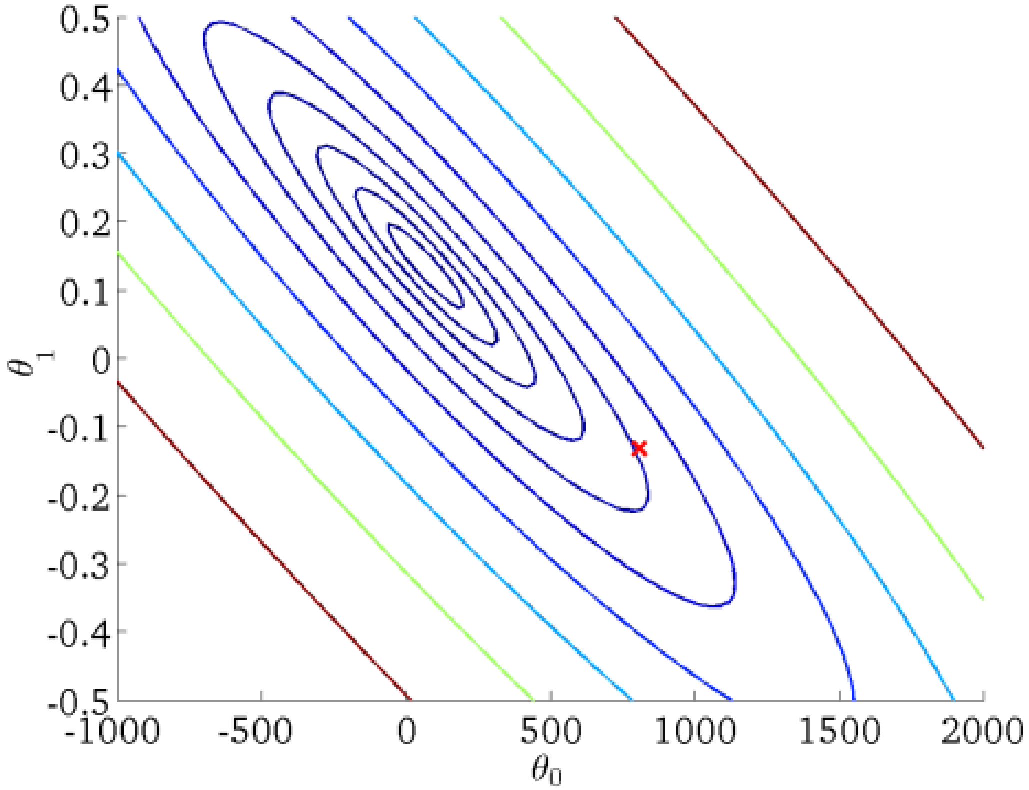


Notice the convexity of the plot

Cost Function Surface Plot

- We can visualize our **Cost Function** $J(\theta_0, \theta_1)$ from above as a contour plot.

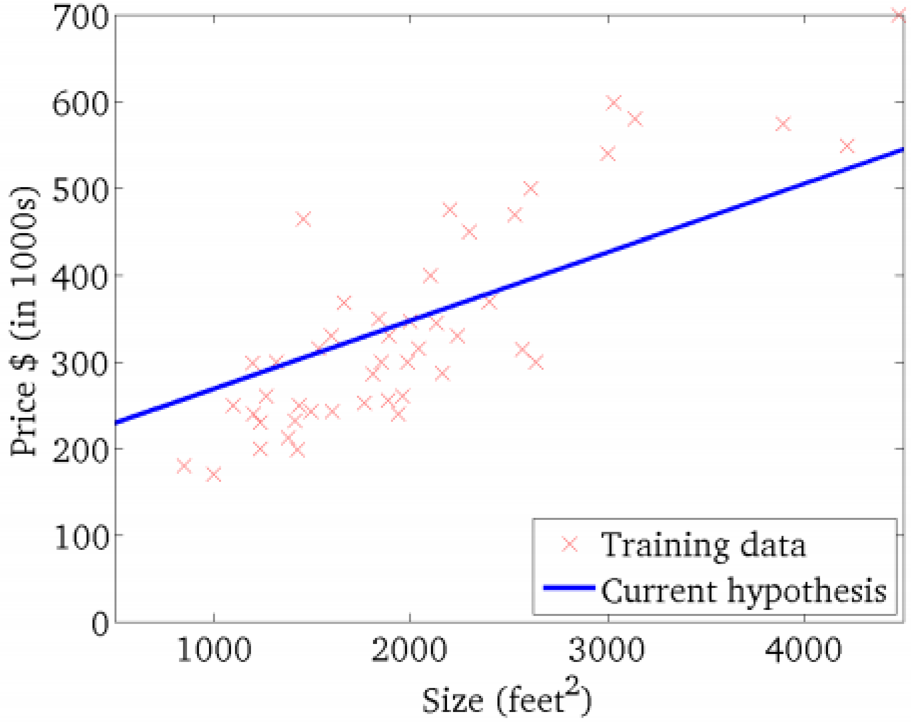
- Where contours will be represented as curves in the graph where values of the cost, $J(\theta_0, \theta_1)$ are constant.



Contour Plots

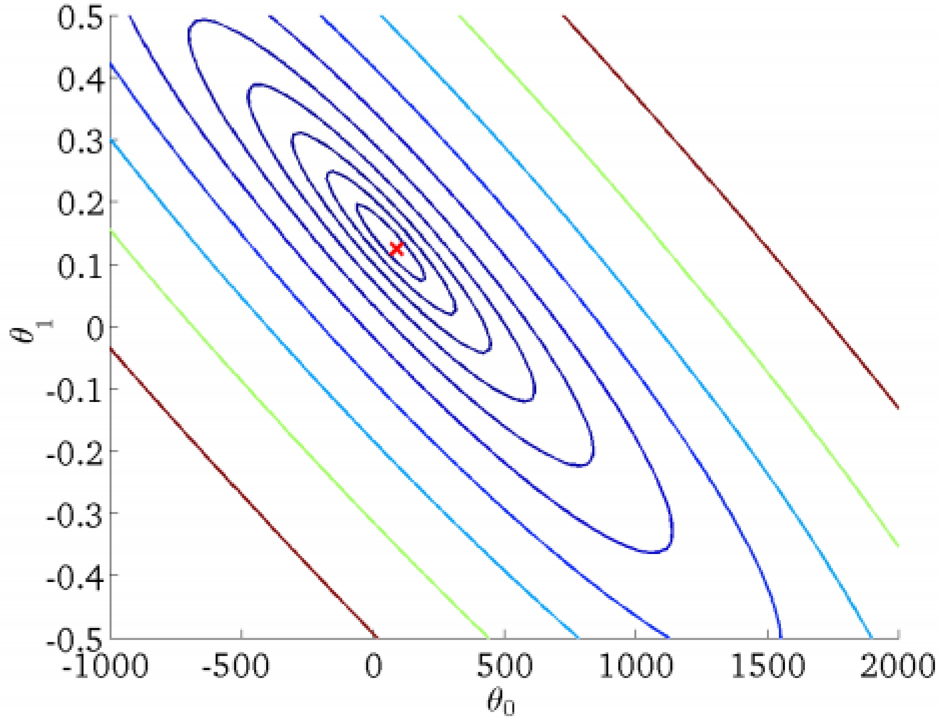
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

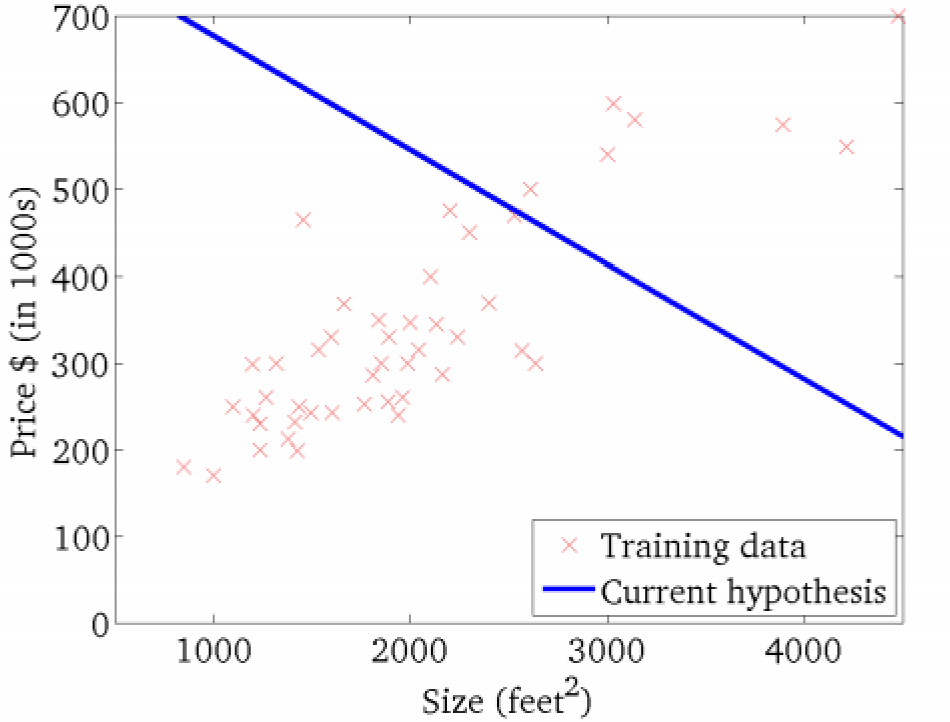
(function of the parameters θ_0, θ_1)



Cost Function Contour Plot

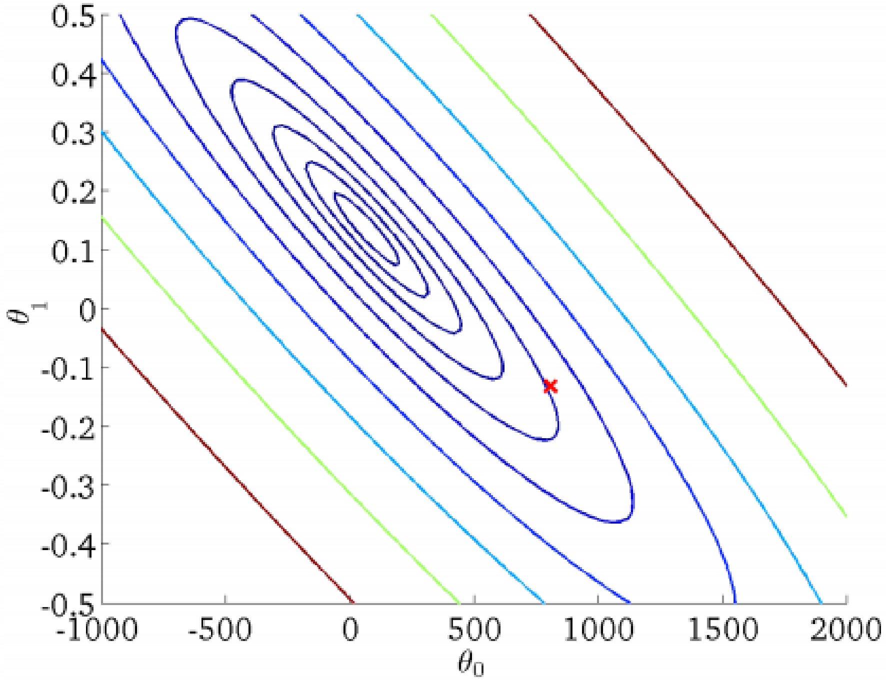
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Direct Solution

- The minimum would occur where partial derivatives equal zero:

$$\frac{dJ}{d\theta_i} = 0$$

- Which results in a direct solution:

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

Show this!

Linear regression is one of a handful of models that permit direct solutions

Direct Solution Derivation

Show: $\theta = (X^T X)^{-1} X^T \mathrm{y}$

Alternate Approach?

- Q: Great! If we have an analytical solution to finding optimal parameters, what's the issue?

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

- A: In high dimensional spaces computing matrix inversion is expensive!

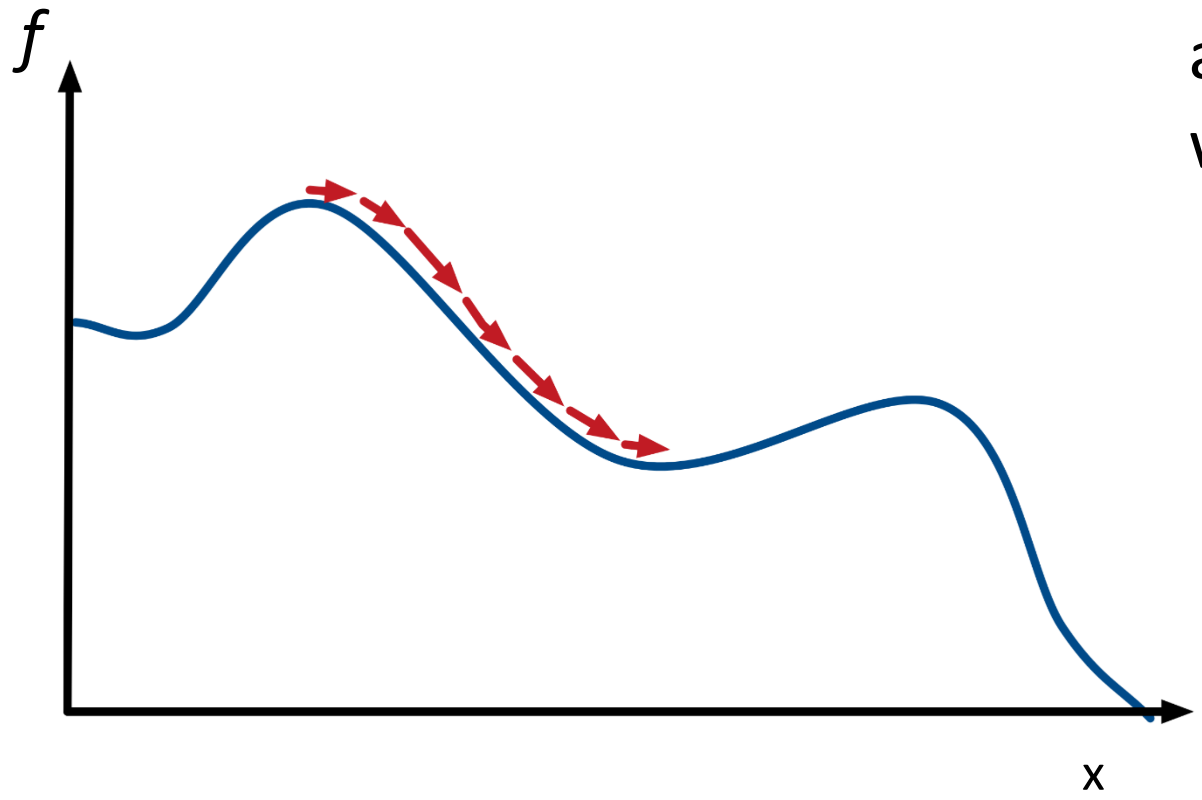
- Instead, we will use a more robust solution known as gradient descent

Gradient Descent

Gradient descent is an iterative algorithm.

We **initialize** a starting point and **repeatedly adjust** based on the direction of **steepest descent**.

