Linear Regression

Dr Mehrdad Ghaziasgar

Content – Part 1

- Simple (Univariate) Linear Regression
    - Model Representation
    - Cost Function
- Gradient Descent
    - Formulation
    - Algorithm
    - Application to Linear Regression

Simple Linear Regression
Model Representation

- Simple (Univariate) Linear Regression
    - Model Representation
    - Cost Function
- Gradient Descent
    - Formulation
    - Algorithm
    - Application to Linear Regression

Model Representation - Introduction

- Running example: predicting housing prices

- We obtain information about:
    - Houses e.g. size (sq. metres), no. of bedrooms, no. of bathrooms, no. of garages, frontage (metres), no. of storeys, garden size (sq. metres) etc. etc.
    - Price that each house last (recently) sold for

- Use a learning algorithm to build a model to predict housing prices

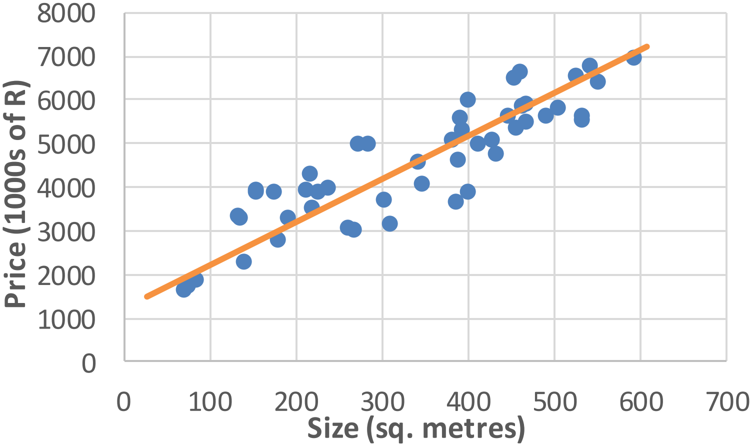
Model Representation - Introduction

Size (sq. m) (x)	Price (1000s of R) (y)
460	6639
70	1681
155	3969
429	5095
...	...

Training set of house sizes and prices

- We obtain the data above

Model Representation - Introduction



Supervised learning:

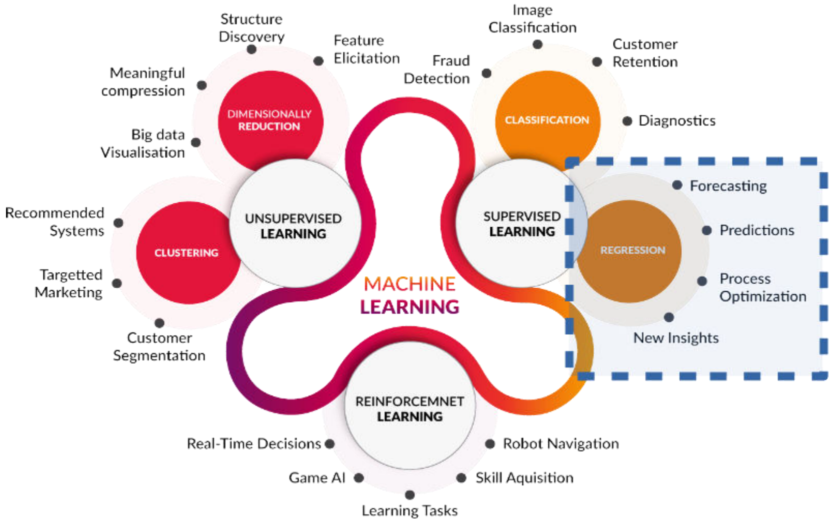
- Given the "right" answer for each data point

Regression:

- Predict a real-valued (continuous) output e.g. price

(**Side note:** the other branch of supervised learning is classification i.e. predict discrete output e.g. category)

Model Representation - Introduction



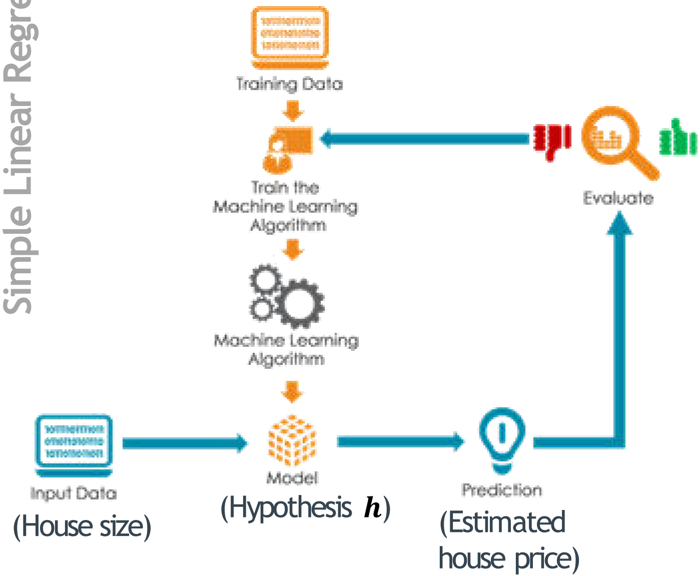
Model Representation - Notation

Size (sq. m) (x)	Price (1000s of R) (y)
460	6639
70	1681
155	3969
429	5095
…	…

Training set of house sizes and prices

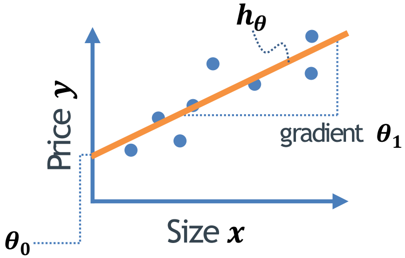
- $\boldsymbol{x}$: Input variable; "features" used to make predictions e.g. 460 sq. metres
- $\boldsymbol{y}$: Output variable; "target" output e.g. R 6639000
- $\boldsymbol{m}$: Number of examples in the training set
- $(\boldsymbol{x}, \boldsymbol{y})$: A specific sample in the training set
- $(\boldsymbol{x}^{(\boldsymbol{i})}, \boldsymbol{y}^{(\boldsymbol{i})})$: the $\boldsymbol{i}$th training example e.g. $\boldsymbol{x}^{(1)}$= 460, $\boldsymbol{x}^{(2)}$= 70… ; $\boldsymbol{y}^{(1)}$= 6639, $\boldsymbol{y}^{(2)}$= 1681

Model Representation – Project Design



- h: Mapping of x values (sizes) onto y values (prices)

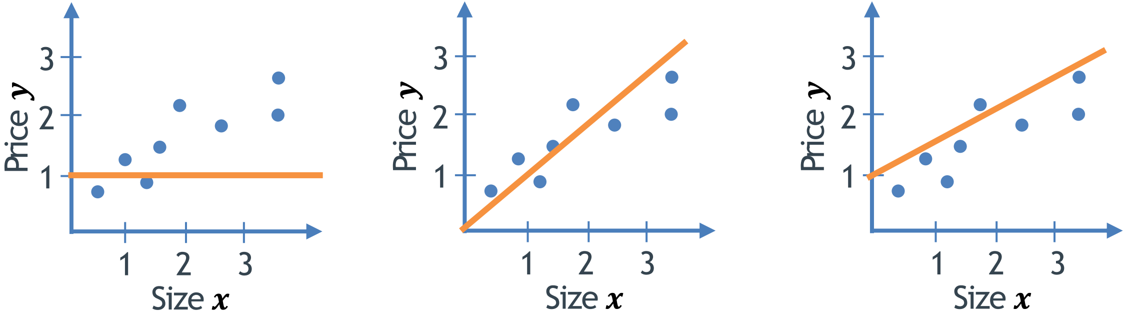
What does h look like?



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

- Equation of a straight line with gradient θ_1 and y-intercept θ_0

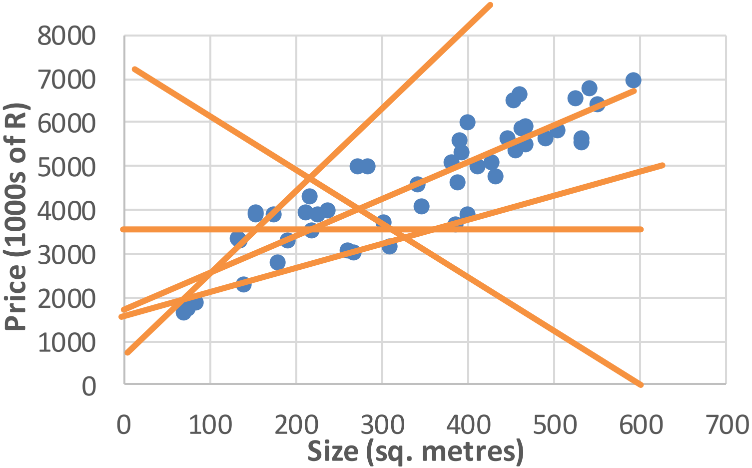
Model Representation - Deciphering $h_\theta(x)$



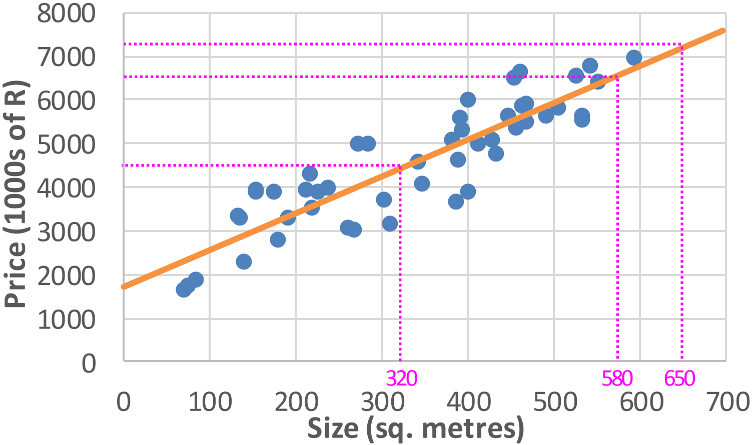
$\theta_0 = 1;\ \theta_1 = 0$ | $\theta_0 = 0;\ \theta_1 = 1$ | $\theta_0 = 1;\ \theta_1 = 0.5$

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

Model Representation - Deciphering $h_\theta(x)$



Model Representation - Using $h_\theta(x)$ to Make Predictions



Given the best fit θ_0 = 1740; θ_1 = 8.4, predict:

- $h_\theta(320)$
  $= 1740 + 8.4(320)$
  $= 4428$
- $h_\theta(580)$
  $= 1740 + 8.4(580)$
  $= 6612$
- $h_\theta(650)$
  $= 1740 + 8.4(650)$
  $= 7200$

Model Representation - Summary

- We need to fit a line (h_θ) onto the housing data (price versus size)
- The line is determined by the parameters θ_0 and θ_1
- Once we have an appropriate line, we can make price predictions on any unknown values in future
- The values of θ_0 and θ_1 will determine how well the line fits the training data
- Very important: the parameter values also determine how accurate future price predictions based on size may be
- This is a simple / univariate linear regression problem

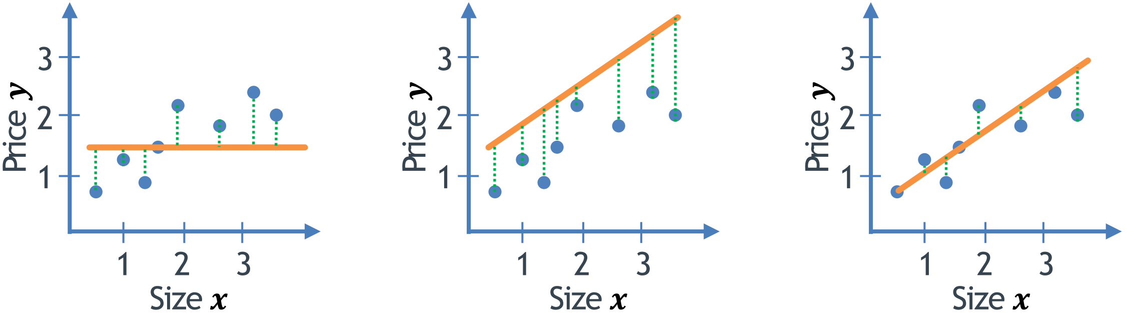
- Golden question: How do we determine the best θ_0 and θ_1 to use??

Simple Linear Regression

Cost Function

- Simple (Univariate) Linear Regression
  - Model Representation
  - Cost Function
- Gradient Descent
  - Formulation
  - Algorithm
  - Application to Linear Regression

Cost Function - Formulation



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

$h_\theta(x) = 1.5$

$\theta_0 = 1$; $\theta_1 = 1$

$h_\theta(x) = 1 + x$

$\theta_0 = 0.5$; $\theta_1 = 0.5$

$h_\theta(x) = 0.5 + 0.5x$

- Given the points in the training set (x, y), choose θ_0 and θ_1 such that the line's predicted prices ($h_\theta(x)$) are "close" to the each actual price (y)

Cost Function - Formulation

- Given the points in the training set $(\boldsymbol{x}, \boldsymbol{y})$, choose $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$ such that the line's predicted prices $(\boldsymbol{h_\theta}(\boldsymbol{x}))$ are "close" to the each actual price $(\boldsymbol{y})$

- Compute the distance between $\boldsymbol{h_\theta}(\boldsymbol{x})$ and $\boldsymbol{y}$ for all the data points:

$$\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)} \text{ for all } (\boldsymbol{i}) \text{ from 1 to } \boldsymbol{m}$$

- Add/sum the distances up

$$\sum_{i=1}^{m}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)}\right)$$

Cost Function - Formulation

- Similar to adding/summing the **squares** of the distances up

$$\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

- Similar to adding/summing the squares of the distances up and dividing by the number of points m to get the total **average** square distance

$$\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

- Similar to multiplying by a half to get **half** the total average square distance

$$\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

The smaller this is, the better the line fits
The larger this is, the worse the line fits
This is the "cost function"

Cost Function - Formulation

- Similar to multiplying by a half to get **half** the total average square distance

$$\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

The smaller this is, the better the line fits
The larger this is, the worse the line fits
This is the “cost function”

- Remember that:

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

- So actually:

$$\frac{1}{2m}\sum_{i=1}^{m}\left(\theta_0+\theta_1 x^{(i)}-y^{(i)}\right)^2$$

- So the cost function (how well the line fits the data) depends directly on θ_0 and θ_1
    - In mathematical terms: it is a “function” of θ_0 and θ_1

Cost Function - Formulation

- So the cost function (how well the line fits the data) depends directly on $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$
    - In mathematical terms: it is a "function" of $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$
    - Conventionally, the function is denoted as $\boldsymbol{J}$
    - Also known as the squared error function

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

or

$$J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1}) = \frac{1}{2m} \sum_{\boldsymbol{i}=1}^{\boldsymbol{m}} \left(\boldsymbol{\theta_0} + \boldsymbol{\theta_1} \boldsymbol{x}^{(\boldsymbol{i})} - \boldsymbol{y}^{(\boldsymbol{i})} \right)^2$$

Cost Function - Formulation

- To choose the best $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$ for a given training set, find $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$ for which the cost function $\boldsymbol{J}$ has the smallest value i.e. minimize $\boldsymbol{J}$
- Mathematically:

$$\underset{\boldsymbol{\theta_0},\boldsymbol{\theta_1}}{\text{minimize}} \; J(\boldsymbol{\theta_0},\boldsymbol{\theta_1})$$

or

$$\underset{\boldsymbol{\theta_0},\boldsymbol{\theta_1}}{\text{minimize}} \; \frac{1}{2m}\sum_{\boldsymbol{i}=1}^{\boldsymbol{m}}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(\boldsymbol{i})}\right)-\boldsymbol{y}^{(\boldsymbol{i})}\right)^2$$

or

$$\underset{\boldsymbol{\theta_0},\boldsymbol{\theta_1}}{\text{minimize}} \; \frac{1}{2m}\sum_{\boldsymbol{i}=1}^{\boldsymbol{m}}\left(\boldsymbol{\theta_0}+\boldsymbol{\theta_1}\boldsymbol{x}^{(\boldsymbol{i})}-\boldsymbol{y}^{(\boldsymbol{i})}\right)^2$$

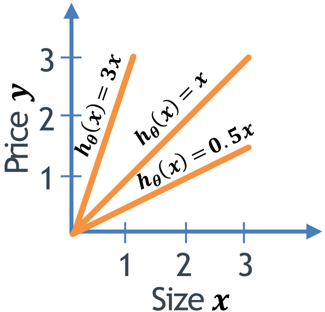
Cost Function - Deciphering the Cost Function J

- To understand what minimizing the cost function means, let's assume:

$$\boldsymbol{\theta_0} = 0$$

- $\boldsymbol{h_\theta}(\boldsymbol{x})$ is a line that passes through the origin (0,0), with changing gradients so:

$$\boldsymbol{h_\theta}(\boldsymbol{x}) = \boldsymbol{\theta_1 x}$$



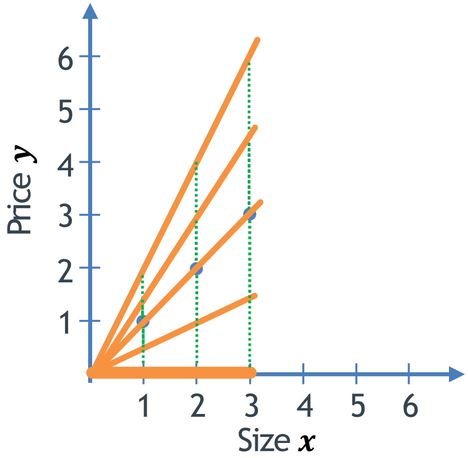
- Therefore the cost function now only depends on $\boldsymbol{\theta_1}$:

$$\underset{\boldsymbol{\theta_1}}{\text{minimize}} \;\; J(\boldsymbol{\theta_1})$$

$$\underset{\boldsymbol{\theta_1}}{\text{minimize}} \;\; \frac{1}{2m}\sum_{\boldsymbol{i}=1}^{\boldsymbol{m}}\left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)}\right)^2$$

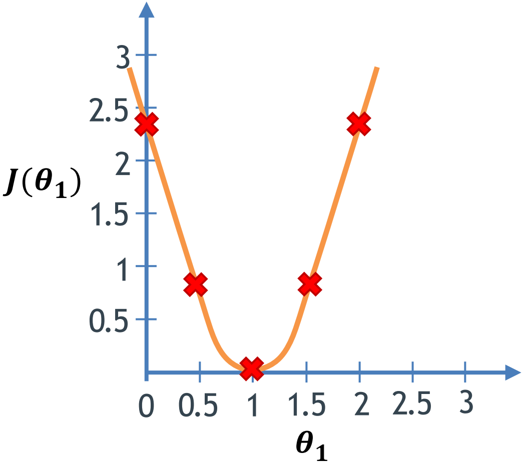
$$\underset{\boldsymbol{\theta_1}}{\text{minimize}} \;\; \frac{1}{2m}\sum_{\boldsymbol{i}=1}^{\boldsymbol{m}}\left(\boldsymbol{\theta_1 x}^{(i)} - \boldsymbol{y}^{(i)}\right)^2$$

Cost Function - Deciphering the Cost Function J



$\theta_1 = 0.5$

$h_\theta(x) = 0.5x \quad (\theta_0 = 0)$



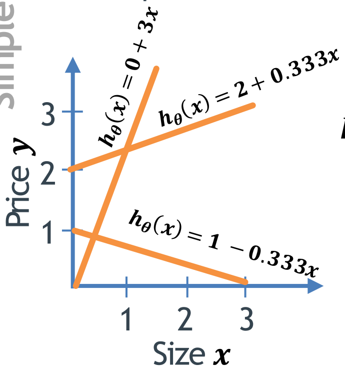
$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(\theta_1 x^{(i)} - y^{(i)}\right)^2$$

Cost Function - Deciphering the Cost Function J With Two Params

- To understand what minimizing the cost function means, now let's take the original cost function :

$$J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$$

- An arbitrary line with any gradient and y-intercept so:



$$h_\theta(x) = \boldsymbol{\theta_0} + \boldsymbol{\theta_1} x$$

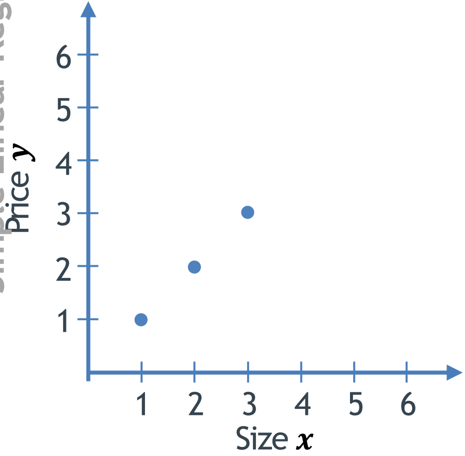
- The cost function depends on both $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$:

$$\underset{\theta_0,\theta_1}{\text{minimize}} \; J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$$

$$\underset{\theta_0,\theta_1}{\text{minimize}} \; \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

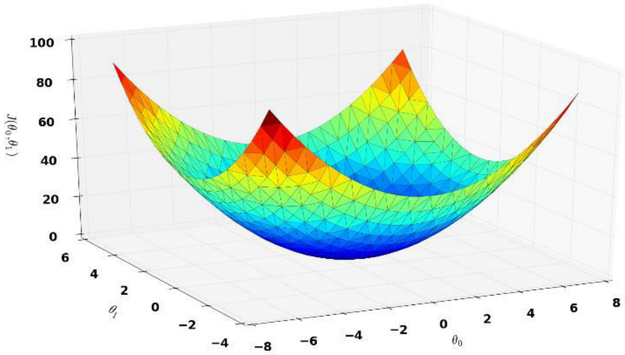
$$\underset{\theta_0,\theta_1}{\text{minimize}} \; \frac{1}{2m}\sum_{i=1}^{m}\left(\boldsymbol{\theta_0} + \boldsymbol{\theta_1} x^{(i)} - y^{(i)}\right)^2$$

Cost Function - Deciphering the Cost Function J With Two Params



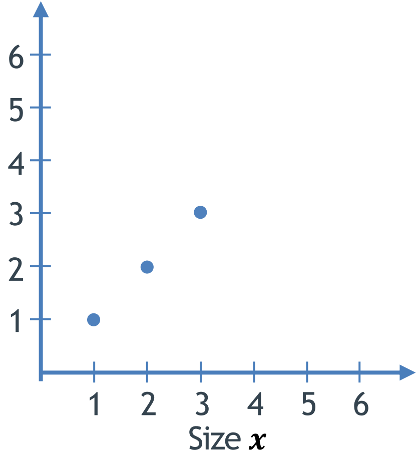
$$h_\theta(x) = \theta_0 + \theta_1 x$$

- The cost function is a 3D bowl-shaped surface:



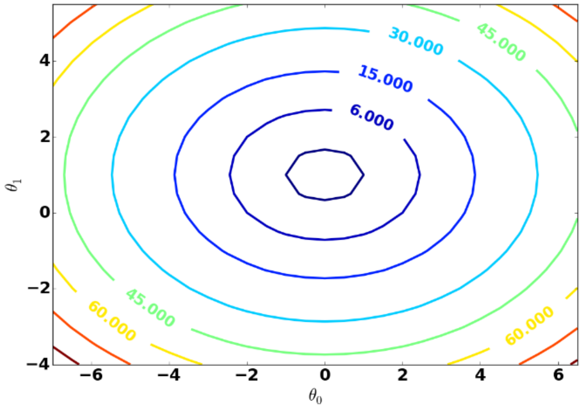
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)^2$$

Cost Function - Deciphering the Cost Function J With Two Params



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

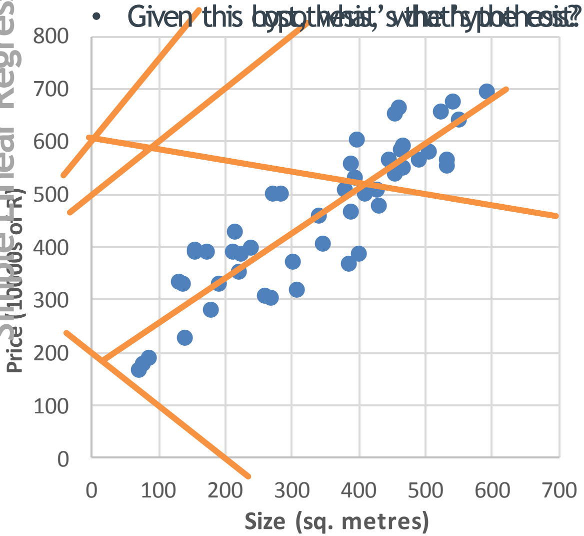
- The cost function as a contour plot:



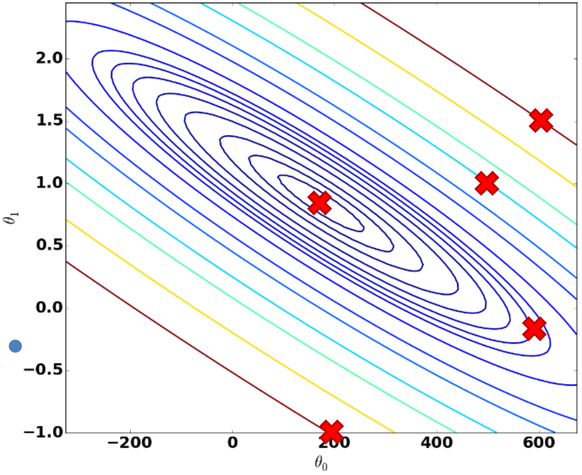
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

Cost Function - Deciphering the Cost Function J With Two Params

- Given this hypothesis, what's the cost?
- Given this cost, what's the hypothesis?



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



$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

Gradient Descent
Formulation

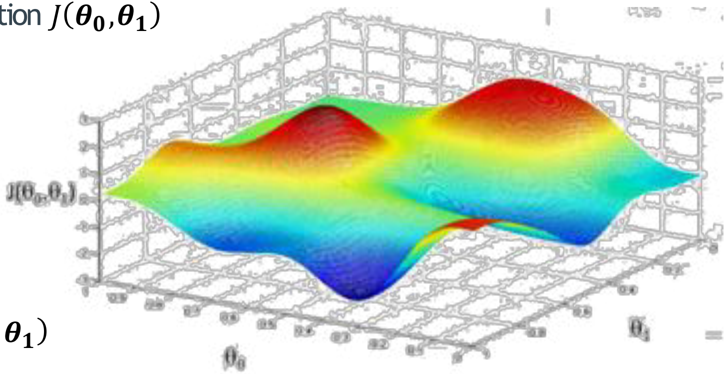
- Simple (Univariate) Linear Regression
  - Model Representation
  - Cost Function
- Gradient Descent
  - Formulation
  - Algorithm
  - Application to Linear Regression

Formulation - Introduction

- How would you go about reaching the minimum (water) below?