Gradient Descent in 1D

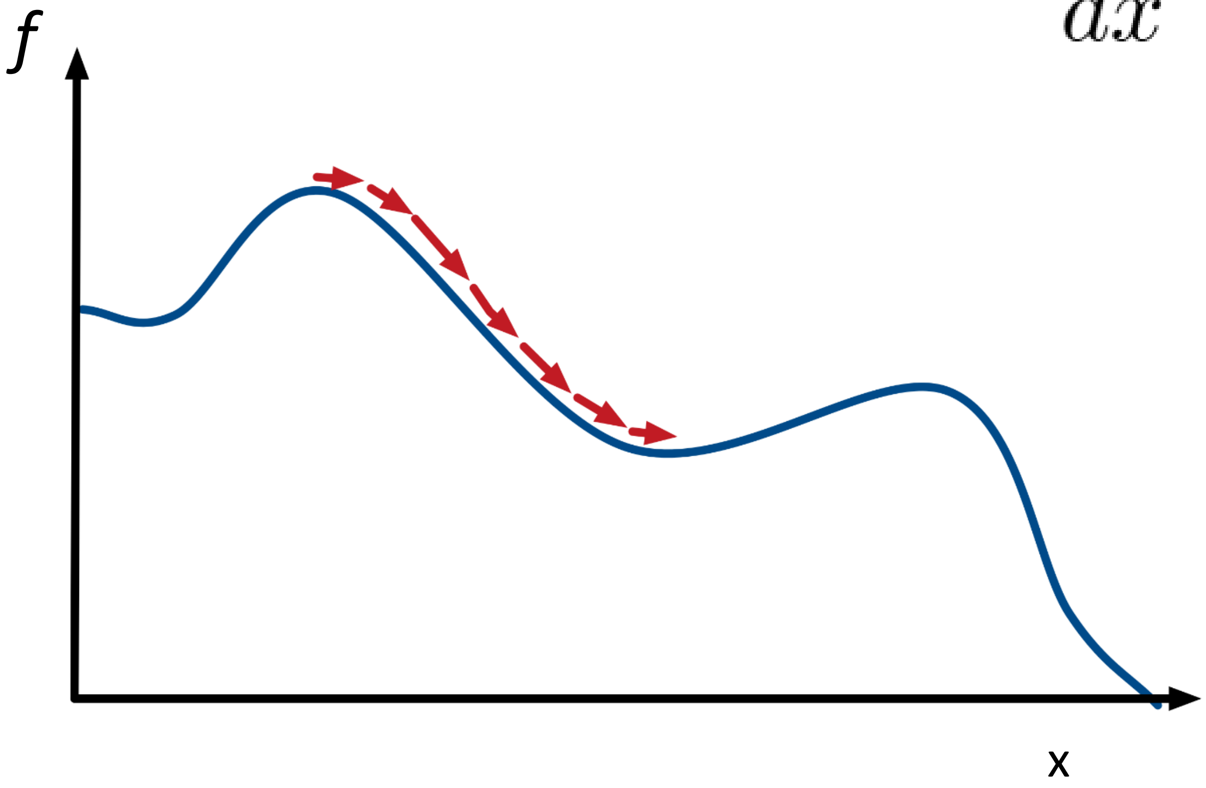


To minimize f(x), we start with a random point and iterate with the update rule:

$$x_t \leftarrow x_{t-1} - \alpha \frac{df}{dx}(x_{t-1})$$

Things to Consider

$$x_t \leftarrow x_{t-1} - \alpha \frac{df}{dx}(x_{t-1})$$



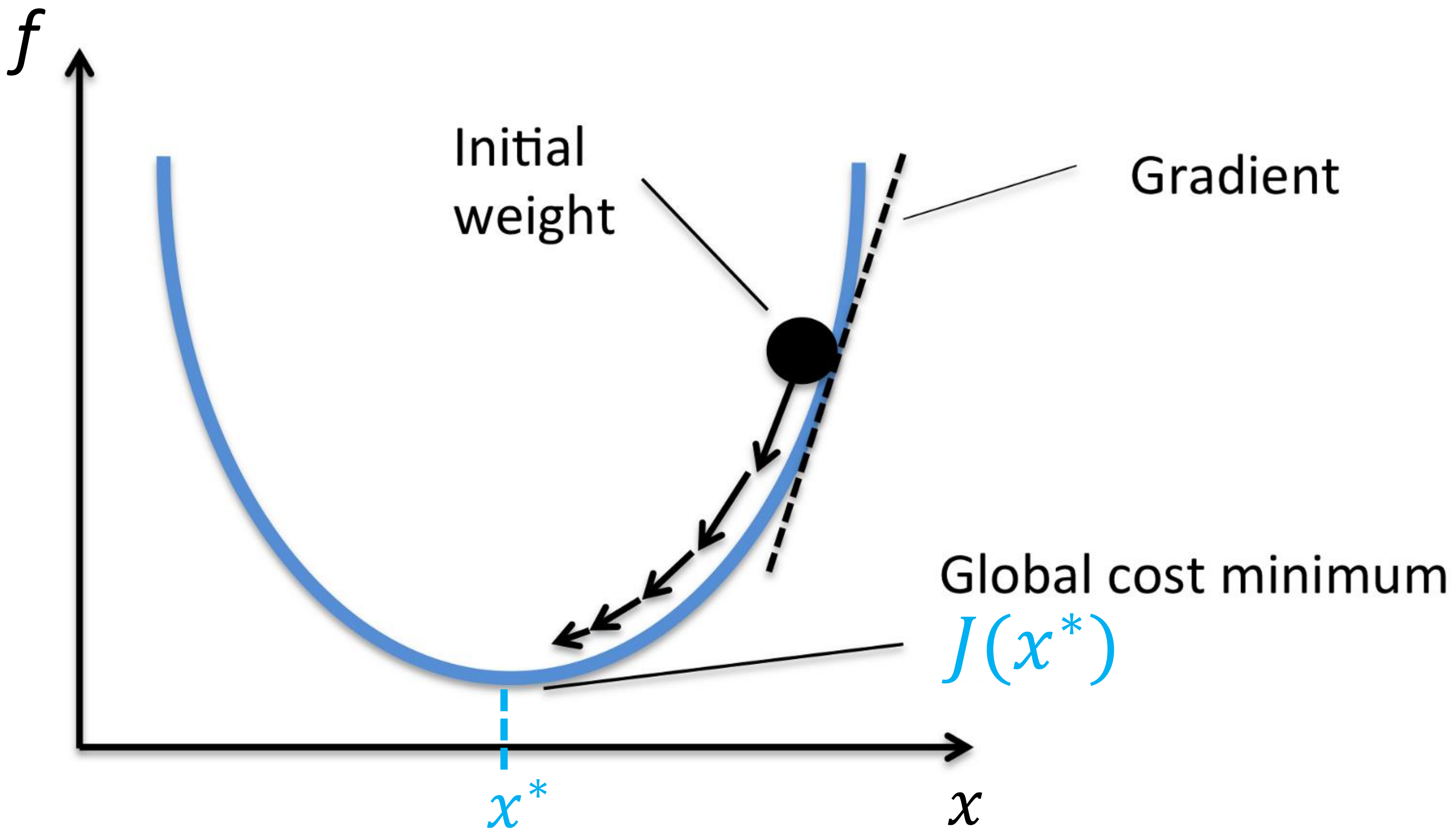
- Local vs global minima?
- Convexity?
- Learning Rate? (α)

GD Summary

Step 1	Initialize weights
Step 2	Compute output based on weights
Step 3	Compute Error
Step 4	Compute Gradients
Step 5	Update the weights
Step 6	Go to step 2 until the error is acceptable

Role of Learning Rate?

Notice the convexity of the plot

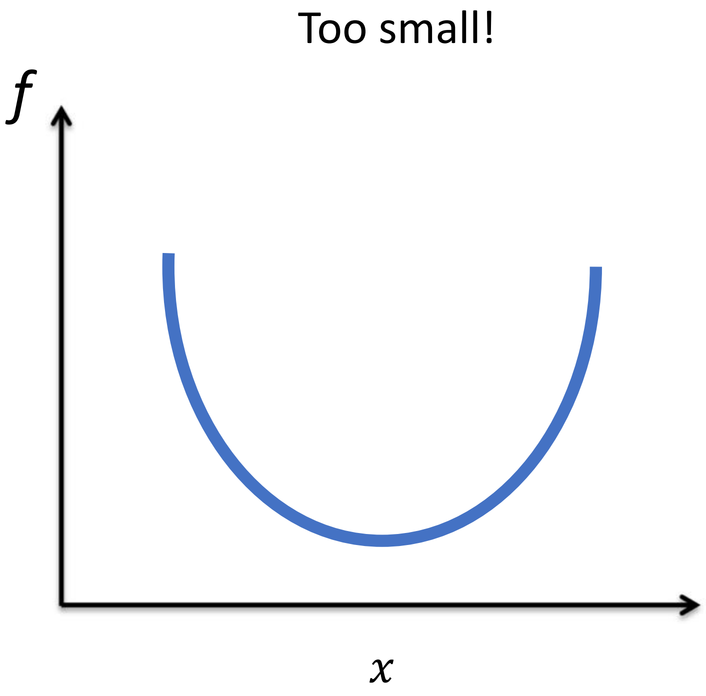


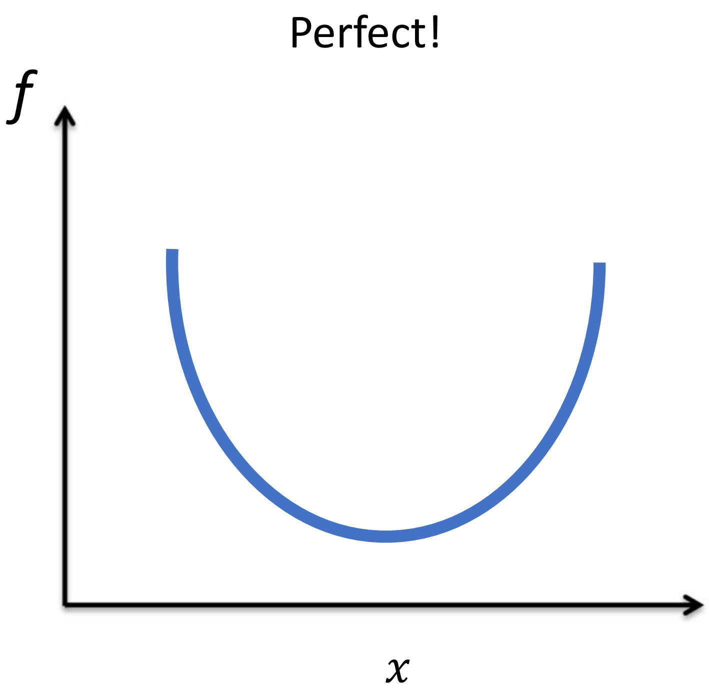
- Observe:
  - if $\partial \mathcal{J} / \partial w_j > 0$, then increasing w_j increases $\mathcal{J}$.
  - if $\partial \mathcal{J} / \partial w_j < 0$, then increasing w_j decreases $\mathcal{J}$.
- The following update decreases the cost function:

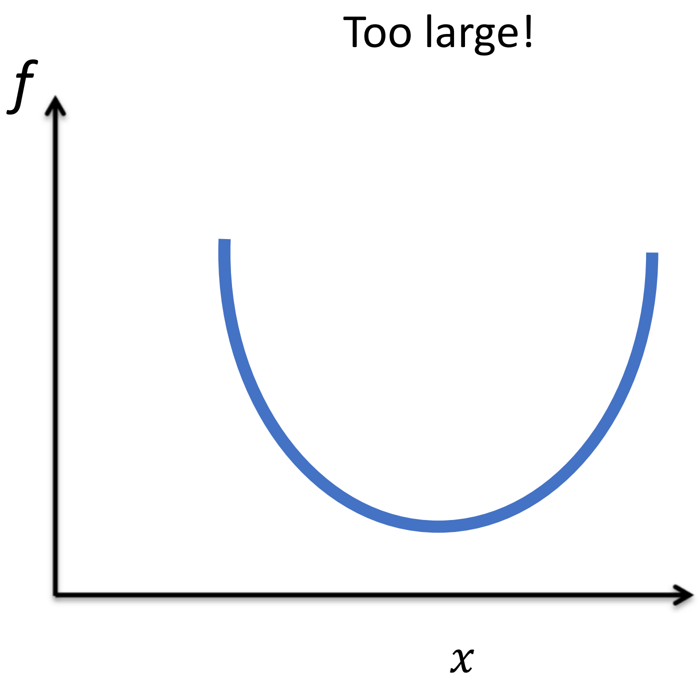
$$
\begin{aligned}
w_j &\leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j} \\
&= w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, x_j^{(i)}
\end{aligned}
$$

- α is a learning rate. The larger it is, the faster $\mathbf{w}$ changes.
  - We'll see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001

Role of Learning Rate?



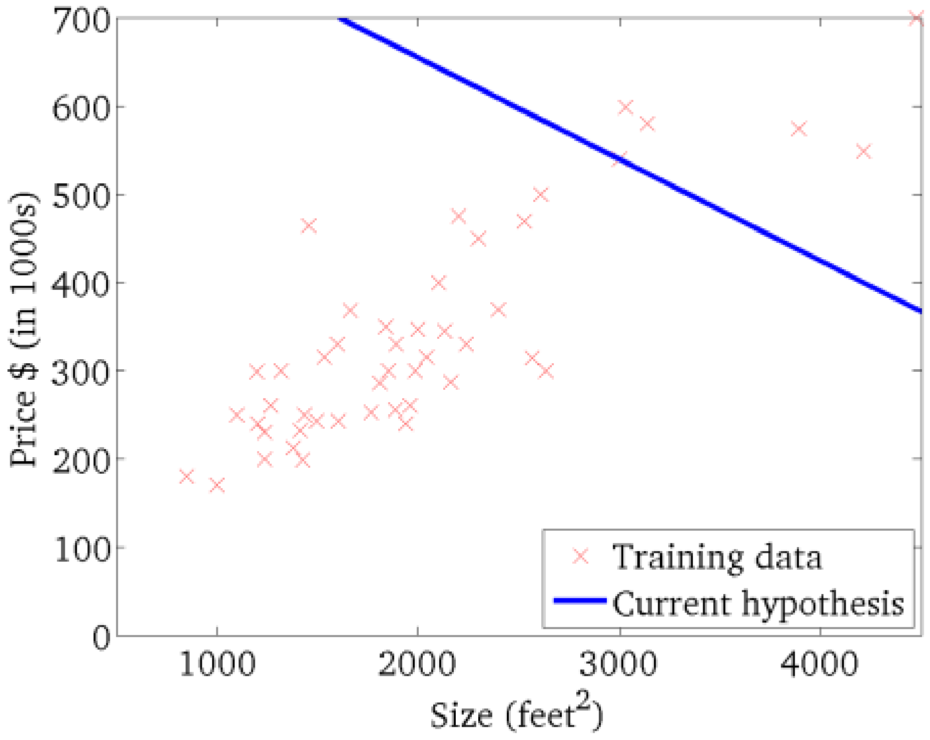




Gradient Descent Example -1

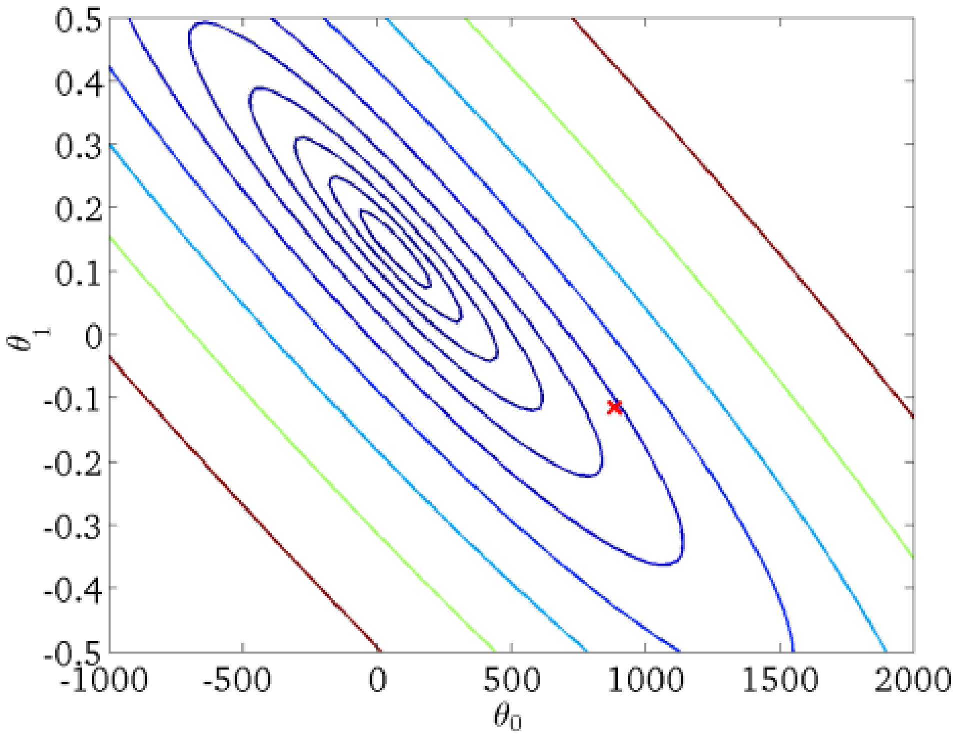
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