Formulation - Minimizing the Cost Function

- Given some arbitrary cost function $J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$



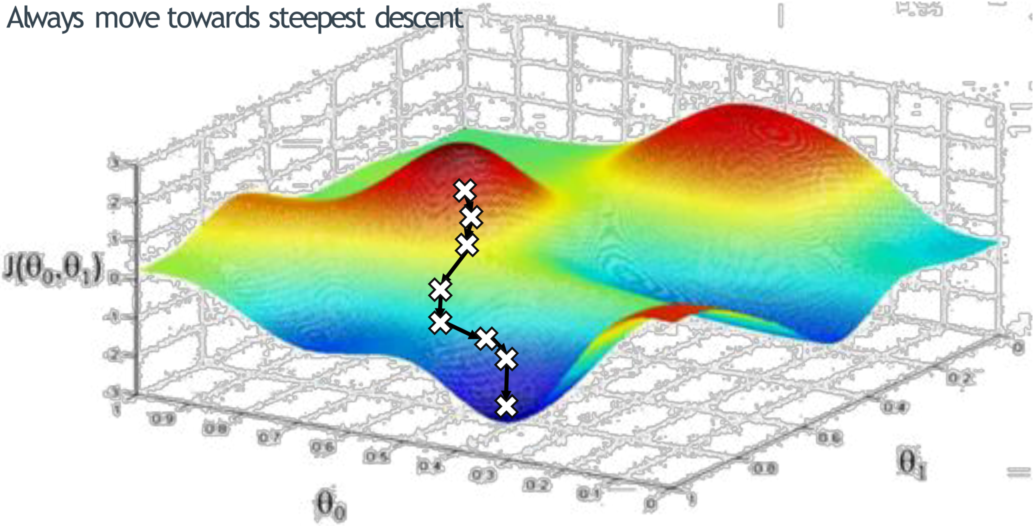
- We want to achieve:

$$\underset{\boldsymbol{\theta_0}, \boldsymbol{\theta_1}}{\text{minimize}} \; J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$$

- Strategy:
    - Initialize $\boldsymbol{\theta_0}, \boldsymbol{\theta_1}$ to some random values
    - Continuously make updates to $\boldsymbol{\theta_0}$ and $\boldsymbol{\theta_1}$ in the direction of “descent”
    - Until the minimum is reached

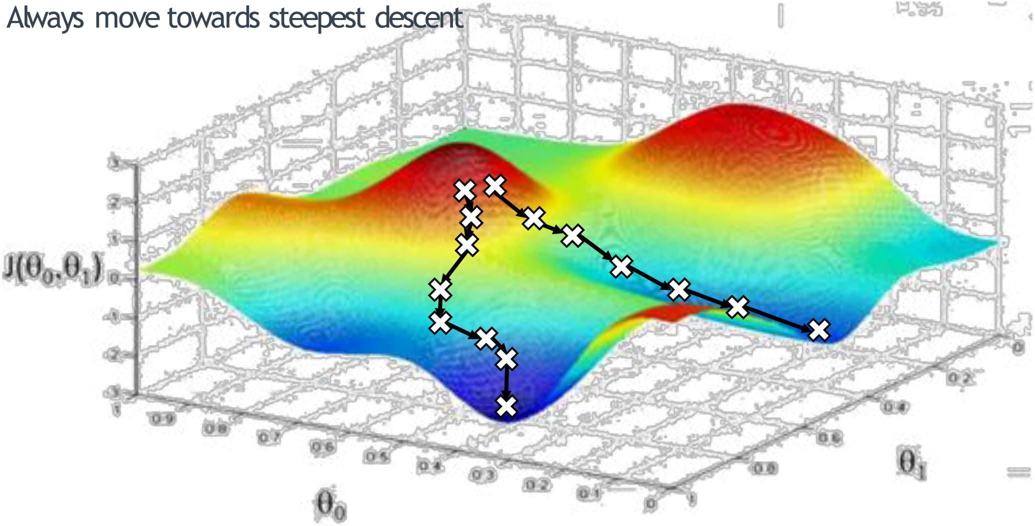
Formulation - Minimizing the Cost Function

- Starting at $\boldsymbol{\theta_0 = 0.7}$ and $\boldsymbol{\theta_1 = 0.4}$
- Always move towards steepest descent



Formulation - Minimizing the Cost Function

- Starting at $\boldsymbol{\theta_0 = 0.67}$ and $\boldsymbol{\theta_1 = 0.43}$
- Always move towards steepest descent



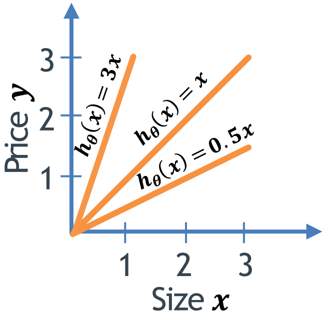
Formulation - Gradient Descent With One Parameter

- To show you how to formulate gradient descent, let's again assume:

$$\theta_0 = 0$$

- $h_\theta(x)$ is a line that passes through the origin (0,0), with changing gradients so:

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



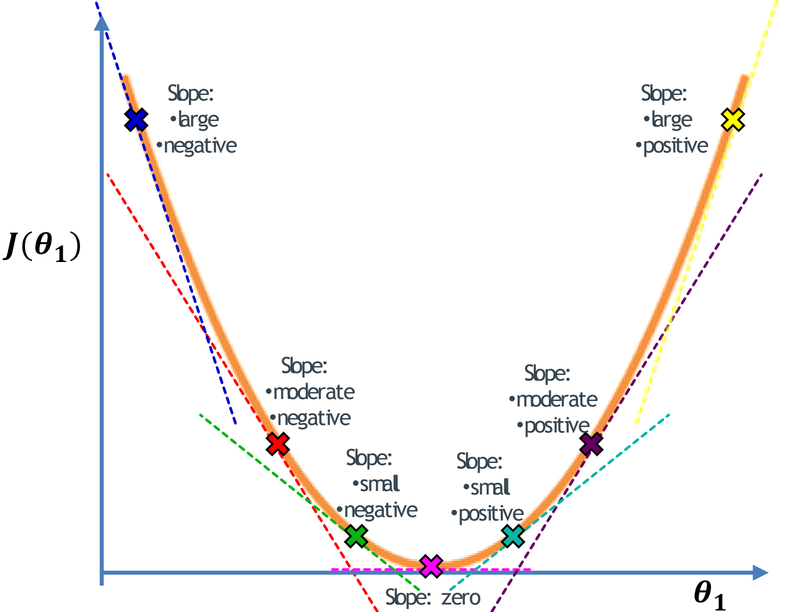
- Therefore the cost function now only depends on θ_1:

$$\underset{\theta_1}{\text{minimize}} \; J(\theta_1)$$

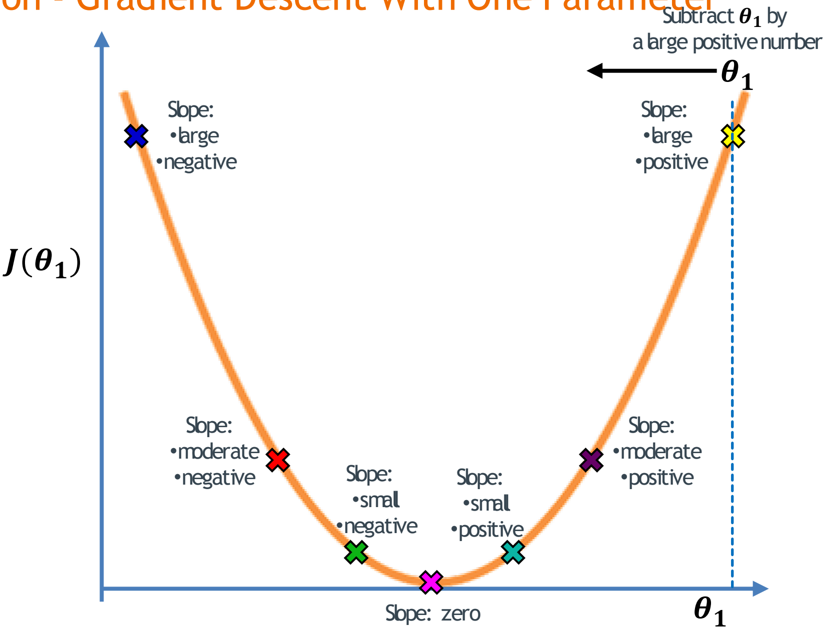
$$\underset{\theta_1}{\text{minimize}} \; \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$\underset{\theta_1}{\text{minimize}} \; \frac{1}{2m}\sum_{i=1}^{m}\left(\theta_1 x^{(i)} - y^{(i)}\right)^2$$

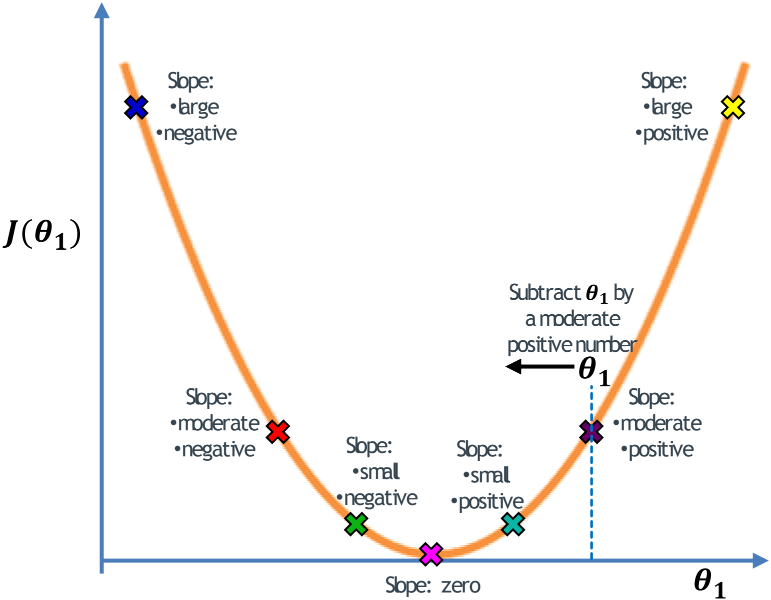
Formulation - Gradient Descent With One Parameter



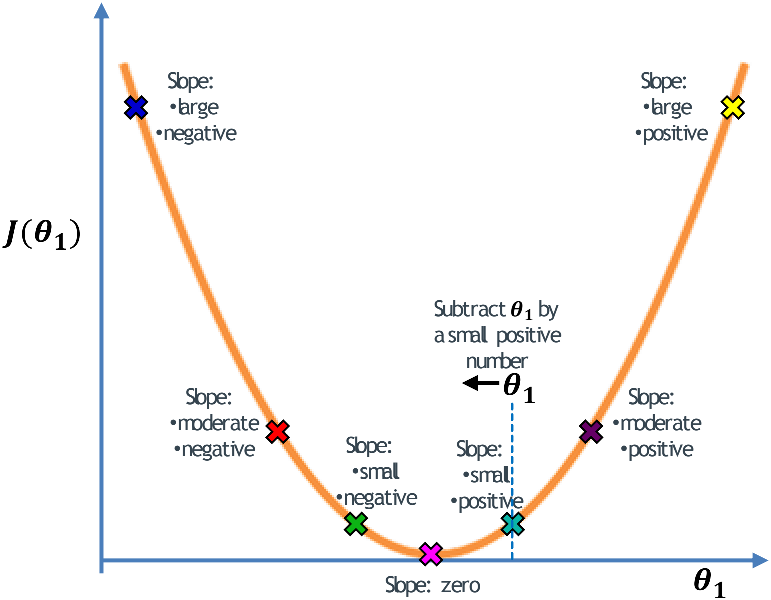
Formulation - Gradient Descent With One Parameter



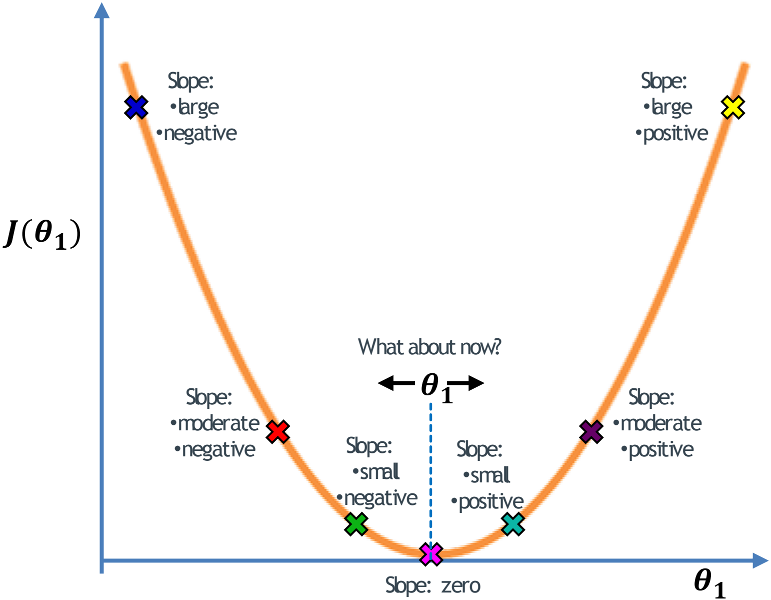
Formulation - Gradient Descent With One Parameter



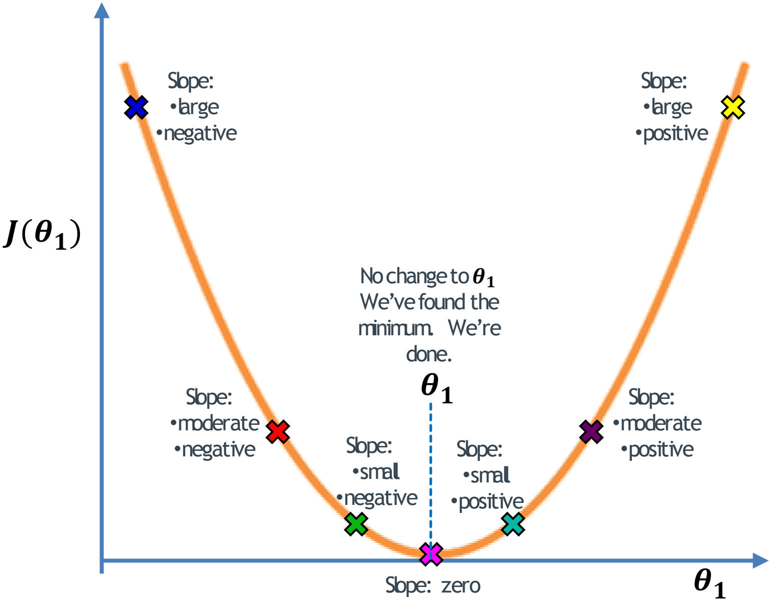
Formulation - Gradient Descent With One Parameter



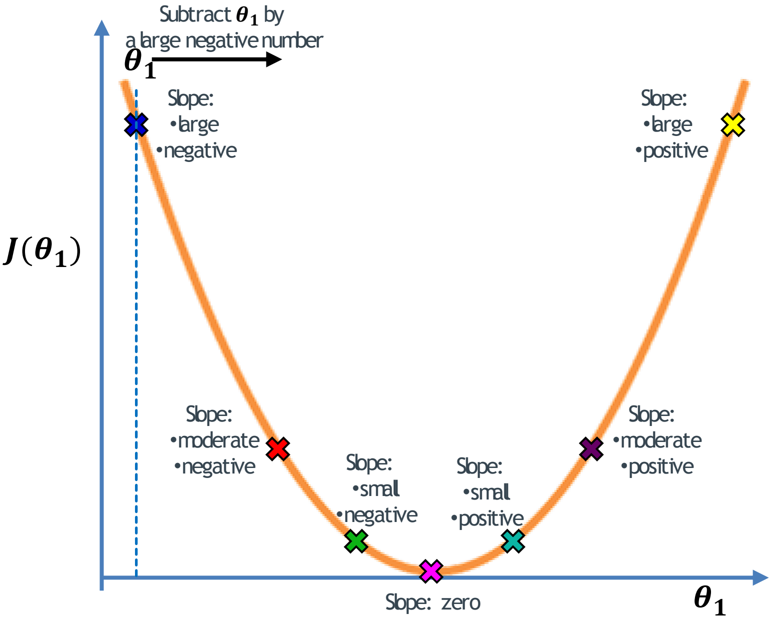
Formulation - Gradient Descent With One Parameter



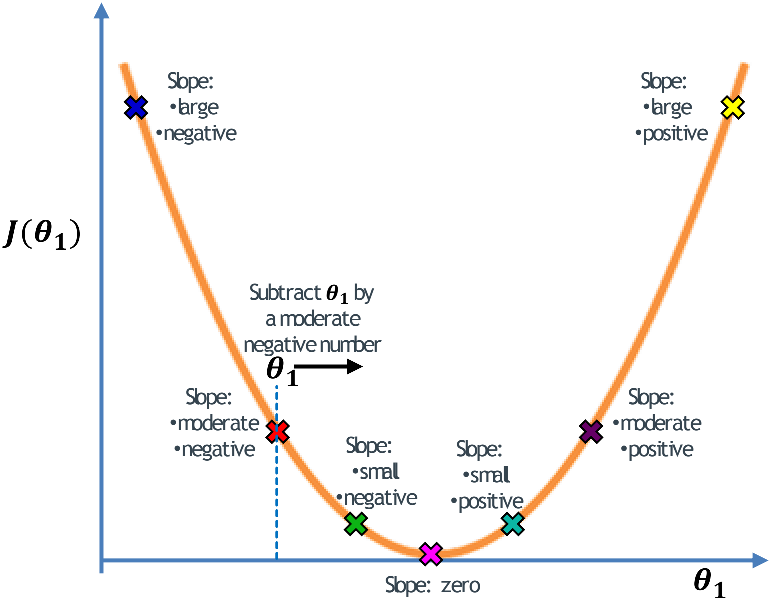
Formulation - Gradient Descent With One Parameter



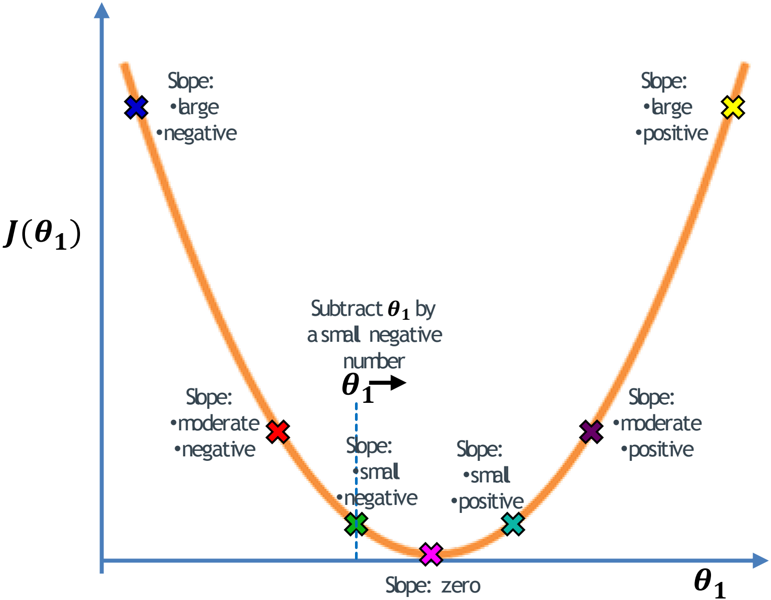
Formulation - Gradient Descent With One Parameter



Formulation - Gradient Descent With One Parameter



Formulation - Gradient Descent With One Parameter



Gradient Descent
Algorithm

- Simple (Univariate) Linear Regression
  - Model Representation
  - Cost Function
- Gradient Descent
  - Formulation
  - Algorithm
  - Application to Linear Regression

Algorithm - Gradient Descent With One Parameter

- For a cost function $J(\boldsymbol{\theta_1})$:

Repeat until convergence:

Update $\boldsymbol{\theta_1}$:

$$\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} - \quad \boxed{\frac{d}{d\boldsymbol{\theta_1}} J(\boldsymbol{\theta_1})}$$

Learning rate

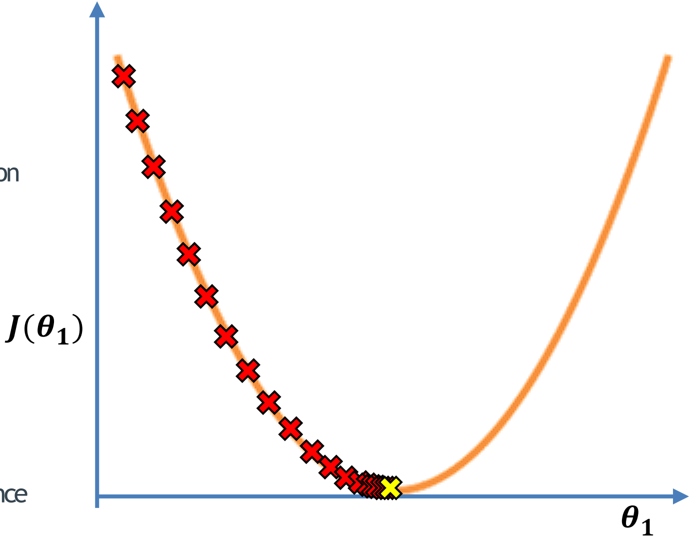
Slope

Algorithm – Deciphering Learning Rate

- With no learning rate:

$$\theta_1 \leftarrow \theta_1 - \frac{d}{d\theta_1} J(\theta_1)$$

- No learning rate: no controls on convergence
  - Could be too slow
  - Could be too fast
- Learning rate is very cost-function specific
- In this case:
  - Too slow
  - 20 steps to convergence



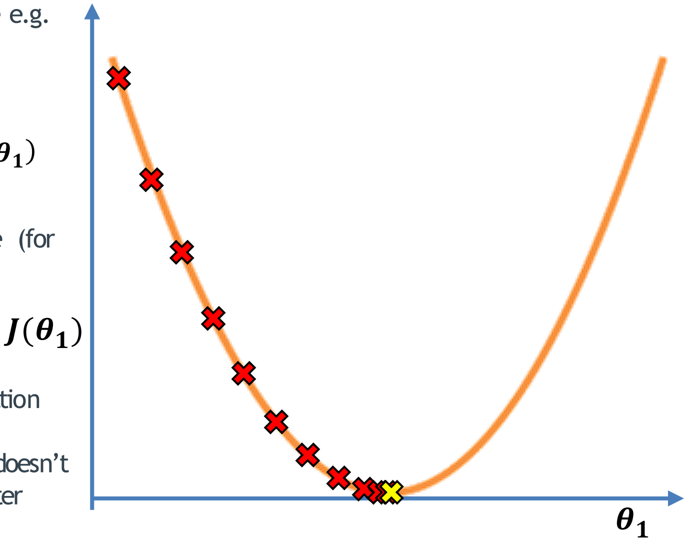
Algorithm – Deciphering Learning Rate

- With appropriate learning rate e.g. $\boldsymbol{\alpha}$ = 1.5 :
  - fast(er) convergence

$$\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} - \mathbf{1.5}\frac{\boldsymbol{d}}{\boldsymbol{d\theta_1}}J(\boldsymbol{\theta_1})$$

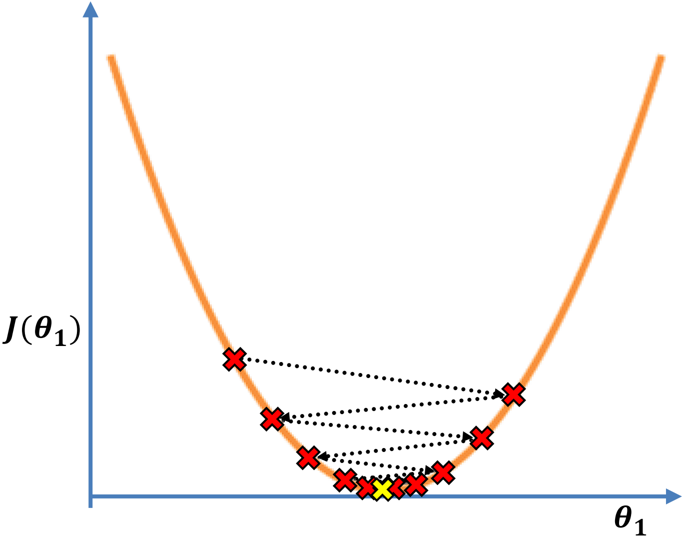
- 12 (vs 20) steps to convergence (for this specific cost function)

- Note: learning rate is cost-function specific
  - Higher learning rate doesn't necessarily mean faster convergence



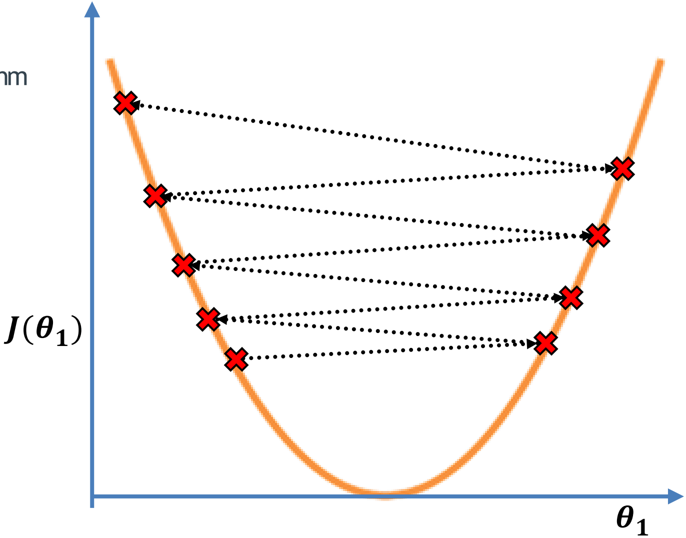
Algorithm – Deciphering Learning Rate

- With learning rate too large:
- **First possibility:** convergence becomes very slow



Algorithm – Deciphering Learning Rate

- With learning rate too large:
- **Second possibility:** the algorithm diverges (doesn't converge)



Algorithm – Gradient Descent With Two Parameters

- For a cost function $J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$:

Repeat until convergence:

Update $\boldsymbol{\theta_0}, \boldsymbol{\theta_1}$ simultaneously:

$$\boldsymbol{\theta_0} \leftarrow \boldsymbol{\theta_0} - \boldsymbol{\alpha} \boxed{\frac{\partial}{\partial \boldsymbol{\theta_0}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})}$$

$$\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} - \boldsymbol{\alpha} \boxed{\frac{\partial}{\partial \boldsymbol{\theta_1}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})}$$