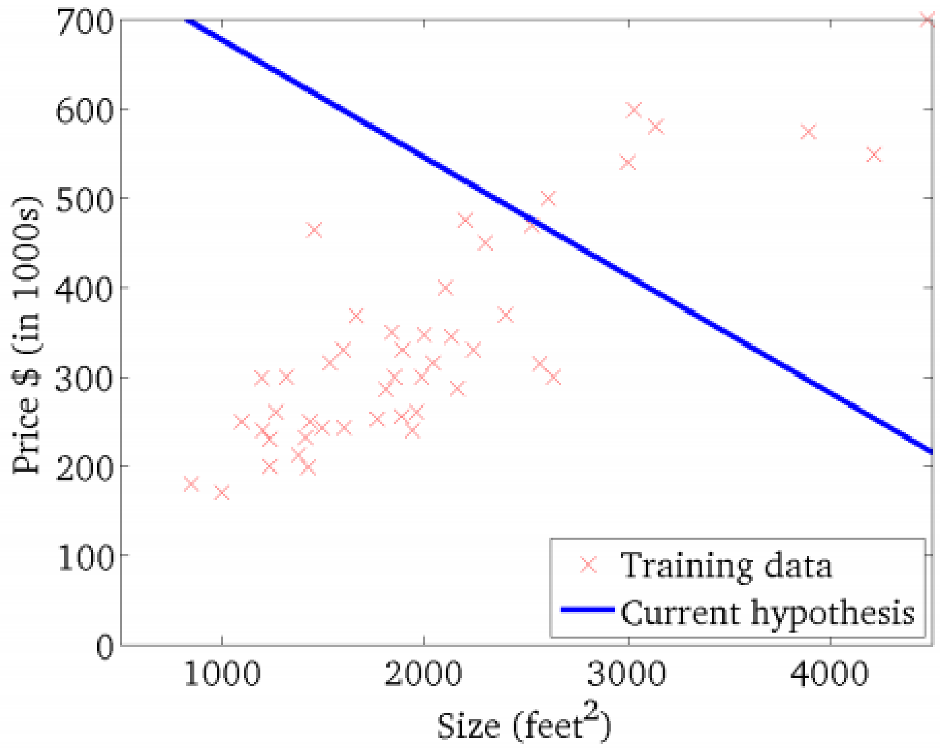
(function of the parameters θ_0, θ_1)



Gradient Descent Example -2

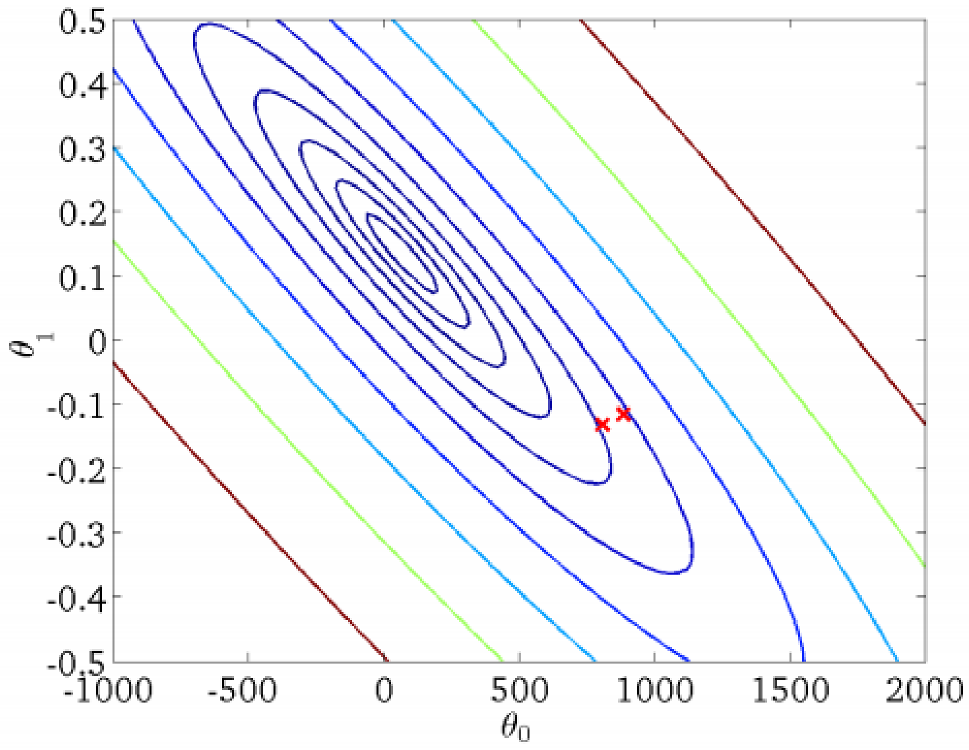
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

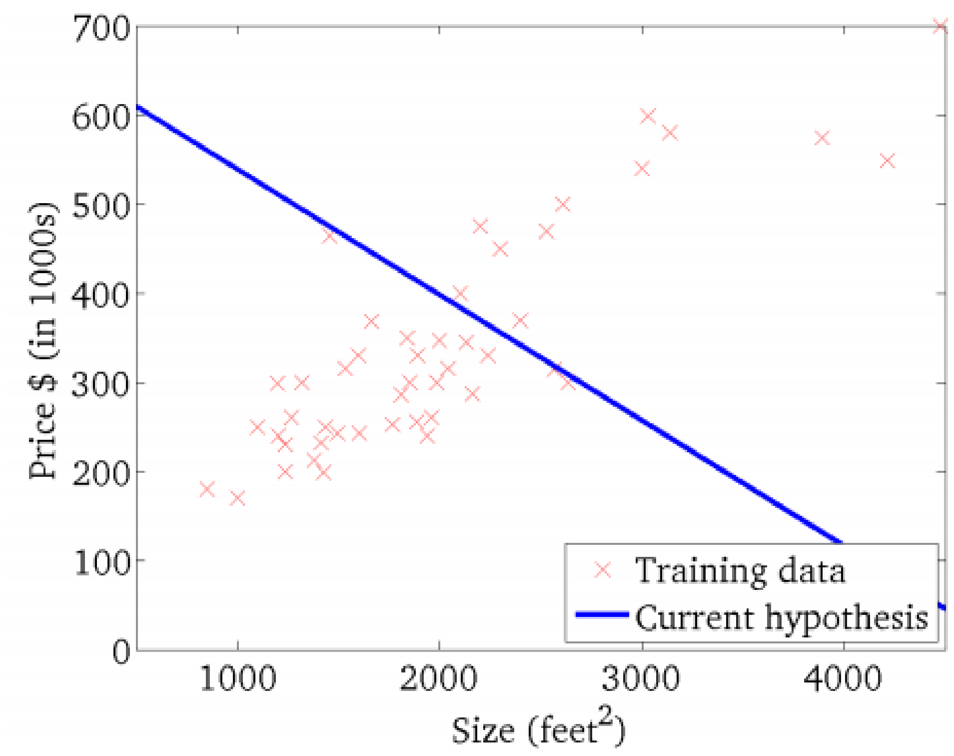
(function of the parameters θ_0, θ_1)



Gradient Descent Example -3

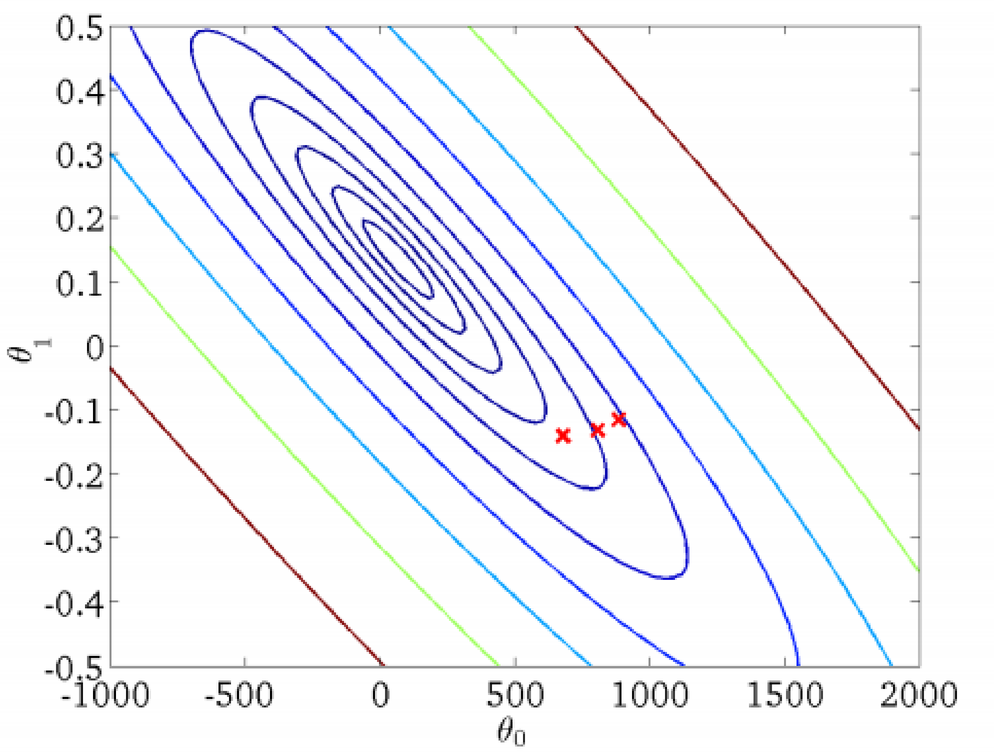
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

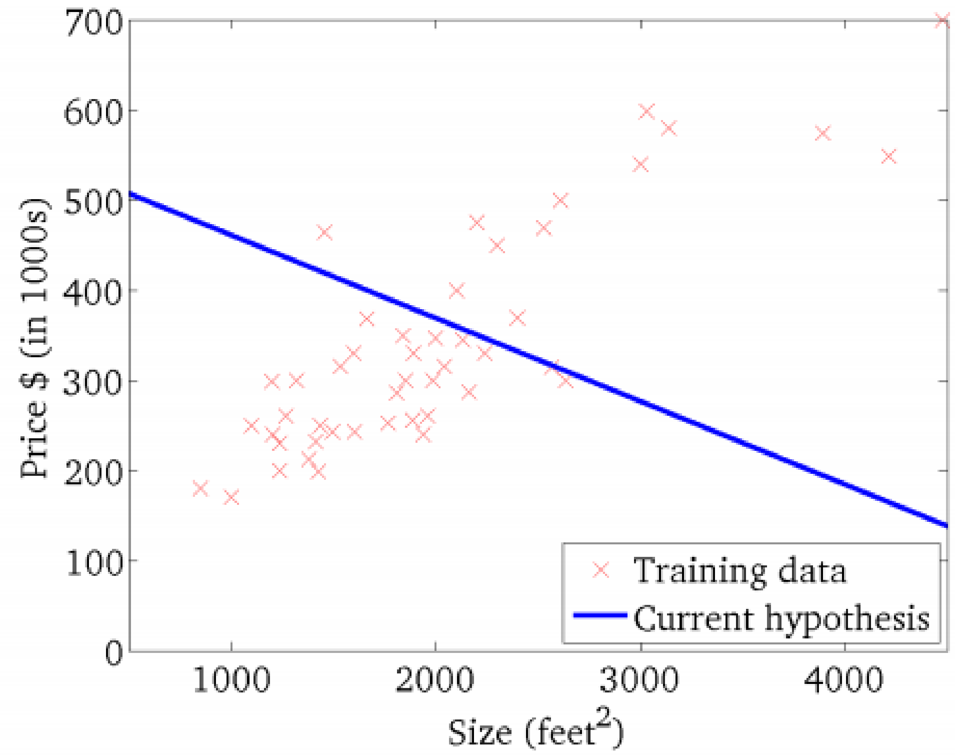
(function of the parameters θ_0, θ_1)



Gradient Descent Example -4

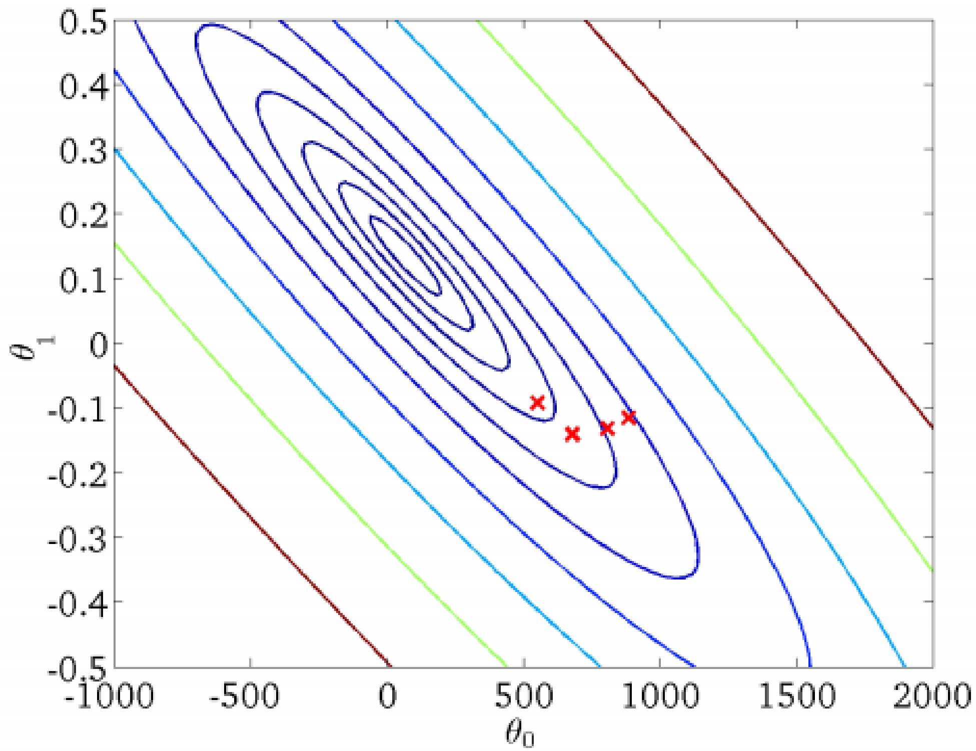
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