Repeat until convergence:

tmptheta0 $= \text{theta0} - \boldsymbol{\alpha} \frac{\partial}{\partial \boldsymbol{\theta_0}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$

tmptheta1 $= \text{theta1} - \boldsymbol{\alpha} \frac{\partial}{\partial \boldsymbol{\theta_1}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$

theta0 = tmptheta0

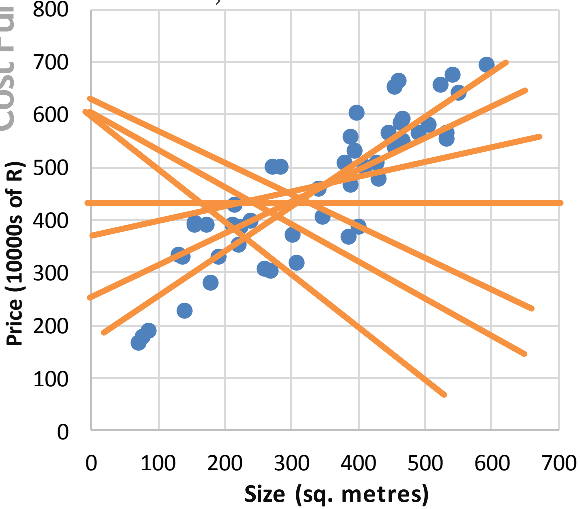
theta1 = tmptheta1

Slope of J in the $\boldsymbol{\theta_1}$ direction

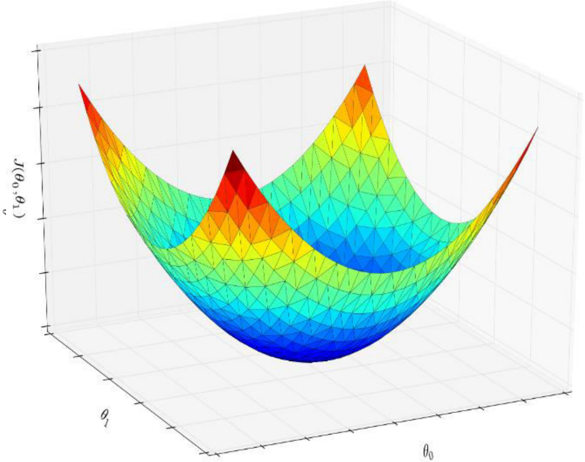
Slope of J in the $\boldsymbol{\theta_0}$ direction

Algorithm – Illustration

- Ok now, let's start somewhere and "descend" the cost function down to the minimum



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



$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

Algorithm – Gradient Descent With Generic Cost Function

- For a cost function $J(\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_n})$ with parameters $\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_n}$:

Repeat until convergence:

Update all $\boldsymbol{\theta_j}$ simultaneously:

$$\boldsymbol{\theta_j} \leftarrow \boldsymbol{\theta_j} - \boldsymbol{\alpha} \frac{\partial}{\partial \boldsymbol{\theta_j}} J(\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_n})$$

Repeat until convergence:

tmptheta0 = theta0 – $\boldsymbol{\alpha} \frac{\partial}{\partial \boldsymbol{\theta_0}} J(\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_n})$

...

tmpthetaj = thetaj – $\boldsymbol{\alpha} \frac{\partial}{\partial \boldsymbol{\theta_j}} J(\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_n})$

...

tmpthetan = thetan – $\boldsymbol{\alpha} \frac{\partial}{\partial \boldsymbol{\theta_n}} J(\boldsymbol{\theta_0}, \ldots, \boldsymbol{\theta_n})$

theta0 = tmptheta0

...

thetaj = tmpthetaj

...

thetan = tmpthetan

Gradient Descent

Gradient Descent Applied to Linear Regression

- Simple (Univariate) Linear Regression
  - Model Representation
  - Cost Function
- Gradient Descent
  - Formulation
  - Algorithm
  - Application to Linear Regression

Gradient Descent Applied to Linear Regression

- For a cost function $J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$:

Repeat until convergence:

Update $\boldsymbol{\theta_0}, \boldsymbol{\theta_1}$ simultaneously:

$$\boldsymbol{\theta_0} \leftarrow \boldsymbol{\theta_0} - \boldsymbol{\alpha} \boxed{\frac{\partial}{\partial \boldsymbol{\theta_0}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})}$$

$$\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} - \boldsymbol{\alpha} \boxed{\frac{\partial}{\partial \boldsymbol{\theta_1}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})}$$

? ?

- Linear Regression:

$$\boldsymbol{h_\theta}(\boldsymbol{x}) = \boldsymbol{\theta_0} + \boldsymbol{\theta_1} \boldsymbol{x}^{(i)}$$

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

$$= \frac{1}{2m} \sum_{i=1}^{m} \left(\boldsymbol{\theta_0} + \boldsymbol{\theta_1} \boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)}\right)^2$$

$$\frac{\partial}{\partial \boldsymbol{\theta_0}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1}) = ?$$

$$\frac{\partial}{\partial \boldsymbol{\theta_1}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1}) = ?$$

Gradient Descent Applied to Linear Regression

- For any parameter $\boldsymbol{\theta_j}$:

$$\frac{\partial}{\partial \boldsymbol{\theta_j}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1}) = \frac{\partial}{\partial \boldsymbol{\theta_j}} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(\boldsymbol{h_\theta}\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{y}^{(i)} \right)^2 = \frac{\partial}{\partial \boldsymbol{\theta_j}} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(\boldsymbol{\theta_0} + \boldsymbol{\theta_1} \boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)} \right)^2$$

- For $\boldsymbol{\theta_0}$:

$$\frac{\partial}{\partial \boldsymbol{\theta_0}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta_0} + \boldsymbol{\theta_1} \boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)} \right)$$

- For $\boldsymbol{\theta_1}$:

$$\frac{\partial}{\partial \boldsymbol{\theta_1}} J(\boldsymbol{\theta_0}, \boldsymbol{\theta_1}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta_0} + \boldsymbol{\theta_1} \boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)} \right) \cdot \boldsymbol{x}^{(i)}$$

Repeat until convergence:

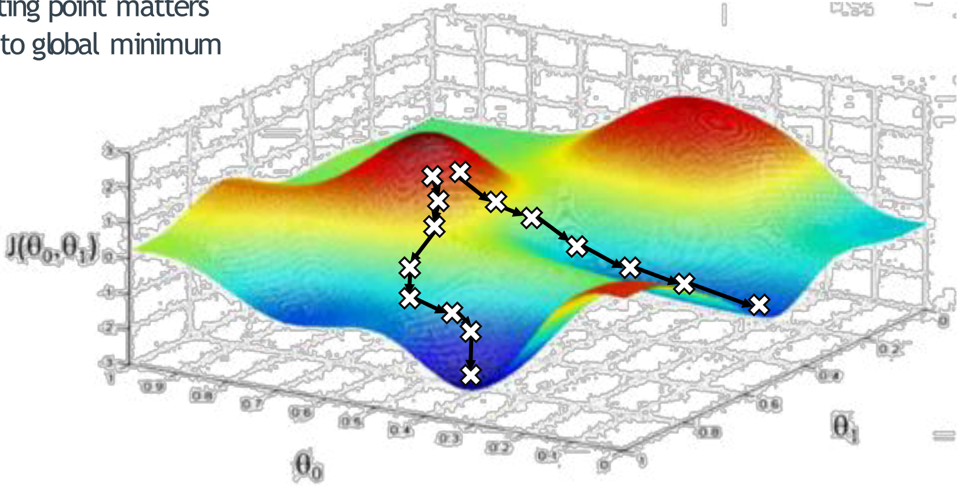
Update $\boldsymbol{\theta_0}, \boldsymbol{\theta_1}$ simultaneously:

$$\boldsymbol{\theta_0} \leftarrow \boldsymbol{\theta_0} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta_0} + \boldsymbol{\theta_1} \boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)} \right)$$

$$\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta_0} + \boldsymbol{\theta_1} \boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)} \right) \cdot \boldsymbol{x}^{(i)}$$

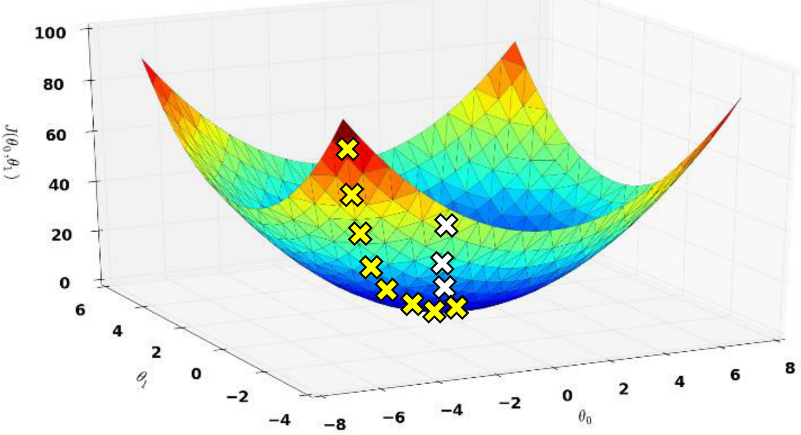
Gradient Descent Applied to Linear Regression

- Gradient descent works for minimizing any generic cost function
- Cost function may have many different local minima
    - Starting point matters
- May not get to global minimum



Gradient Descent Applied to Linear Regression

- The cost function of linear regression is "convex" i.e. it has only one minimum – global minimum
- With the right learning rate $\boldsymbol{\alpha}$, it always converges
- Starting point only affects how quickly it converges (i.e. starting out close VS far)



Note on Gradient Descent

- This kind of gradient descent is actually called "Batch Gradient Descent"
  - One update uses all the training samples

- Other types of gradient descent (not covered in this course – feel free to look them up):
  - Mini-Batch Gradient Descent
  - Stochastic Gradient Descent

Content - Part 2

- Linear Regression With Multiple Variables
    - Model Representation
    - Gradient Descent With Multiple Variables

- Practical Tips On Implementing Linear Regression
    - Diagnosing the Learning Rate
    - Feature Scaling

Linear Regression With Multiple Variables

Model Representation

- Linear Regression With Multiple Variables
  - Model Representation
  - Gradient Descent With Multiple Variables
- Practical Tips On Implementing Linear Regression
  - Diagnosing the Learning Rate
  - Feature Scaling

Previously (With Simple Linear Regression)

x Size (sq. m)	y Price (1000s of R)
460	6639
70	1681
155	3969
429	5095
…	…

Training set of house sizes and prices

- $\boldsymbol{x}$: Input variable; "features" used to make predictions e.g. 460 sq. metres
- $\boldsymbol{y}$: Output variable; "target" output e.g. R 6639000
- $\boldsymbol{m}$: Number of examples in the training set
- $(\boldsymbol{x}, \boldsymbol{y})$: A specific samples in the training set
- $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$: the $\boldsymbol{i}$th training example e.g. $\boldsymbol{x}^{(1)}$= 460, $\boldsymbol{x}^{(2)}$= 70… ; $\boldsymbol{y}^{(1)}$= 6639, $\boldsymbol{y}^{(2)}$= 1681

Model Representation – Notation With Multiple Variables

x Size (sq. m)	x_2 No. of Rooms	x_3 Age (years)	x_4 No. of Garages	y Price (1000s of R) (y)
460	4	12	2	6639
70	1	5	0	1681
155	3	8	2	3969
429	6	10	3	5095
…	…	…	…	…

Training set of house features and prices

- $\boldsymbol{y}$: Output variable; "target" output e.g. R6639000
- $\boldsymbol{m}$: Number of examples in the training set
- $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$: the $\boldsymbol{i}$th training example
- $\boldsymbol{y}^{(i)}$: the $\boldsymbol{i}$th target e.g. $\boldsymbol{y}^{(1)}$= 6639, $\boldsymbol{y}^{(2)}$= R1681 etc.

Model Representation – Notation With Multiple Variables

x_1	x_2	x_3	x_4	y
Size (sq. m)	No. of Rooms	Age (years)	No. of Garages	Price (1000s of R) (y)
460	4	12	2	6639
70	1	5	0	1681
155	3	8	2	3969
429	6	10	3	5095
…	…	…	…	…

Training set of house features and prices

- $\boldsymbol{n}$: Number of features; in the above case $\boldsymbol{n = 4}$
- $\boldsymbol{x_j}$: the $\boldsymbol{j}$th input variable (column); e.g. $\boldsymbol{x_1}$ is the size, $\boldsymbol{x_2}$ is the no. of rooms etc.
- $\boldsymbol{x^{(i)}}$: is now a set of $\boldsymbol{n}$ values i.e a vector
- $\boldsymbol{x_j^{(i)}}$: is the $\boldsymbol{j}$th feature of the $\boldsymbol{i}$th example i.e. a number

Model Representation – Notation With Multiple Variables

x_1 Size (sq. m)	x_2 No. of Rooms	x_3 Age (years)	x_4 No. of Garages	y Price (1000s of R) (y)
460	4	12	2	6639
70	1	5	0	1681
155	3	8	2	3969
429	6	10	3	5095
…	…	…	…	…

Training set of house features and prices

$$X = \begin{bmatrix} 460 & 4 & 12 & 2 \\ 70 & 1 & 5 & 0 \\ 155 & 3 & 8 & 2 \\ 429 & 6 & 10 & 3 \\ \dots & \dots & \dots & \dots \end{bmatrix}$$

$$y = \begin{bmatrix} 6639 \\ 1681 \\ 3969 \\ 5095 \\ \dots \end{bmatrix}$$

Model Representation – Hypothesis

- The previous hypothesis (with simple linear regression) with only one variable $\boldsymbol{x}$:

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

- With more variables: $\boldsymbol{x_1, \ldots, x_4}$:

$$h_\theta(x) = \theta_0 + \theta_1 \overset{\text{Size}}{x_1} + \theta_2 \overset{\text{\#Rooms}}{x_2} + \theta_3 \overset{\text{Age}}{x_3} + \theta_4 \overset{\text{\#Garages}}{x_4}$$

- E.g.

$$h_\theta(x) = 174 + 0.84x_1 + 1.8x_2 - 1.5x_3 + 3x_4$$

Model Representation – Notation With Multiple Variables

- In the generic case with $\boldsymbol{n}$ features:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- $\boldsymbol{n}$-dimensional hypothesis
    - We can't plot/visualize it for $n > 2$

- For consistency in the notation we add a 0th feature $x_0 = 1$ so that:

$$h_\theta(x) = \theta_0 \underbrace{x_0}_{=1} + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Model Representation – Vectorized Representation

- In the generic case with $\boldsymbol{n}$ features:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Everything in this column = 1

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ x_0^{(3)} & x_1^{(3)} & x_2^{(3)} & \dots & x_n^{(3)} \\ \dots & \dots & \dots & \dots & \dots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix}$$

- Given the two matrix-representations above, $h_\theta(x)$ can be computed using the matrix operation:

$$h_\theta(x) = X\theta$$

Model Representation – Vectorized Representation

- In the specific case with **4** features (in the example):

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$X = \begin{bmatrix} 1 & 460 & 4 & 12 & 2 \\ 1 & 70 & 1 & 5 & 0 \\ 1 & 155 & 3 & 8 & 2 \\ 1 & 429 & 6 & 10 & 3 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}$$

- Given the two matrix-representations above, $h_\theta(x)$ can be computed using the matrix operation:

$$h_\theta(x) = X\theta$$

Linear Regression With Multiple Variables

Gradient Descent With Multiple Variables

- Linear Regression With Multiple Variables
  - Model Representation
  - Gradient Descent With Multiple Variables
- Practical Tips On Implementing Linear Regression
  - Diagnosing the Learning Rate
  - Feature Scaling

Gradient Descent With Multiple Variables

- In the generic case with $\boldsymbol{n}$ features/variables:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

= 1

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \dots & x_n^{(2)} \\ \dots & \dots & \dots & \dots \\ x_0^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

- Cost function:

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

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix}$$

- Generic gradient descent update equation (as seen before):

Repeat until convergence:

Update all θ_j simultaneously:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$$

Gradient Descent With Multiple Variables

- For any parameter $\boldsymbol{\theta_j}$:

$$\frac{\partial}{\partial \boldsymbol{\theta_j}} J(\boldsymbol{\theta}) = \frac{\partial}{\partial \boldsymbol{\theta_j}} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(h_{\theta}\left(x^{(i)}\right) - y^{(i)} \right)^2$$

- Previously with only one variable $\boldsymbol{x}$: $\boldsymbol{\theta_0}, \boldsymbol{\theta_1}$:

Repeat until convergence:

Update $\boldsymbol{\theta_0}, \boldsymbol{\theta_1}$ simultaneously:

$$\boldsymbol{\theta_0} \leftarrow \boldsymbol{\theta_0} - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)$$

$$\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right) \cdot x^{(i)}$$

- With variables $\boldsymbol{n}>1$: $\boldsymbol{\theta_0}, \boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_n}$:

Repeat until convergence:

Update all $\boldsymbol{\theta}$ simultaneously:

$$\boldsymbol{\theta_0} \leftarrow \boldsymbol{\theta_0} - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right) \cdot x_0^{(i)}$$

$$\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right) \cdot x_1^{(i)}$$

$$\ldots$$

$$\boldsymbol{\theta_n} \leftarrow \boldsymbol{\theta_n} - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right) \cdot x_n^{(i)}$$

Gradient Descent With Multiple Variables – Vectorized Representation

- Given:

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ x_0^{(3)} & x_1^{(3)} & x_2^{(3)} & \dots & x_n^{(3)} \\ \dots & \dots & \dots & \dots & \dots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \dots \\ y^{(m)} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix}$$

$$h_\theta(x) = X\theta$$

- The updates to all theta can be made simultaneously using:

$$\theta \leftarrow \theta - \frac{\alpha}{m}\left[X^T(X\theta - y)\right]$$

Gradient Descent With Multiple Variables – Vectorized Representation

- As by the way: given:

$$\boldsymbol{X} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ x_0^{(3)} & x_1^{(3)} & x_2^{(3)} & \dots & x_n^{(3)} \\ \dots & \dots & \dots & \dots & \dots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \dots \\ y^{(m)} \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix}$$

$$h_{\theta}(x) = X\theta$$

- The cost $J(\boldsymbol{\theta})$ can be computed as:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

Practical Tips

Diagnosing the Learning Rate

- Linear Regression With Multiple Variables
  - Model Representation
  - Gradient Descent With Multiple Variables
- Practical Tips On Implementing Linear Regression
  - Diagnosing the Learning Rate
  - Feature Scaling

Diagnosing the Learning Rate

- Learning rate is specific to each problem

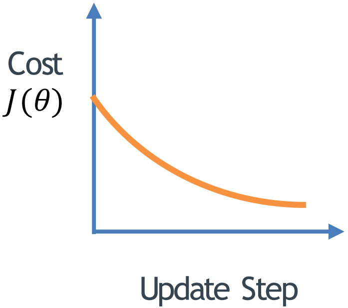
- Too small: slow convergence
- Too large:
    - Slow convergence
    - Divergence (no convergence)

- Helpful tip: Proven that if the learning rate $\boldsymbol{\alpha}$ is small enough, the cost J will reduce in every iteration of gradient descent

- Possible strategy: keep track of the cost at each update
    - Plot a graph of the cost at each update
    - Use the plot to determine if $\boldsymbol{\alpha}$ is:
        - Too small
        - Too large
    - Use a series of plots to pick the best $\boldsymbol{\alpha}$

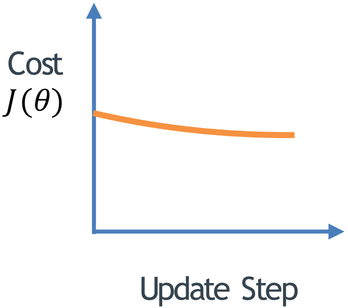
Diagnosing the Learning Rate

- If $\boldsymbol{\alpha}$ is just right:



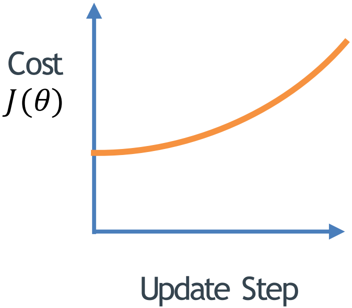
Diagnosing the Learning Rate

- If $\boldsymbol{\alpha}$ is too small:



Diagnosing the Learning Rate

- If $\boldsymbol{\alpha}$ is too large :



Diagnosing the Learning Rate - Strategy

- Try a range of values for $\boldsymbol{\alpha}$ on an increasing scale e.g.

  0.001 0.003 0.01 0.03 0.1 0.3 1

- Plot the curve of $\boldsymbol{J(\theta)}$ over the data for a number of iterations for every $\boldsymbol{\alpha}$
- Pick the $\boldsymbol{\alpha}$ value that converges the fastest

Practical Tips

Feature Scaling

- Linear Regression With Multiple Variables
  - Model Representation
  - Gradient Descent With Multiple Variables
- Practical Tips On Implementing Linear Regression
  - Diagnosing the Learning Rate
  - Feature Scaling

Feature Scaling – Concept

- Concept: Having features on a similar scale helps gradient descent converge faster
- Opposite: Having features on very different scales slows gradient descent down

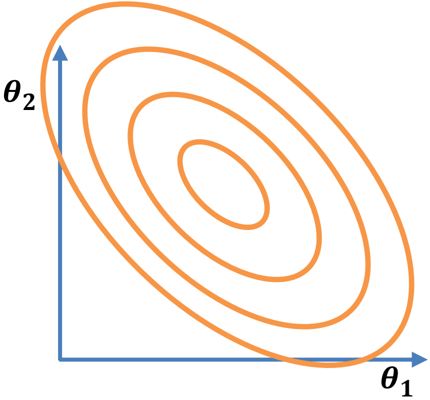
- Concept: scale features to similar scales

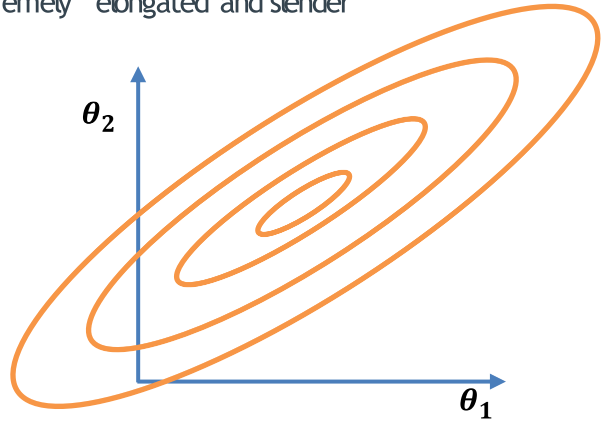
- Features may be on different scales

- E.g.:
  - House size: 0 – 5000 sq. metres i.e. 0 – 5000
  - No of bedrooms: 1 – 12 rooms i.e. 0 – 12
  - Age: 0 – 200 years i.e. 0 – 200

Feature Scaling - Intuition

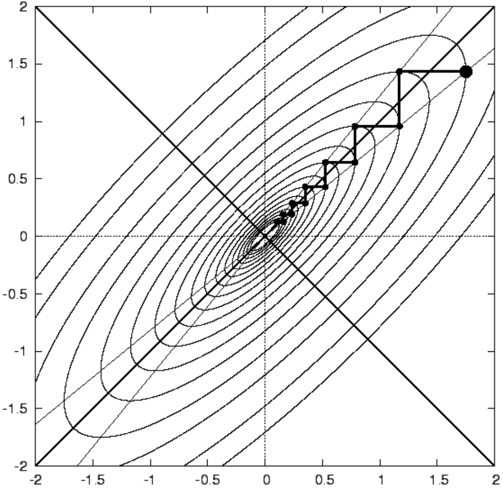
- One example E.g.:
    - No of bedrooms: 0 – 12
    - House size: 0 – 5000
- More extreme example E.g.:
    - Feature 1: 0 – 5
    - Feature 2: 0 – 20000000
- This causes the cost function to be extremely “elongated and slender”





Feature Scaling - Intuition

- This can cause gradient descent to take a long time to reach the minimum
- May also meander around a lot

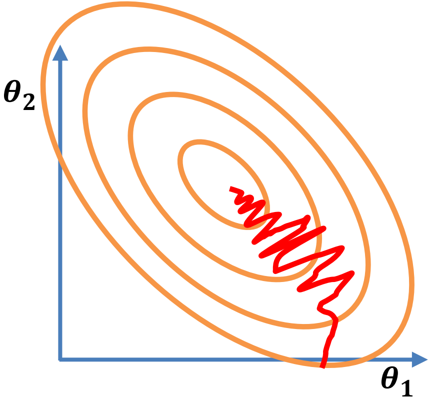


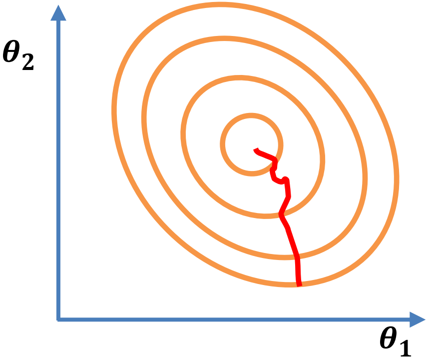
Feature Scaling - Intuition

- One example E.g.:
  - No of bedrooms: 0 – 12
  - House size: 0 – 5000
- Scale the features:

$$x_1 = \frac{\text{Bedrooms}}{12} \qquad x_2 = \frac{\text{Size}}{5000}$$

- With a smaller and more circular cost function, gradient descent can converge faster





Feature Scaling – Intuition

- Applying the simple division:

$$\boldsymbol{x_1} = \frac{\text{Bedrooms}}{12} \qquad\qquad \boldsymbol{x_2} = \frac{\text{Size}}{5000}$$

- Has the effect of re-scaling $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$.