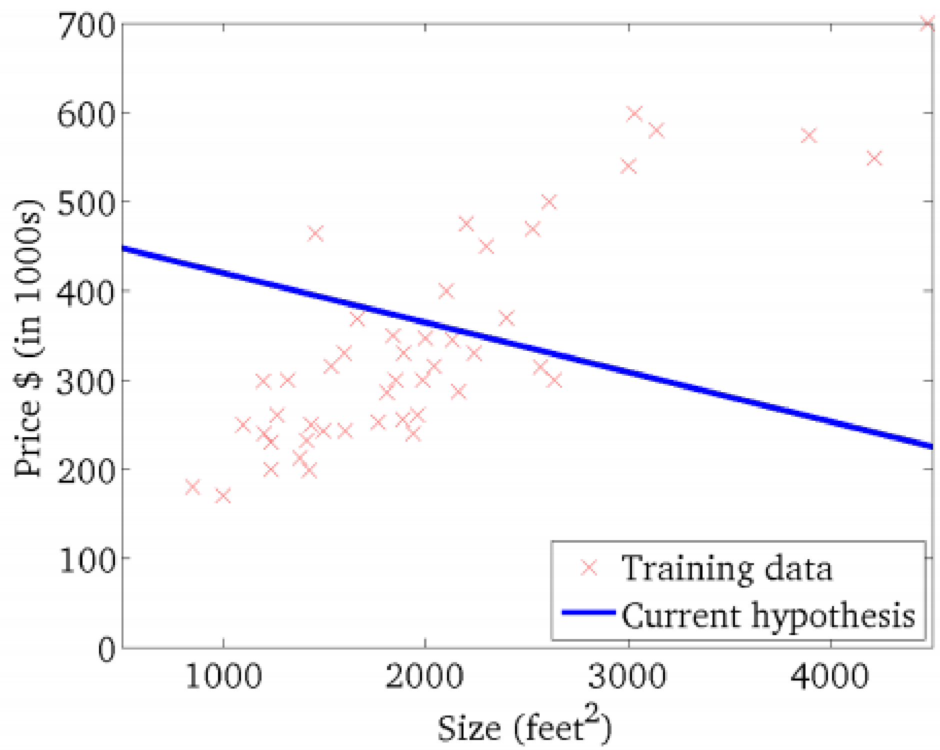
(function of the parameters θ_0, θ_1)



Gradient Descent Example -5

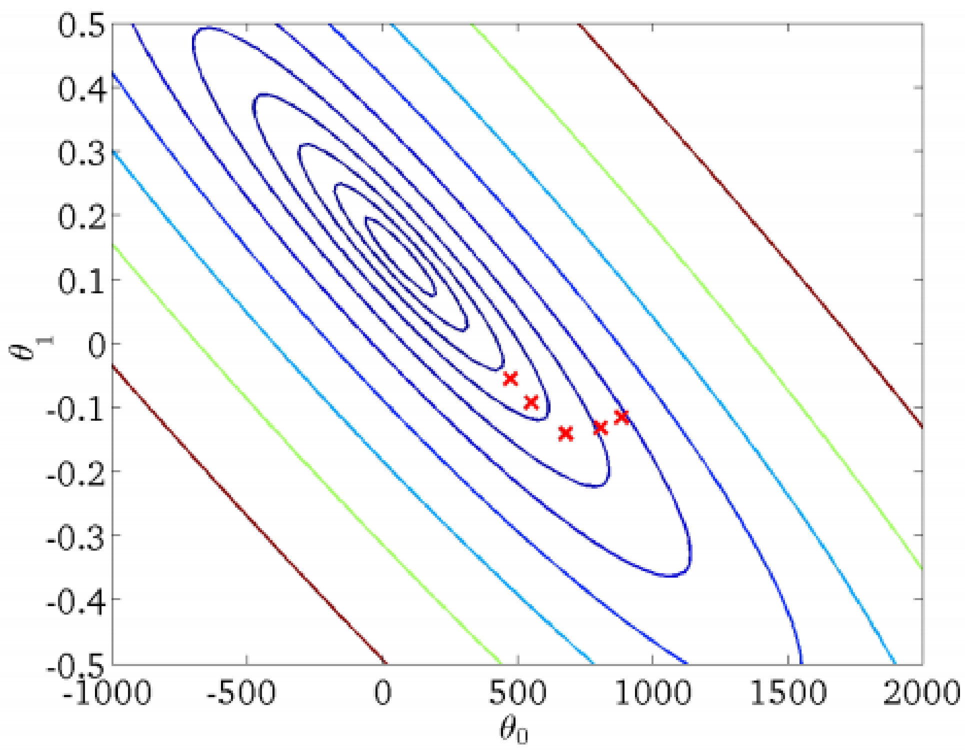
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

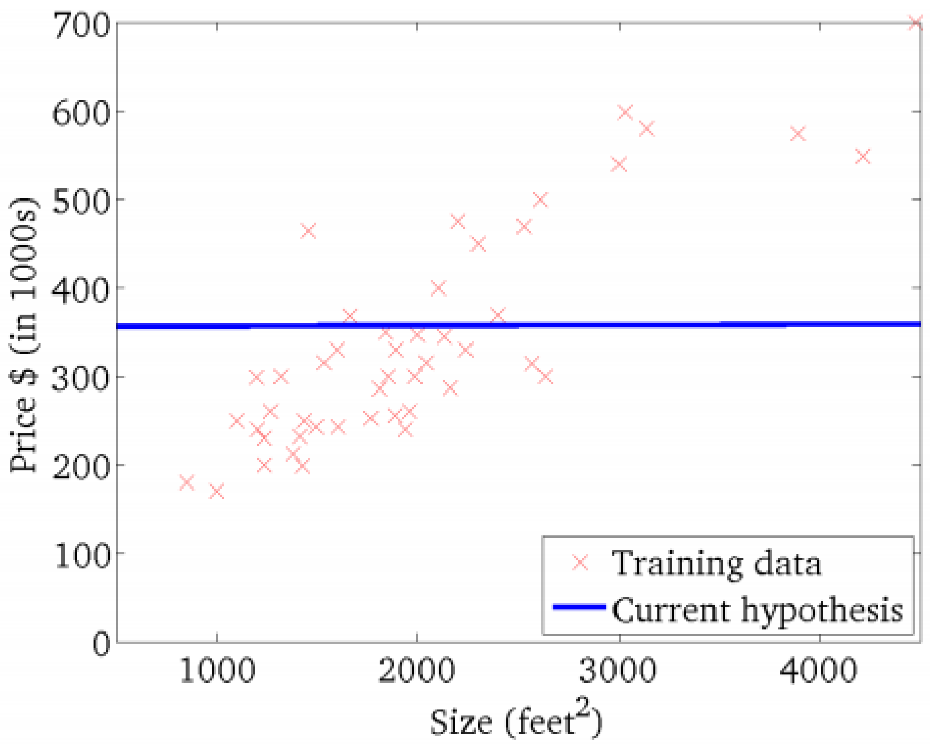
(function of the parameters θ_0, θ_1)



Gradient Descent Example -6

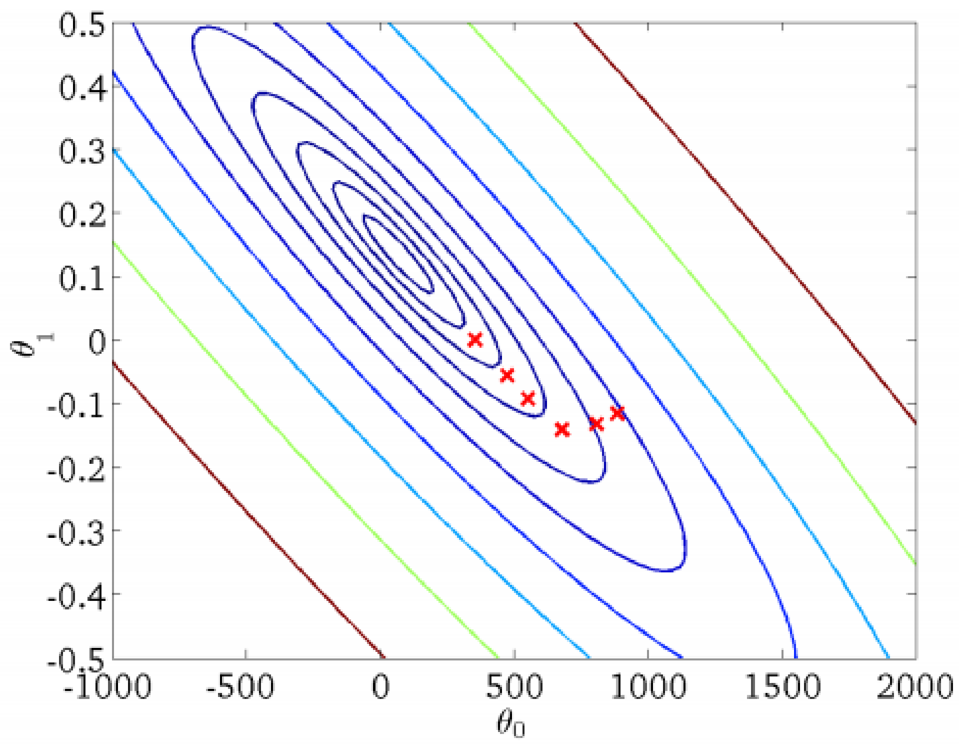
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

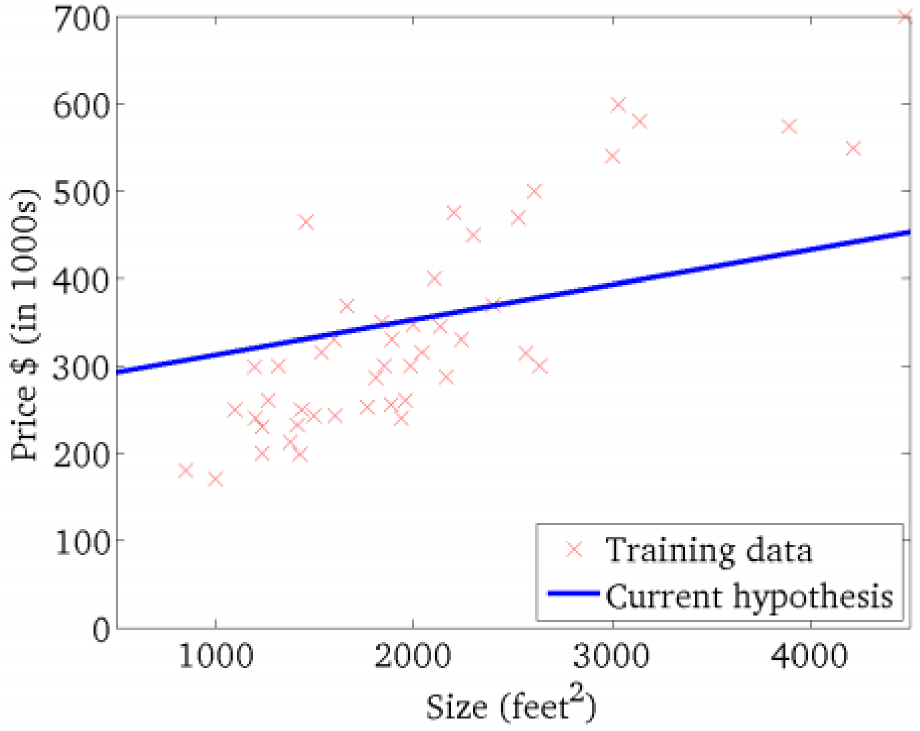
(function of the parameters θ_0, θ_1)



Gradient Descent Example -7

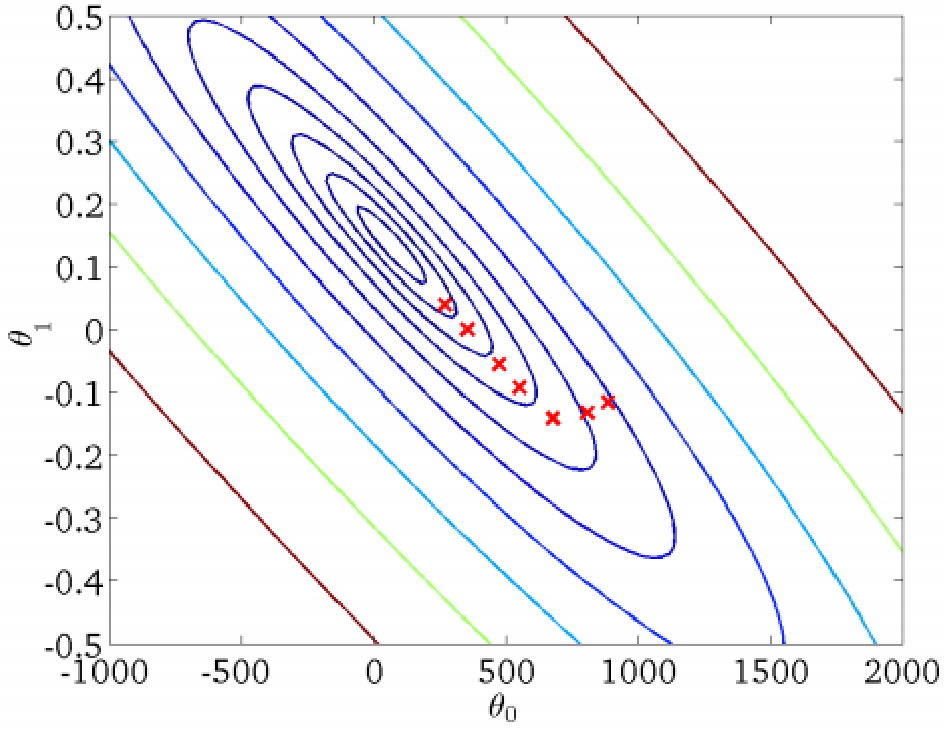
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

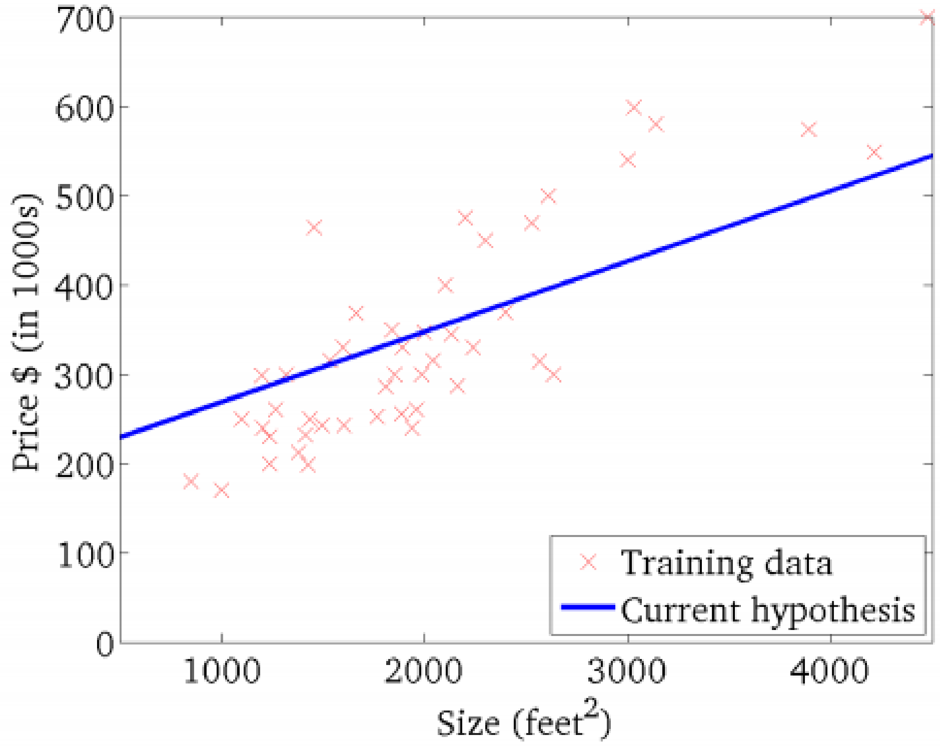
(function of the parameters θ_0, θ_1)