  - $\boldsymbol{x_1}$: re-scaled to 0 - 1
  - $\boldsymbol{x_2}$: re-scaled to 0 – 1

Feature Scaling – Concept

- $\boldsymbol{x_0 = 1}$ in all cases, so no scaling is needed

- For all other features: aim is to re-scale every feature to **approximately** $\boldsymbol{-1 \leq x_1 \leq 1}$ range

- Approximately means all of the following are **okay**:
  - $\boldsymbol{0 \leq x_1 \leq 4}$
  - $\boldsymbol{-2 \leq x_2 \leq 1}$
  - $\boldsymbol{-0.5 \leq x_3 \leq 3}$

- The following are examples of **not okay**:
  - $\boldsymbol{-80 \leq x_4 \leq 90}$
  - $\boldsymbol{-0.0001 \leq x_5 \leq 0.0004}$

Feature Scaling – Method

- Applied to every feature except $\boldsymbol{x_0}$

- Replace $\boldsymbol{x_j}$ with $\boldsymbol{x_j - \mu_j}$
  - where $\boldsymbol{\mu_j}$ is the average of all values of $\boldsymbol{x_j}$
  - This results in a 0 mean for this feature

$$\boldsymbol{x_j \leftarrow \frac{x_j - \mu_j}{s_j}}$$

- Divide by $\boldsymbol{S_j}$ which can be either of:
  - The range of values i.e. $\mathbf{max}(\boldsymbol{x_j}) - \mathbf{min}(\boldsymbol{x_j})$ OR
  - The standard deviation of values of $\boldsymbol{x_j}$

$$\boldsymbol{X} = \begin{bmatrix} \mathbf{1} & \mathbf{460} & \mathbf{2} \\ \mathbf{1} & \mathbf{70} & \mathbf{0} \\ \mathbf{1} & \mathbf{155} & \mathbf{2} \\ \mathbf{1} & \mathbf{429} & \mathbf{3} \end{bmatrix}$$

- Results in the approximate range:
  - $\mathbf{-0.5 \le \boldsymbol{x_j} \le 0.5}$

$$\boldsymbol{X\_scaled} = \begin{bmatrix} \mathbf{1} & \mathbf{0.46} & \mathbf{0.08} \\ \mathbf{1} & \mathbf{-0.53} & \mathbf{0.58} \\ \mathbf{1} & \mathbf{-0.32} & \mathbf{0.08} \\ \mathbf{1} & \mathbf{0.38} & \mathbf{0.42} \end{bmatrix}$$

Content - Part 3

- Linear Regression Using The Normal Equation

- Complex (Non-Linear) Features

- Model Evaluation
    - Testing a Model
    - Comparing Models
    - Getting More Mileage Out of Your Data

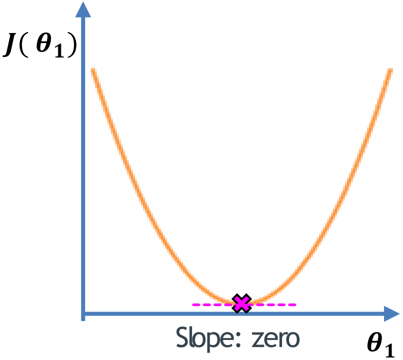
Linear Regression Using the Normal Equation

- Linear Regression Using The Normal Equation
- Complex (Non-Linear) Features
- Model Evaluation
  - Testing a Model
  - Comparing Models
  - Getting More Mileage Out of Your Data

Intuition

- A second analytical alternative to solving for parameters $\boldsymbol{\theta}$
    - (First one was gradient descent - algorithmic solution to $\boldsymbol{\theta}$)

- For a very simple hypothesis $h_{\theta}(x) = \theta_1 x_1$ the cost function looks like the below:



- Alternative to getting $\boldsymbol{\theta}$: solve for $\boldsymbol{\theta}$ in:

$$\frac{d}{d\,\theta_1} J(\theta_1) = 0$$

- The only unknown is θ_1. Solvable

- With many more features, concept is the same:
    - Solve for $\boldsymbol{\theta}$ in:

$$\frac{d}{d\,\theta_1} J(\theta_0, \theta_1, \dots, \theta_n) = 0$$

Normal Equation Definition

- Given a feature matrix $\boldsymbol{X}$, and the corresponding output matrix $\boldsymbol{y}$
- The following equation solves for $\boldsymbol{\theta}$ that best fits the data:

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

- Where:
  - $\boldsymbol{X}^T$ is the transpose of $\boldsymbol{X}$
  - $(\boldsymbol{X}^T \boldsymbol{X})^{-1}$ is the inverse of $(\boldsymbol{X}^T \boldsymbol{X})$
  - $\boldsymbol{X}$ includes feature $\boldsymbol{x_0}$ which is a column of 1s

- E.g.

$$\boldsymbol{X} = \begin{array}{c} \begin{array}{ccccc} & \text{Size} & \text{\#Rms} & \text{Age} & \text{\#Grgs} \end{array} \\ \begin{bmatrix} 1 & 460 & 4 & 12 & 2 \\ 1 & 70 & 1 & 5 & 0 \\ 1 & 155 & 3 & 8 & 2 \\ 1 & 429 & 6 & 10 & 3 \end{bmatrix} \end{array} \qquad \boldsymbol{y} = \begin{array}{c} \text{Price} \\ \begin{bmatrix} 6639 \\ 1681 \\ 3969 \\ 5095 \end{bmatrix} \end{array}$$

Normal Equation - Notes

- To get the inverse In SciPy:
- First import scipy.linalg as spla, then either
    - spla.inv(X) OR:
    - spla.pinv(X)
    - Rather use spla.pinv(X)

- To get matrix transpose in SciPy:
    - Easiest way: X.T

- No need for scaling at all
    - It's not descending a function

Normal Equation VS Gradient Descent

Normal Equation

Pros

- Doesn't need any iterations - finds the solution in one go immediately
- No need to choose α
- Preferable over gradient descent if possible

Cons

- Can be impossibly slow to compute $(X^TX)^{-1}$ when n gets large $(n > 10000)$
  - $O(n)^3$ time
- $(X^TX)^{-1}$ can be non-invertible
  - E.g. when $n \gg m$

Gradient Descent

Pros

- Works well even for very large n

Cons

- Need to choose α
- Could take many iterations to find best θ fit

Complex (Non-Linear) Features

- Linear Regression Using The Normal Equation
- Complex (Non-Linear) Features
- Model Evaluation
  - Testing a Model
  - Comparing Models
  - Getting More Mileage Out of Your Data

Concept

- Up to now, we've used simple linear features e.g. Size, No of Rooms etc.
- Possible to use linear regression to learn more complex features:
  - Combinations of features
  - Higher-order features

Combinations Of Features

- Assume we've got 4 features: size, no of rooms, height and width of properties
- Up to now: Linear combination of features

$$h_\theta(x) = \theta_0 + \theta_1 \cdot \text{Size} + \theta_2 \cdot \text{Rooms} + \theta_3 \cdot \text{Height} + \theta_4 \cdot \text{Width}$$

- Given insight into the problem, you can create new features using these basic features
  - If you think they are more "telling" / "feature-ful"

Combinations Of Features

- E.g. Divide Size by the No of Rooms to get the "Size-to-rooms-ratio" (STRR)
    - (Maybe) cramming more rooms in a smaller area (smaller STRR) means lower quality i.e. price
    - Conversely, (maybe) placing fewer rooms in a larger area (larger STRR) means higher quality i.e. price

- We can replace the two features $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ with just a single STRR feature:

$$\boldsymbol{h_\theta(x) = \theta_0 + \theta_1} \cdot \mathbf{Size} + \theta_2 \cdot \mathbf{Rooms} + \theta_3 \cdot \mathbf{Height} + \theta_4 \cdot \mathbf{Width}$$

$$\boldsymbol{h_\theta(x) = \theta_0 + \theta_1} \cdot \mathbf{STRR} + \boldsymbol{\theta_2} \cdot \mathbf{Height} + \boldsymbol{\theta_3} \cdot \mathbf{Width}$$

- Where STRR is now feature $\boldsymbol{x_1} = (\mathbf{Size} \cdot \mathbf{Rooms})$

Combinations Of Features

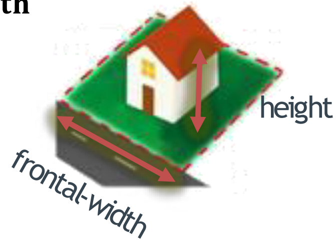
- E.g. 2: Multiply the frontal-width of the house by the height of the house to get the area of the frontal display (frontal-display area - FDA)
    - (Maybe) Having a larger frontal display area leads to a better first impression → higher demand → higher price
    - Conversely (maybe) a smaller frontal display area leads to a let-down first impression → lower demand → lower price

- We can replace the two features $\boldsymbol{x_2}$ and $\boldsymbol{x_3}$ with just a single FDA feature:

$$h_\theta(x) = \theta_0 + \theta_1 \cdot \mathbf{STRR} + \theta_2 \cdot \mathbf{FDA}$$

$$h_\theta(x) = \theta_0 + \theta_1 \cdot \mathbf{STRR} + \theta_2 \cdot \mathbf{Height} + \theta_3 \cdot \mathbf{Width}$$

- Where FDA is now feature $\boldsymbol{x_2} = (\mathbf{Height} \cdot \mathbf{Width})$

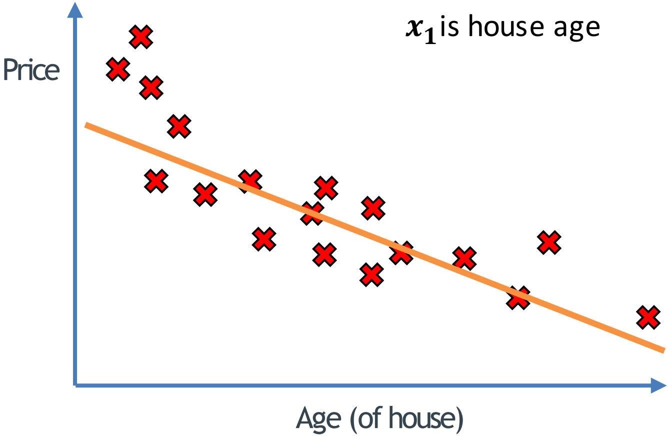


Higher-Order Features

- Closely related to previous idea

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

x_1 is house age

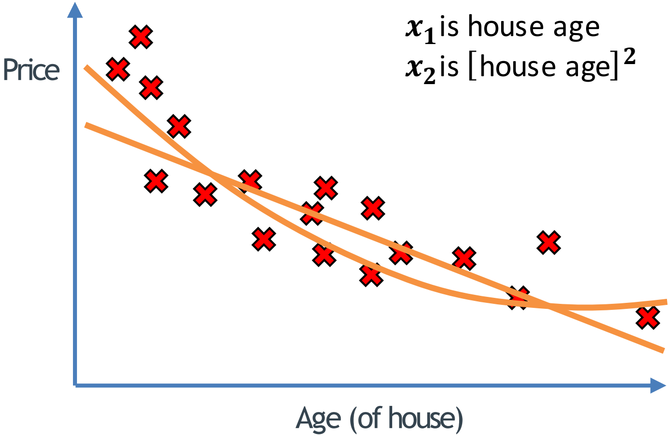


Higher-Order Features

- Maybe a second-order polynomial (quadratic function) makes more sense i.e. fits this data better

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

x_1 is house age
x_2 is $[\text{house age}]^2$

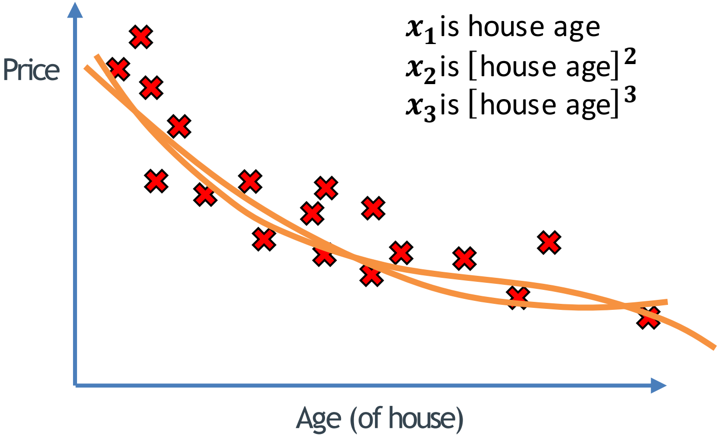


Higher-Order Features

- Or maybe a third-order polynomial makes more sense i.e. fits this data better

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

x_1 is house age
x_2 is $[\text{house age}]^2$
x_3 is $[\text{house age}]^3$

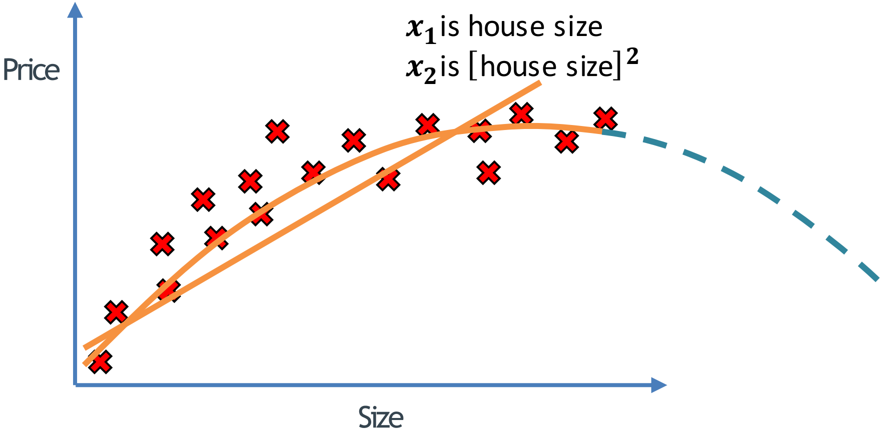


Higher-Order Features

- Another example: Price vs Size
- Maybe a quadratic function fits better

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

x_1 is house size
x_2 is [house size]^2



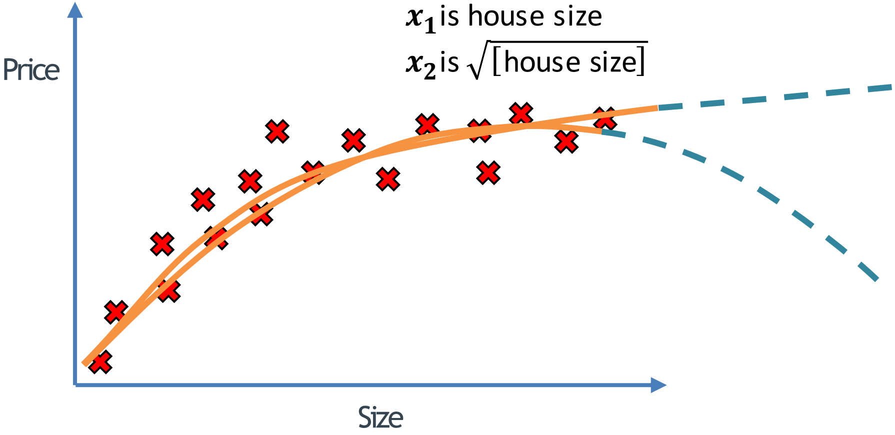
Higher-Order Features

- Another example: Price vs Size
- Maybe a square root function actually makes more sense

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

x_1 is house size

x_2 is $\sqrt{[\text{house size}]}$



Complex Features – Important Points

- The list of feature possibilities (combinations / higher-order features) are endless
    - Provides flexibility VS Challenging to choose good features

- Extremely important to scale features if you're using gradient descent with complex features:
    - If $\boldsymbol{x_1}$ has the range $\mathbf{0} \leq \boldsymbol{x_1} \leq \mathbf{1000}$
    - Then:
    - $\boldsymbol{x_1^2}$ has the range $\mathbf{0} \leq \boldsymbol{x_2} \leq \mathbf{1,000,000}$
    - $\boldsymbol{x_1^3}$ has the range $\mathbf{0} \leq \boldsymbol{x_3} \leq \mathbf{1,000,000,000}$
    - $\sqrt{\boldsymbol{x_1}}$ has the range $\mathbf{0} \leq \boldsymbol{x_3} \leq \mathbf{32}$

- When trying to predict on a set of test data, produce the same combination / higher-order features
    - Format has to be exactly the same as those used in training

Model Evaluation
Testing a Model

- Linear Regression Using The Normal Equation
- Complex (Non-Linear) Features
- Model Evaluation
  - Testing a Model
  - Comparing Models
  - Getting More Mileage Out of Your Data

Testing a Model

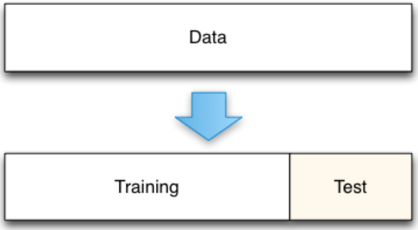
- Given a training set: train a predictive model
- Question: how well does this predict?

- One strategy: Test on the training data:
  - Model is tailor-made for the training data
  - Will (likely) give a good result
  - Not a good indicator of accuracy

- Another strategy: Collect more data
  - Very impractical
  - Expensive

Testing a Model

- Good strategy: Divide up the data that you have
  - Training portion (between 50% and 90%)
  - Testing portion (between 50% and 10%)
  - VERY important to randomize first

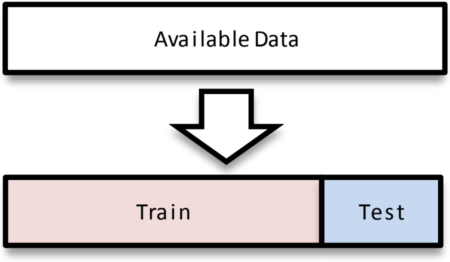


- Do not use testing data in training at all
  - It is “unseen” to the model

- Once model is trained, pass testing data to model to make predictions
  - Then apply a metric to determine performance

Testing a Model

- Good strategy: Divide up the data that you have
  - Training portion (between 50% and 90%)
  - Testing portion (between 50% and 10%)
  - VERY important to randomize first



$$X = \begin{bmatrix} 460 & 4 & 12 & 2 \\ 70 & 1 & 5 & 0 \\ 155 & 3 & 8 & 2 \\ 429 & 6 & 10 & 3 \\ 313 & 3 & 4 & 1 \\ 56 & 2 & 1 & 0 \\ \hline 434 & 1 & 9 & 3 \\ 123 & 7 & 3 & 1 \end{bmatrix} \qquad y = \begin{bmatrix} 6639 \\ 1681 \\ 3969 \\ 5095 \\ 3412 \\ 1234 \\ \hline 2334 \\ 2786 \end{bmatrix}$$

Training portion

Testing portion

Testing a Model - Metrics

- Three metrics:
  - Mean Absolute Error: $$\mathbf{MAE} = \frac{1}{m}\sum_{i=1}^{m}\left|y^{(i)} - y_{pred}^{(i)}\right|$$ $\mathbf{0} - \infty$ The smaller the better
  - Mean Squared Error: $$\mathbf{MSE} = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - y_{pred}^{(i)}\right)^{2}$$ $\mathbf{0} - \infty$ The smaller the better
  - Chi-squared error R^2: [Placeholder - Look it up] $-\infty - \mathbf{1}$

    $-\infty$ - Very poor fit

    1 - Perfect fit

Evaluating a Model

Comparing Models

- Linear Regression Using The Normal Equation
- Complex (Non-Linear) Features
- Model Evaluation
  - Testing a Model
  - Comparing Models
  - Getting More Mileage Out of Your Data

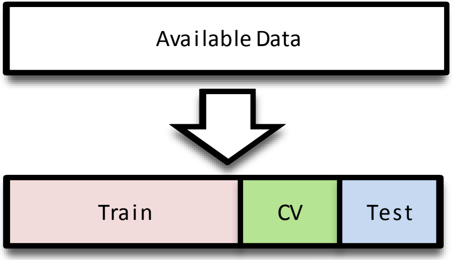
Comparing Models

- The list of features to try are endless
- Given several models/options to compare e.g.
    - quadratic features <u>VS</u> square root features
    - Using Size + No Of Rooms <u>VS</u> Size + No Of Garages <u>VS</u> No of Rooms + No Of Garages
- How to compare them?

- Possible strategy:
    - Use the same strategy used to test a single model i.e.
        - Divide available data into training and testing sets
        - Train all models on training sets
        - Test all models on testing sets
        - Conclude which is the best
    - Technically not statistically valid; more valid strategy:
        - After picking the “winner” we need to test it on a final piece of data to quote its performance

Comparing Models

- Valid strategy: Divide up the data that you have
  - Training portion (between 50% and 80%)
  - Cross-validation portion (between 25% and 10%)
  - Testing portion (between 25% and 10%)
  - VERY important to randomize first



$$X = \begin{bmatrix} 460 & 4 & 12 & 2 \\ 70 & 1 & 5 & 0 \\ 155 & 3 & 8 & 2 \\ 429 & 6 & 10 & 3 \\ \hline 313 & 3 & 4 & 1 \\ 56 & 2 & 1 & 0 \\ \hline 434 & 1 & 9 & 3 \\ 123 & 7 & 3 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 6639 \\ 1681 \\ 3969 \\ 5095 \\ \hline 3412 \\ 1234 \\ \hline 2334 \\ 2786 \end{bmatrix}$$

Training portion

CV portion

Testing portion

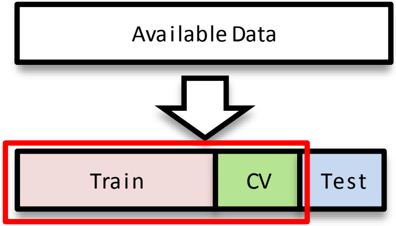
Evaluating a Model

Getting More Mileage Out of Your Data

- Linear Regression Using The Normal Equation
- Complex (Non-Linear) Features
- Model Evaluation
  - Testing a Model
  - Comparing Models
  - Getting More Mileage Out of Your Data

Getting More Mileage Out of Your Data

- Given a limited-size data set, possible to use
- techniques to stretch out "Training" and "CV" portions
- out further:
    - k-Fold Cross-validation
    - Leave-one-out Cross-validation



$$X = \begin{bmatrix} 460 & 4 & 12 & 2 \\ 70 & 1 & 5 & 0 \\ 155 & 3 & 8 & 2 \\ 429 & 6 & 10 & 3 \\ \hline 313 & 3 & 4 & 1 \\ 56 & 2 & 1 & 0 \\ \hline 434 & 1 & 9 & 3 \\ 123 & 7 & 3 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 6639 \\ 1681 \\ 3969 \\ 5095 \\ \hline 3412 \\ 1234 \\ \hline 2334 \\ 2786 \end{bmatrix}$$

Training portion

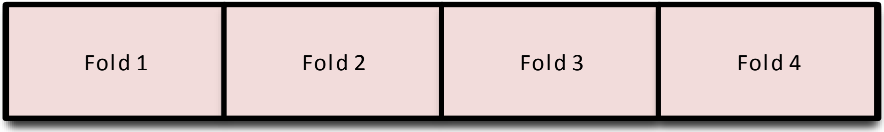
CV portion

Testing portion

We can stretch out these parts

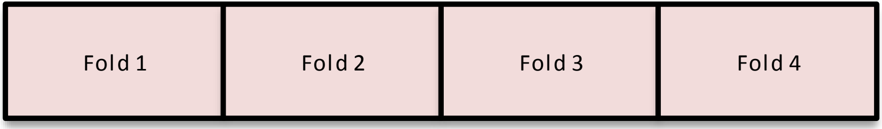
k-Fold Cross Validation

- Given the Training + CV parts below
  - Treat them as one big “Training Set”
  - Divide it up into *k*-Folds
    - E.g. 4 folds below



k-Fold Cross Validation

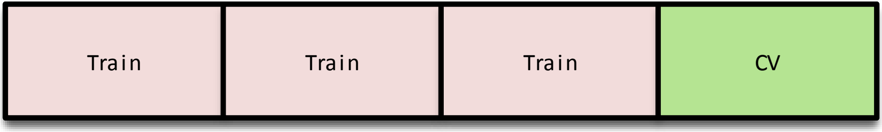
- Given the Training + CV parts below
  - Treat them as one big “Training Set”
  - Divide it up into *k*-Folds
    - E.g. 4 folds below
  - Train *k* times:
    - Each time take 1 fold as the CV set and the remaining folds as training sets



k-Fold Cross Validation

- Given the Training + CV parts below
    - Treat them as one big "Training Set"
    - Divide it up into *k*-Folds
        - E.g. 4 folds below
    - Train *k* times:
        - Each time take 1 fold as the CV set and the remaining folds as training sets
        - Train on all the data marked as "Train"; Test on the data marked "CV"
        - Get an evaluation score for the model

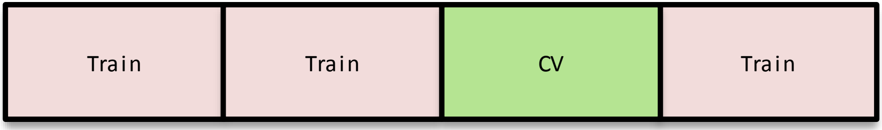
First Time



k-Fold Cross Validation

- Given the Training + CV parts below
    - Treat them as one big "Training Set"
    - Divide it up into *k*-Folds
        - E.g. 4 folds below
    - Train *k* times:
        - Each time take 1 fold as the CV set and the remaining folds as training sets
        - Train on all the data marked as "Train"; Test on the data marked "CV"
        - Get an evaluation score for the model

Second Time



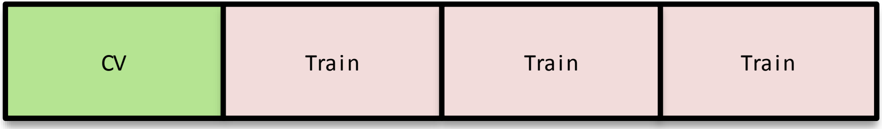
k-Fold Cross Validation

- Given the Training + CV parts below
    - Treat them as one big "Training Set"
    - Divide it up into *k*-Folds
        - E.g. 4 folds below
    - Train *k* times:
        - Each time take 1 fold as the CV set and the remaining folds as training sets
        - Train on all the data marked as "Train"; Test on the data marked "CV"
        - Get an evaluation score for the model

k-Fold Cross Validation

- Given the Training + CV parts below
  - Treat them as one big “Training Set”
  - Divide it up into *k*-Folds
    - E.g. 4 folds below
  - Train *k* times:
    - Each time take 1 fold as the CV set and the remaining folds as training sets
    - Train on all the data marked as “Train”; Test on the data marked “CV”
    - Get an evaluation score for the model

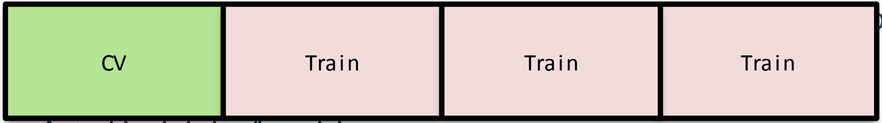
Fourth Time



k-Fold Cross Validation

- Given the Training + CV parts below
    - Treat them as one big "Training Set"
    - Divide it up into *k*-Folds
        - E.g. 4 folds below
    - Train *k* times:
        - Each time take 1 fold as the CV set and the remaining folds as training sets
        - Train on all the data marked as "Train"; Test on the data marked "CV"
        - Get an evaluation score for the model

Fourth Time



od average

- Arguably statistically valid
- Note: Once done, test on the test set (which was set aside)

THE END

Of Linear Regression