Gradient Descent Example -8

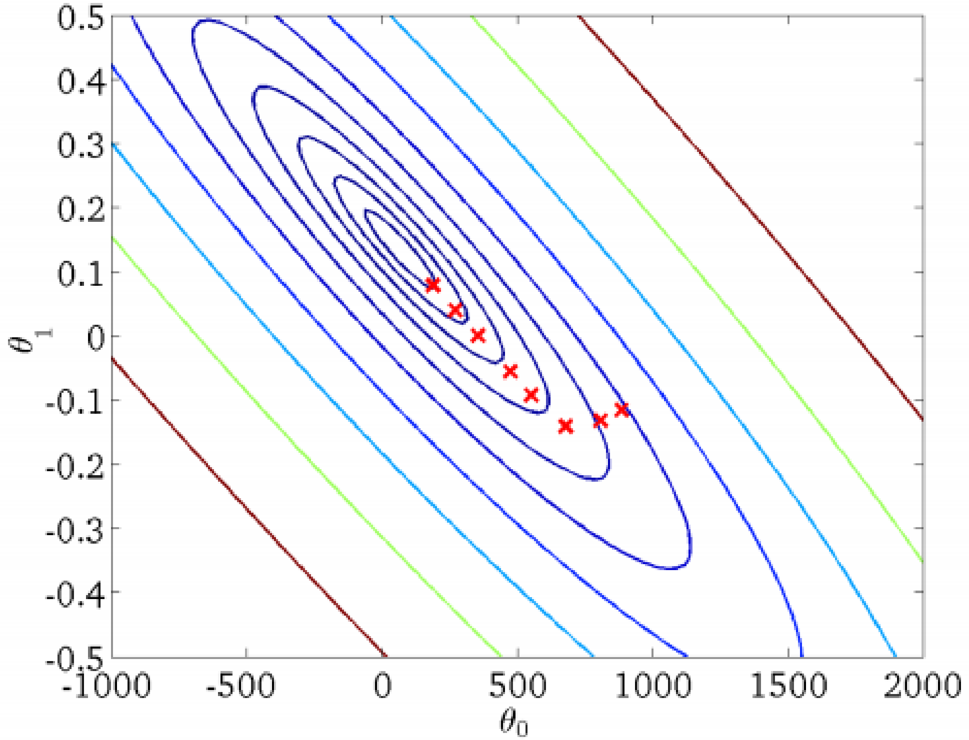
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

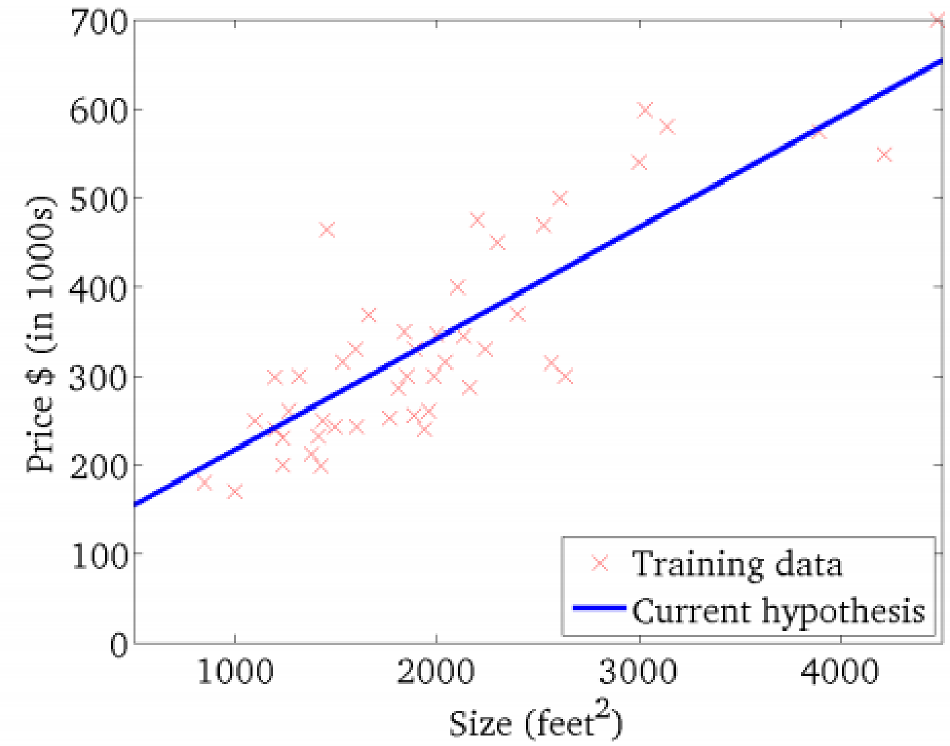
(function of the parameters θ_0, θ_1)



Gradient Descent Example -9

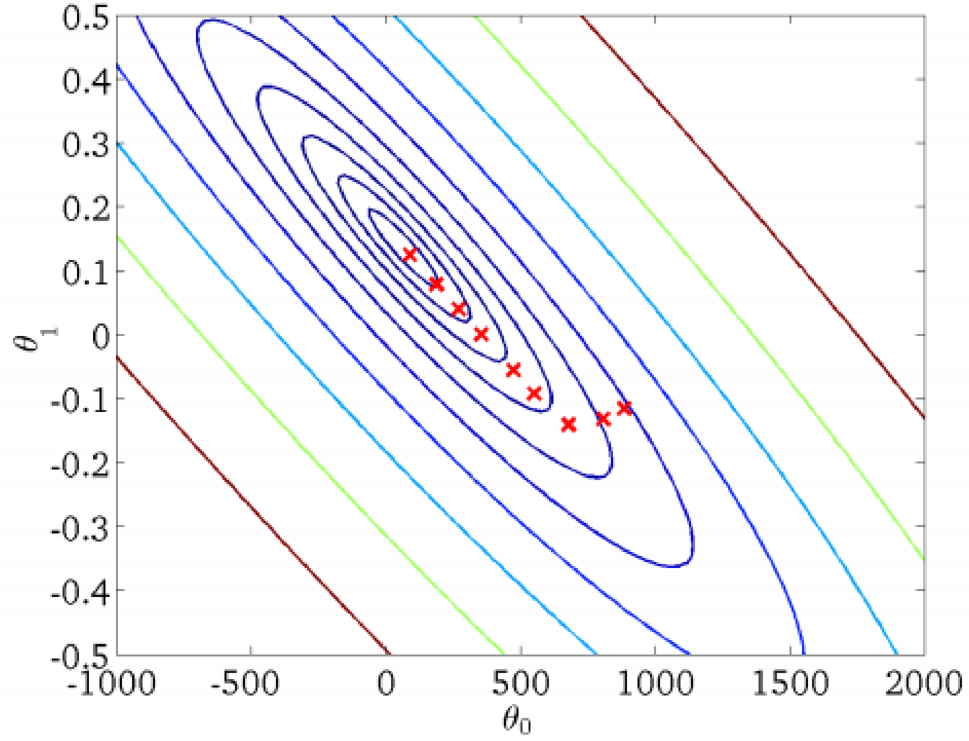
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Gradient Descent Mathematically

Determine gradients:

Coding Example:

- Let’s write some code!

Break

Multivariable Regression

- Suppose we have multiple inputs $x_1, \dots, x_D$. This is referred to as multivariable regression.
- This is no different than the single input case, just harder to visualize.

- Linear model:

$$\hat{y} = \sum_j \theta_j x_j + \theta_0$$

Implementation

- Computing the prediction using a **for loop**:

$$\hat{y} = \sum_j \theta_j x_j + \theta_0$$

```
y_pred = bias
for j in range (N):
    y_pred + = theta[j] * x[j]
```

- For-loops in Python are slow, so we **vectorize** algorithms by expressing them in terms of vectors and matrices.

$$\boldsymbol{\theta} = (\theta_1, \dots, \theta_D)^T \qquad \mathbf{x} = (x_1, \dots, x_D)^T$$

$$\hat{\boldsymbol{y}} = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x} + \theta_0$$

- Matrix/Vector multiplication is much faster:

```
y_pred = np.dot(theta, x) + bias
```

Why Vectorize?

- The equations, and the **code, will be simpler and more readable**. Gets rid of dummy variables/indices!

- Vectorized code is much faster
  - Cut down on Python interpreter overhead
  - Use highly optimized linear algebra libraries
  - **Matrix multiplication is very fast on a Graphical Processing Unit (GPU)**

Matrix of Data

- Vectorization requires that we organize all the training examples into the design matrix **X** with one row per training example, and all the targets into the target vector $\boldsymbol{y}$.

one feature across
all training examples

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \end{pmatrix} = \begin{pmatrix} 8 & 0 & 3 & 0 \\ 6 & -1 & 5 & 3 \\ 2 & 5 & -2 & 8 \end{pmatrix}$$

Sample 2

one training
example (vector)

Design Matrix

- Computing the predictions for the entire dataset:

$$\mathbf{X}\boldsymbol{\theta} + \theta_0\mathbf{1} = \begin{pmatrix} \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}^{(1)} + \theta_0 \\ \vdots \\ \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}^{(N)} + \theta_0 \end{pmatrix} = \begin{pmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(N)} \end{pmatrix} = \widehat{\boldsymbol{y}}$$

Loss

- Computing the squared error cost across the entire dataset:

$$\hat{\boldsymbol{y}} = \mathbf{X\theta} + \theta_0 \mathbf{1}$$

$$\mathcal{J} = \frac{1}{2N} ||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2$$

- In Python:

```
y_pred = np.dot(X, theta) + bias
cost = np.sum((y_pred - y) ** 2) / (2. * N)
```

Compute Gradients

- Partial derivatives: derivatives of a multivariate function with respect to one of its arguments

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

- To compute, take the single variable derivatives, pretending the other arguments are constant.

$$\hat{y} = \sum_j \theta_j x_j + \theta_0 \qquad \frac{\partial \hat{y}}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}\left[\sum_{j'} \theta_{j'} x_{j'} + \theta_0\right] \qquad \frac{\partial \hat{y}}{\partial \theta_0} = \frac{\partial}{\partial \theta_0}\left[\sum_{j'} \theta_{j'} x_{j'} + \theta_0\right]$$

$$= x_j \qquad\qquad = 1$$

Compute Gradients

- Chain rule for derivatives:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

$$\hat{y} = \sum_j \theta_j x_j + \theta_0$$

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\hat{y}} \frac{\partial \hat{y}}{\partial \theta_j}$$

$$= \frac{\mathrm{d}}{\mathrm{d}\hat{y}}\left[\frac{1}{2}(\hat{y} - y)^2\right] \cdot x_j$$

$$= (\hat{y} - y) \cdot x_j$$

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = \hat{y} - y$$

- Cost derivative (averages over data points):

$$\frac{\partial \mathcal{J}}{\partial \theta_j} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}^{(i)} - y^{(i)}){x_j}^{(i)}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_0} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}^{(i)} - y^{(i)})$$

Concise Notation

- The **bias** is often included inside the design matrix **X** for convenience as a column of ones.

$$\hat{y} = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_0$$
$$\hat{y} = \theta_0(1) + \theta_1 x_1 + \theta_2 x_2 + \cdots$$

$$\hat{y} = \sum_{j=0}^{N} \theta_j x_j$$

Parameter Update

- Cost derivative (averages over data points):

$$\boxed{\hat{y} = \sum_{j=0}^{N} \theta_j x_j} \longrightarrow \frac{\partial \mathcal{J}}{\partial \theta_j} = \frac{1}{N} \sum_{i=0}^{N} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$$

only change starting index and design matrix **X**

- Parameter Update:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial \mathcal{J}(\theta_j)}{\partial \theta_j}$$

Vectorize the update

- We can also vectorize this gradient computation:

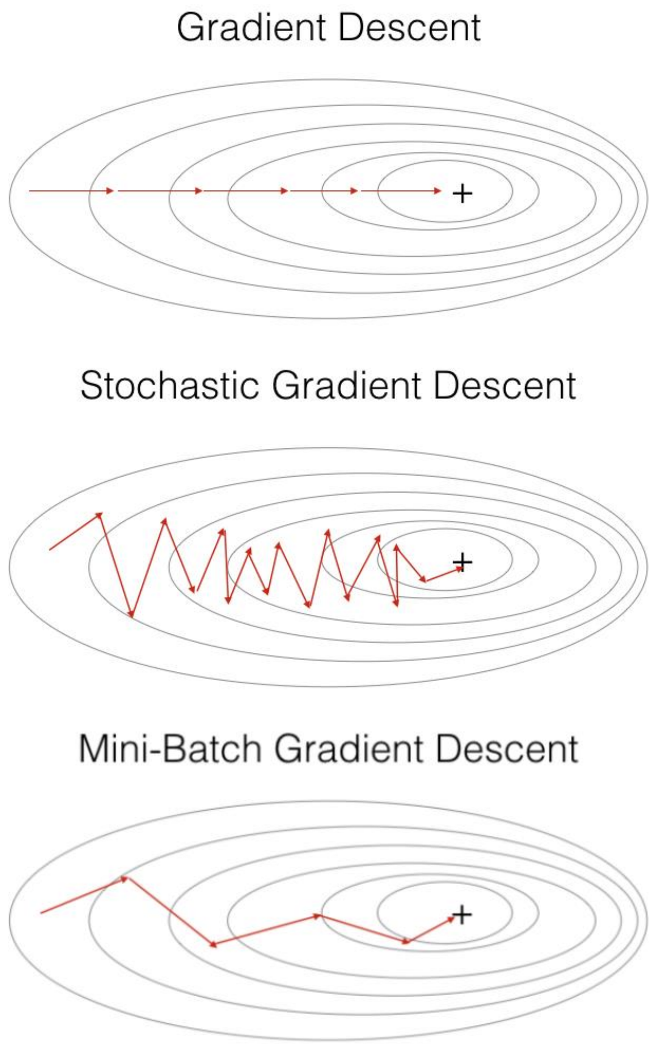
$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} = (\mathbf{y} - \hat{\mathbf{y}})^T \mathbf{X} \qquad\qquad \hat{\boldsymbol{y}} = \mathbf{X}\boldsymbol{\theta}$$

- and parameter update:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}}$$

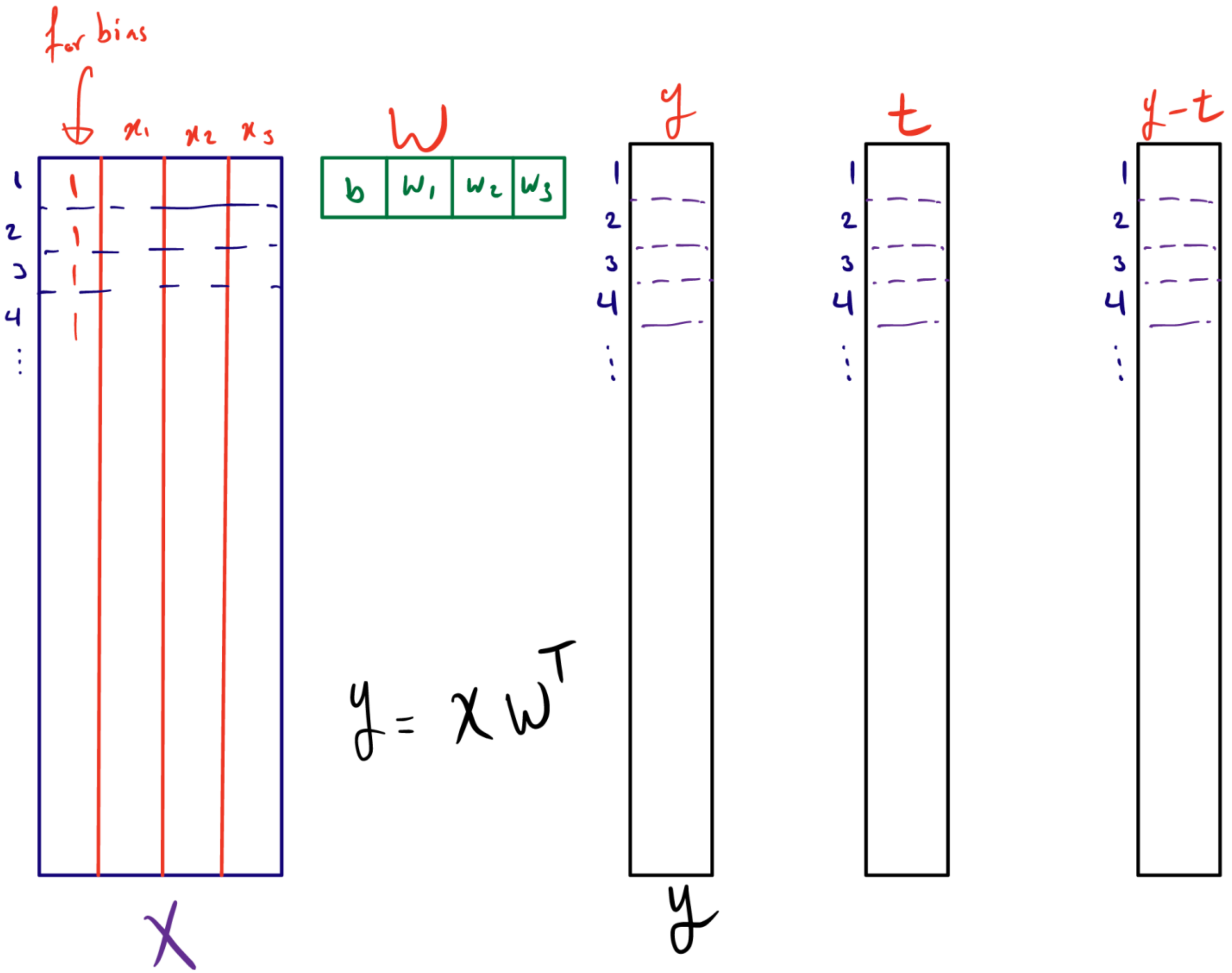
Batching and convergence

- Iteration:
  - Each time we update the weights is called an iteration
- Epoch:
  - Each time the model sees (learns) the whole dataset.
- Full batch GD:
  - Whole dataset in one batch.
  - One epoch is one iteration.
- Mini batch GD:
  - Break dataset to k smaller batches (mini batch).
  - One epoch takes k iterations.
- Stochastic GD:
  - Each of the n samples is a batch.
  - One epoch takes n iterations.



ref

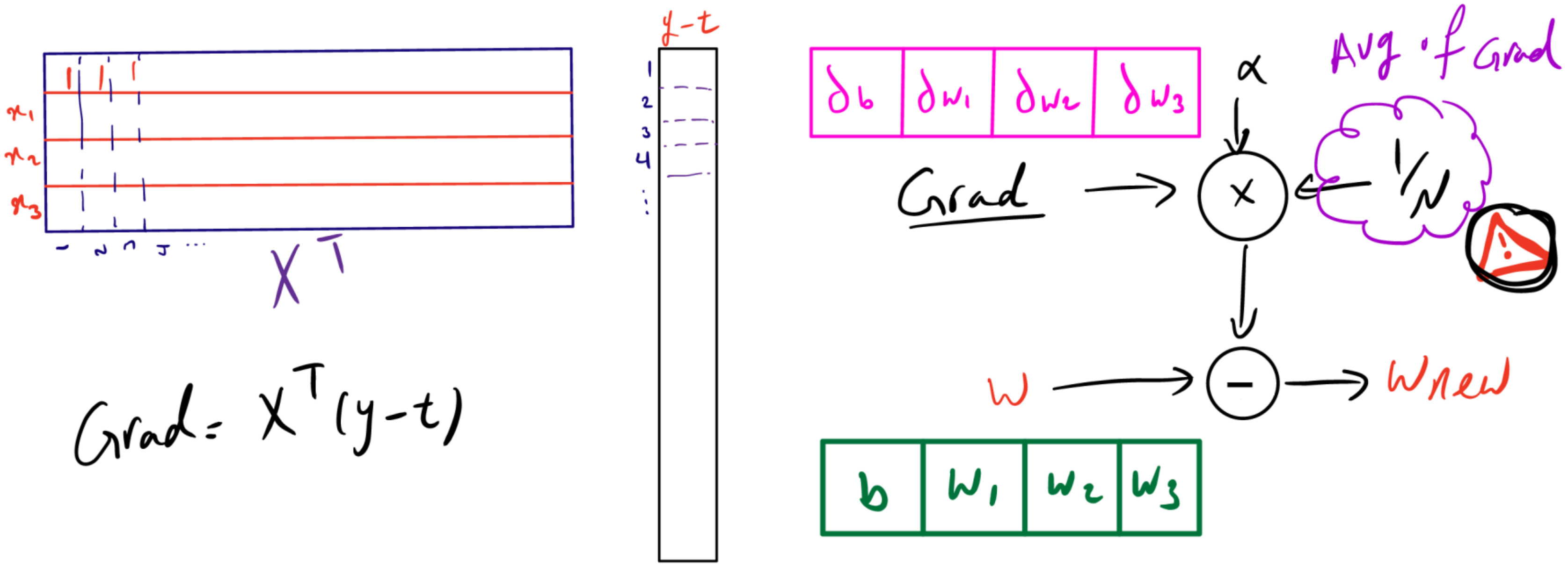
Full Batch -1



$$\frac{\partial J}{\partial w} = \frac{1}{N}[\sum_{i=0}^{N} x^{(i)}(y^{(i)} - t^{(i)})]$$

$$w_j = w_j - \alpha \times \frac{\partial J}{\partial w}$$

Full Batch -2



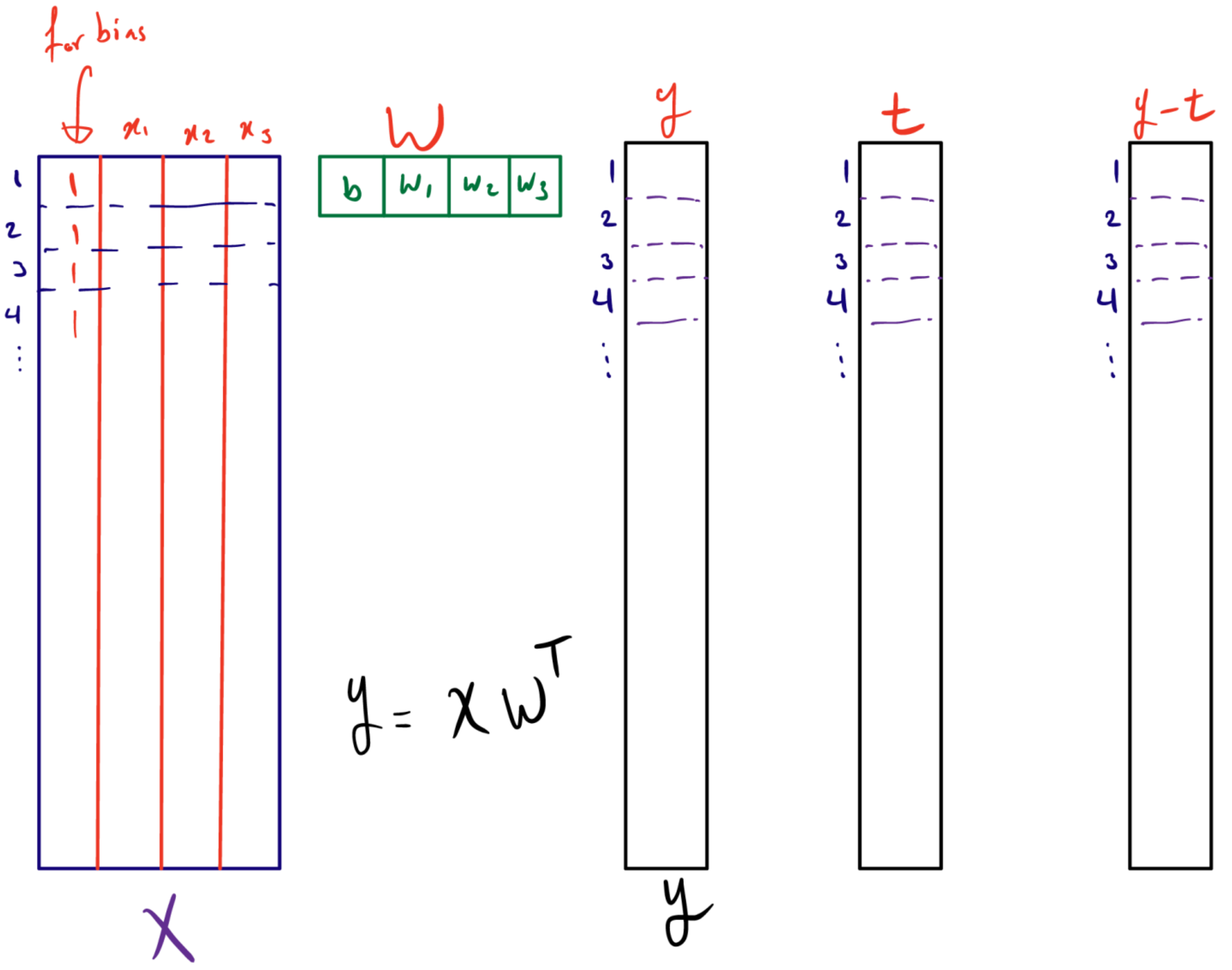
Grad = $X^T(y-t)$

1 iteration Per Epoch

$$\frac{\partial J}{\partial w} = \frac{1}{N}\left[\sum_{i=0}^{N} x^{(i)}\left(y^{(i)} - t^{(i)}\right)\right]$$

$$w_j = w_j - \alpha \times \frac{\partial J}{\partial w}$$

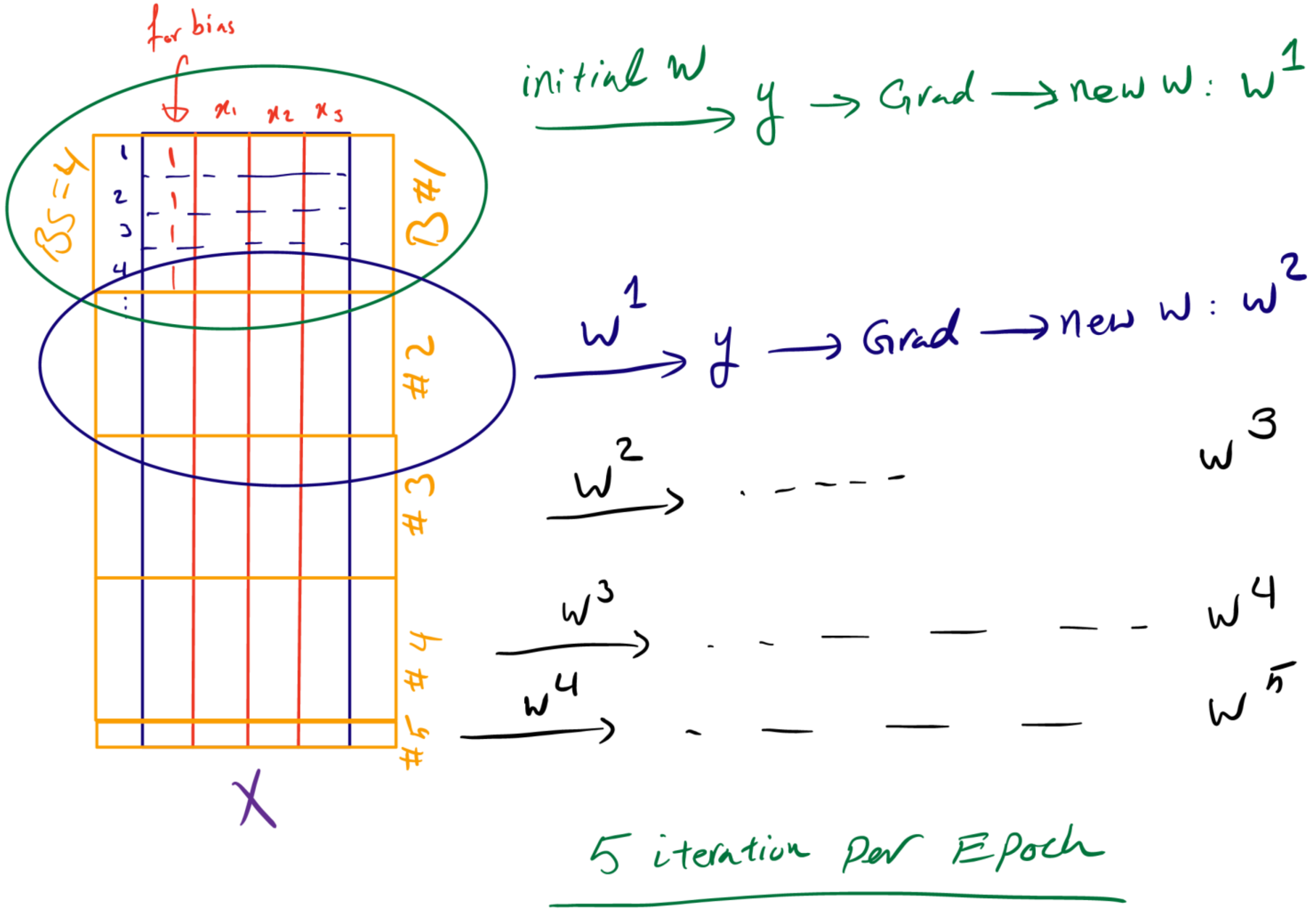
Full Batch -3



$$\frac{\partial J}{\partial w} = \frac{1}{N}[\sum_{i=0}^{N} x^{(i)}(y^{(i)} - t^{(i)})]$$

$$w_j = w_j - \alpha \times \frac{\partial J}{\partial w}$$

Mini Batch


for bias
x1 x2 x3
BS=4
B#1
#2
#3
#4
#5
X
initial W → y → Grad → new W : W^1
W^1 → y → Grad → new W : W^2
W^2 → W^3
W^3 → W^4
W^4 → W^5
5 iteration per Epoch

Break

“alternative approach that allows us to model our problem using probability”

Maximal Likelihood Estimation

Readings:

- **Chapter 8.3 MML Textbook**

Maximum Likelihood Estimation (MLE)

- A frequentist approach for estimating the parameters of a model given some observed data.
- Uses **probability distributions to model the uncertainty**.

- The general approach for using MLE is:
  1. Observe some data.
  2. Write down a **model for how we believe the data was generated**.
  3. **Set the parameters** of our model to values which **maximize the likelihood of the parameters given the data**.

Models

- A model is a formal representation of our beliefs, assumptions, and simplifications surrounding some event or process.

- **Example:** Building a model for flipping a coin
  - The coin has two faces and an edge
  - The faces have different designs
  - The coin can sit on either face or the edge
  - The weight of the coin
  - The diameter and thickness of the coin

Models Con’t

- **What assumptions can we make?**
  - The different designs probably cause the coin’s center of mass to slightly favor one side over another.
  - There’s no way to measure the force or angle exerted on the coin when it’s flipped.

Models Con’t

- **First attempt at a model** (without simplifications):
  - The initial position of the coin is drawn from a Bernoulli distribution (i.e. flipper’s preference).
  - The force exerted on the coin is drawn from an exponential distribution.
  - The angle in which the force is exerted is drawn from a truncated normal distribution on the interval [-π, π].
  - The center of mass of the coin is at some coordinate (x,y,z) in a system (center of the coin is the origin).
  - The force of gravity is … OK, I think you get the picture.

Models Con't

- The real world can be complicated. **A simplified model can often do just as well or better!**
- Let's make a simplified model:
  - The outcome of the flip is drawn from a **Bernoulli distribution** with the probability of heads p, and the probability of tails (1-p).
- **Our simplified model only has a single parameter!**

MLE Example: Coin Flips -1

- **Step 1:** To start let us assume we observed the following sequence of coin flips:

- X = heads, heads, tails, heads, tails, tails, tails, heads, tails, tails

MLE Example: Coin Flips -2

- **Step 2a:** Write down a model for how we believe the data was generated (we'll start with a single flip):

$$L_x(p) = P(x|p) = p^x(1-p)^{1-x}$$

likelihood function
(for a single flip)

Recall that we're modeling the outcome of a coin flip by a Bernoulli distribution, where the parameter p represents the probability of getting a heads (x=1).

MLE Example: Coin Flips -3

- **Step 2b:** Write down a model for how we believe the data was generated:

$$L_X(p) = P(X|p) = \prod_{x \in X} p^x (1-p)^{1-x}$$

likelihood function
(for all flips)

Since the **coin flips are iid**, we can write the likelihood of seeing a particular sequence as the **product of each individual flip**

MLE Example: Coin Flips -4

- **Step 2c:** We can generalize this further…

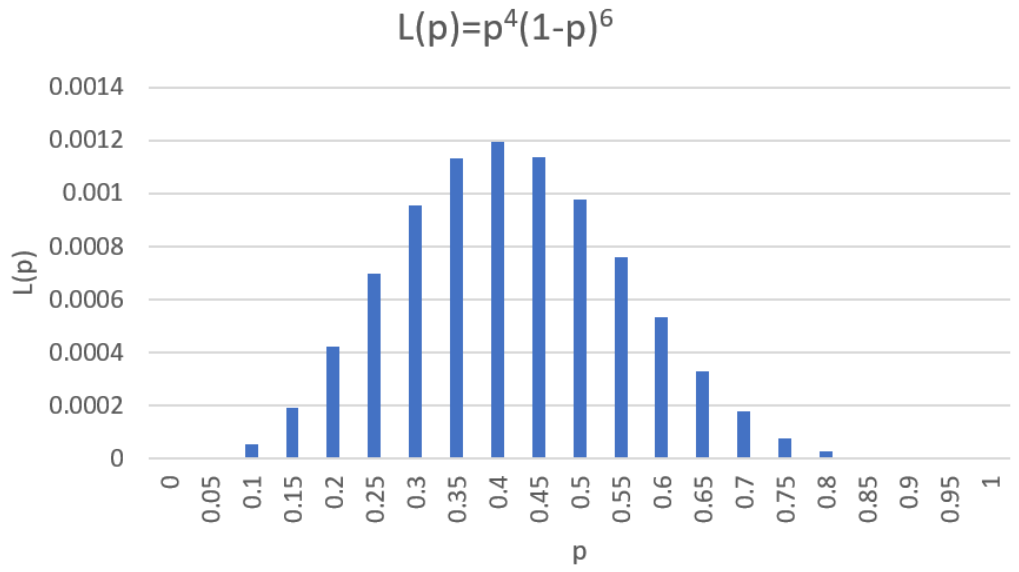
$$L(p) = p^h \cdot (1 - p)^{n-h}$$

where n is the number of coin flips with h the number of heads that were recorded

In our data (X):

We had observed

h=4, n=10



MLE Example: Coin Flips -3

- Step 2d: We can generalize this further…

$$L(p) = p^h \cdot (1 - p)^{n-h}$$

where n is the number of coin flips with h the number of heads that were recorded

- Log-Likelihood Function

$$l(p) = h \cdot \log(p) + (n - h) \cdot \log(1 - p)$$

MLE Example: Coin Flips -4

- **Step 3: Set the parameters** of our model to values which **maximize the likelihood of the parameters given the data**

$$l(p) = h \cdot \log(p) + (n - h) \cdot \log(1 - p)$$

- Maximum => take derivative of function *l(p)* with respect to p

$$l'(p) = \frac{h}{p} - \frac{n-h}{1-p}$$

Setting $l'(p)$ to 0 gives us:

$$\boldsymbol{p = h/n}$$

Linear Regression
(Maximum Likelihood Estimation)

Readings:

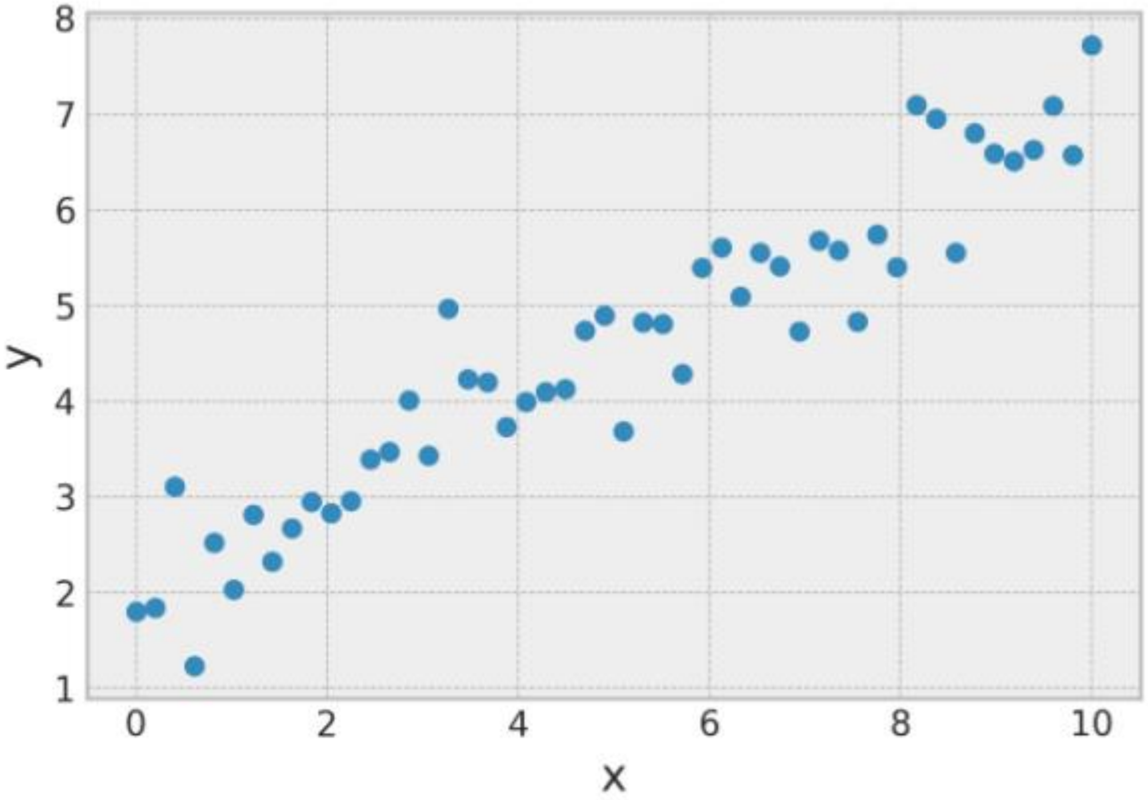
- **Chapter 9.1-2 MML Textbook**

Linear Model

- We'd like to build a model of the data in order to predict new values of y given x.

- The data almost looks like a line, so let's start with a linear model.

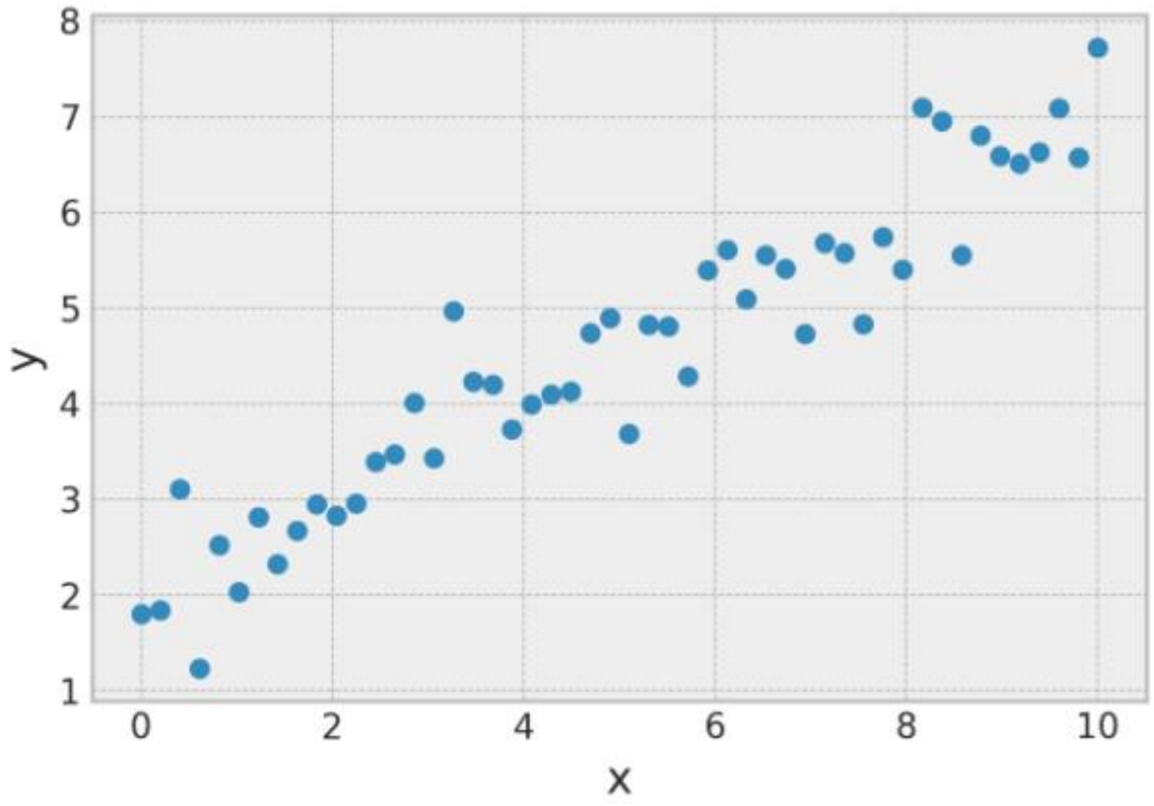
$$y = \theta_1 x + \theta_0$$



Linear Model

- Q: How do we account for the deviations we're observing?

- A: Imagine we're using a sensor to collect this data. Most sensors have some amount of error in their measurements. We can think of the **deviations from our model as being caused by an error prone sensor.**



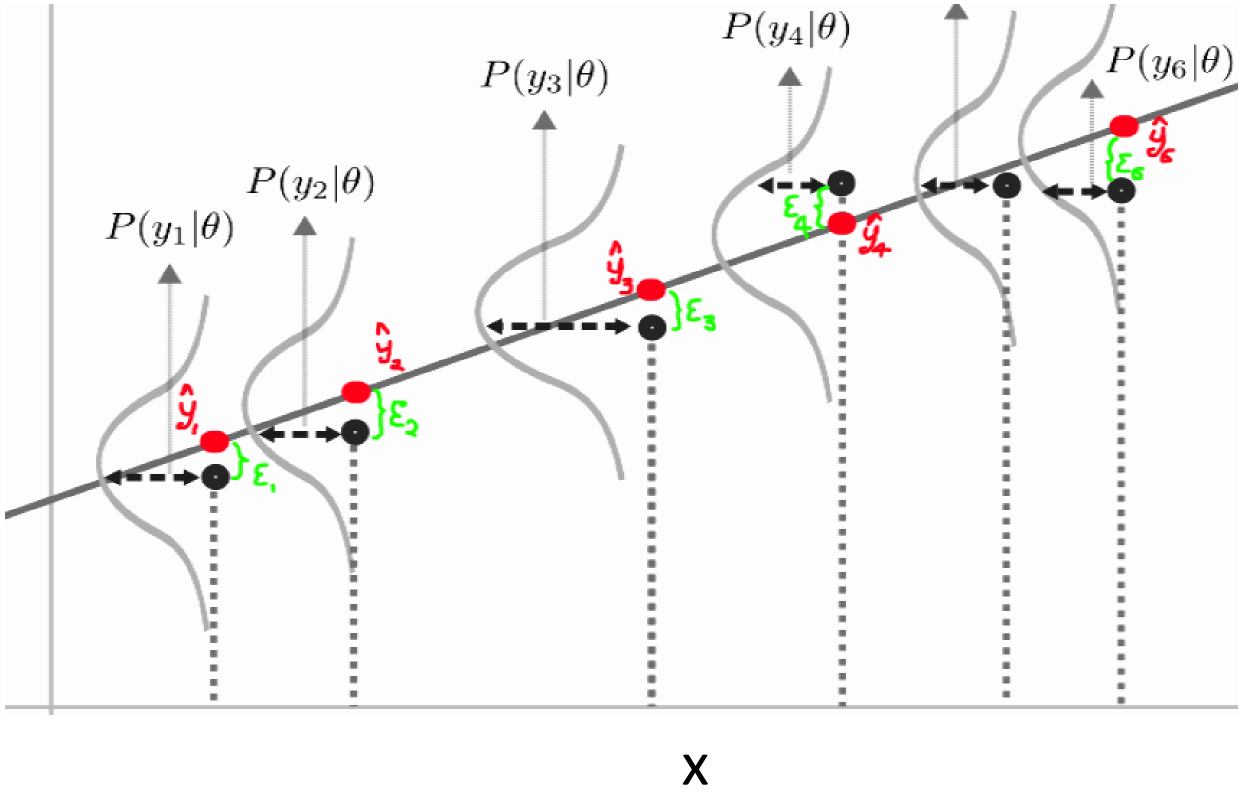
Linear Model

- common to model the error as being drawn from a Gaussian distribution with mean zero and variance σ².

$$\epsilon \sim N(0, \sigma^2)$$

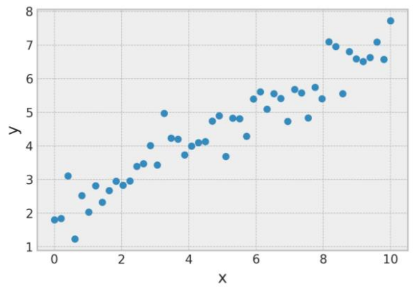
$$y = \theta_1 x + \theta_0 + \epsilon$$

$$y \sim N(\overset{\hat{y}}{\boxed{\theta_1 x + \theta_0}}, \sigma^2)$$



MLE Example: Linear Regression -1

- **Step 1:** Obtain some sample data:



$$\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$$

- **Step 2a:** Write down a model for how we believe the data was generated (we'll start with a single sample):

$$f(y|x, \theta_0, \theta_1, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y-(\theta_1 x+\theta_0))^2}{2\sigma^2}}$$

likelihood function
(for a single point)

This time we are modeling the outcome with a Gaussian distribution.

MLE Example: Linear Regression -2

- **Step 2b:** Write down a model for how we believe the data was generated:

$$\mathcal{L}(\theta_0, \theta_1, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \prod_{n=1}^{N} e^{\frac{-(y_n-(\theta_1 x_n+\theta_0))^2}{2\sigma^2}}$$

likelihood function
(for all points)

since each point is **iid**, we can write the likelihood function with respect to all the observed points as a product of each point.

$$\mathcal{L}(\theta_0, \theta_1, \sigma^2) = \log[\frac{1}{\sqrt{2\pi\sigma^2}} \prod_{n=1}^{N} e^{\frac{-(y_n-(\theta_1 x_n+\theta_0))^2}{2\sigma^2}}]$$

MLE Example: Linear Regression -3

- **Step 2c: Rewrite in terms of log-likelihood**

Recall $\log(ab) = \log(a) + \log(b)$

$$\mathcal{L}(\theta_0, \theta_1, \sigma^2) = \log[\frac{1}{\sqrt{2\pi\sigma^2}} \prod_{n=1}^{N} e^{\frac{-(y_n-(\theta_1 x_n+\theta_0))^2}{2\sigma^2}}]$$

$$= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \sum_{n=1}^{N} \frac{-(y_n - (\theta_1 x_n + \theta_0))^2}{2\sigma^2}$$

$$= \log(1) - \log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y - (\theta_1 x_n + \theta_0))^2$$

$$= -\log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y - (\theta_1 x_n + \theta_0))^2$$

MLE Example: Linear Regression -4

- **Step 2d: Rewrite in terms of log-likelihood**

$$\mathcal{L}(\theta_0, \theta_1, \sigma^2) = -\log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y - (\theta_1 x_n + \theta_0))^2$$

- To clean things up a bit more, let's write the output of our line as a single value:

$$\hat{y} = \theta_1 x + \theta_0$$

- Now our log-likelihood can be written as:

$$\mathcal{L}(\theta_0, \theta_1, \sigma^2) = -\log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y - \hat{y}_n)^2$$

multiply by -1 to make it a negative log-likelihood

Sum of Squared Errors

- Step 3: **Parameters which maximize the log-likelihood are the same as the ones that minimize the negative log-likelihood.**

$$\mathcal{L}(\theta_0, \theta_1, \sigma^2) = \log\left(\sqrt{2\pi\sigma^2}\right) + \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y - \hat{y}_n)^2$$

- Removing any constant's which don't include our θs won't alter the solution. We can simplify what we're trying to minimize:

$$\sum_{n=1}^{N}(y - \hat{y}_n)^2$$

Sum of Squared Errors

Solving for Parameters

- The maximum likelihood estimates for our slope and intercept (parameters) can be found by minimizing the sum of squared errors.

$$\mathcal{L}(\theta) = \sum_{n=1}^{N} (y_n - \hat{y}_n)^2$$

$$= \sum_{n=1}^{N} y_n - (\theta_1 x_n + \theta_0))^2$$

$$= (\boldsymbol{y} - \boldsymbol{X\theta})^T (\boldsymbol{y} - \boldsymbol{X\theta})$$

$$= \|\boldsymbol{y} - \boldsymbol{X\theta}\|^2$$

vector notation

This is the same as our **empirical risk minimization** where we **assumed a sum of squared error loss function**!

Solving for the Parameters (θ)

- We start by taking the partial derivative with respect to parameters **θ**:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left((\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})\right)$$

$$= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left(\boldsymbol{y}^\top\boldsymbol{y} - 2\boldsymbol{y}^\top\boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\theta}\right)$$

$$\boxed{= -\boldsymbol{y}^\top\boldsymbol{X} + \boldsymbol{\theta}^\top\boldsymbol{X}^\top\boldsymbol{X}}$$

Q: How do we find our parameters (θ)?

Solving for the Parameters (θ)

- At this point we have two options for finding the parameters **θ:**

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = -\boldsymbol{y}^\top \boldsymbol{X} + \boldsymbol{\theta}^\top \boldsymbol{X}^\top \boldsymbol{X}$$

(1) iterative solution for θ:

$$\boxed{\theta_i \leftarrow \theta_i - \alpha \frac{\partial \mathcal{L}(\theta_i)}{\partial \theta_i}}$$

(2) direct solution for θ:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = \boldsymbol{0}$$

$$\boldsymbol{\theta}_{\mathrm{ML}}^\top \boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{y}^\top \boldsymbol{X}$$

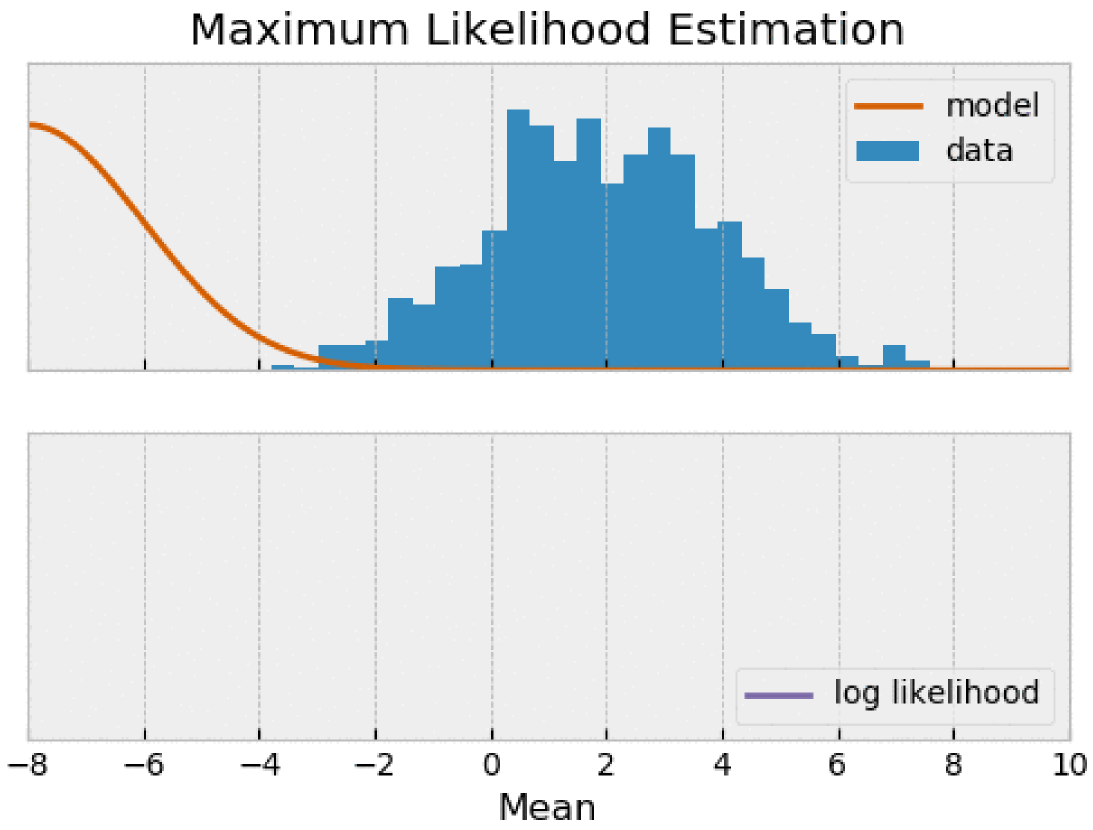
$$\boldsymbol{\theta}_{\mathrm{ML}}^\top = \boldsymbol{y}^\top \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$$

$$\boldsymbol{\theta}_{\mathrm{ML}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

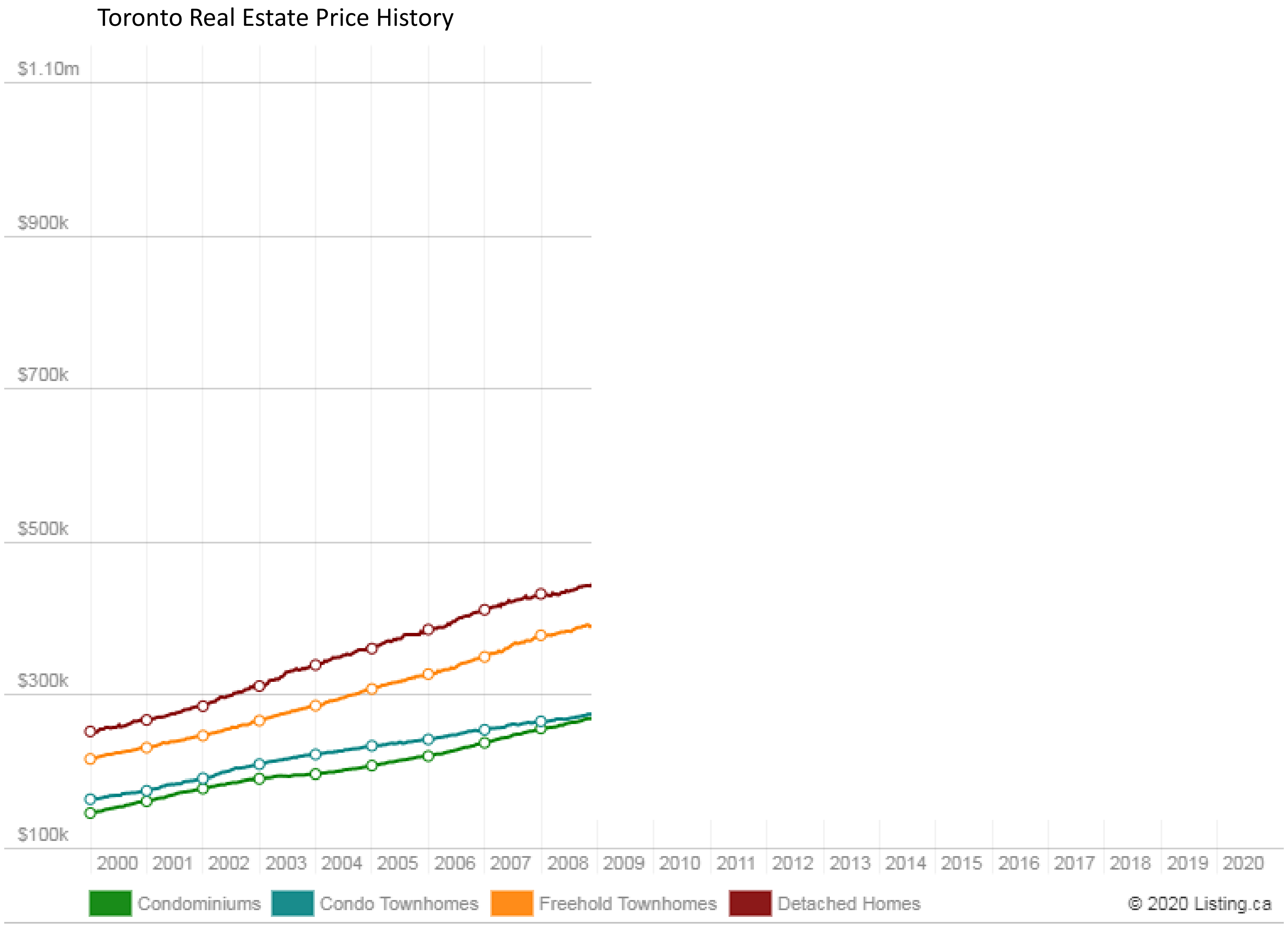
Summary: Maximum Likelihood

- Imagine we have some data generated from a Gaussian distribution with a known variance, but we don't know the mean.

- You can think of MLE as taking the Gaussian, sliding it over all possible means, and choosing the mean which causes the model to fit the data best.

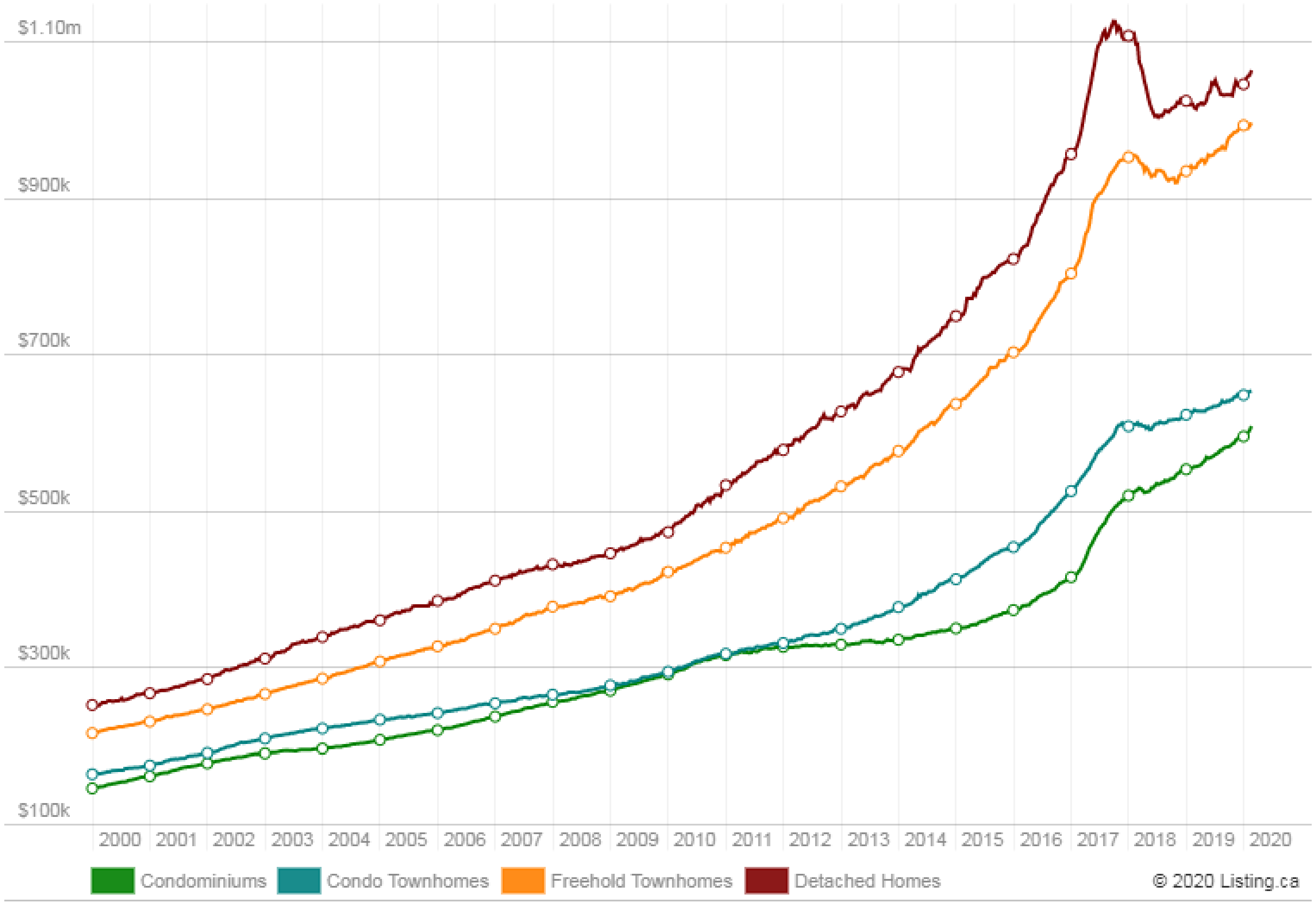


Nonlinear Regression



Nonlinear Regression

Toronto Real Estate Price History



Next Time

- Week 9 Q/A Support Session
  - Project 4 – Linear Regression
- Reading Assignment
- Week 10 Lecture – Nonlinear Regression
  - Polynomial Regression
  - Optimization and Convexity
  - Regularization
  - Classification
  - Neural Networks