Supervised Learning

Introduction

Some slides were adapted/taken from various sources, including Prof. Andrew Ng's Coursera Lectures, Stanford University, Prof. Kilian Q. Weinberger's lectures on Machine Learning, Cornell University, Prof. Sudeshna Sarkar's Lecture on Machine Learning, IIT Kharagpur, Prof. Bing Liu's lecture, University of Illinois at Chicago (UIC), CS231n: Convolutional Neural Networks for Visual Recognition lectures, Stanford University and many more. We thankfully acknowledge them. Students are requested to use this material for their study only and **NOT** to distribute it.

Supervised Learning Setup

- Training data comes in pairs of inputs (x,y), where $x \in R^d$ is the input instance and y its label.

$$D = \{(\mathbf{x}_1, y_1), \dots \dots (\mathbf{x}_n, y_n)\} \subseteq \mathcal{R}^d \times C$$

where:

- R^d is the d-dimensional feature space
- x_i is the input vector of the ith sample
- y_i is the label of the ith sample
- C is the label space

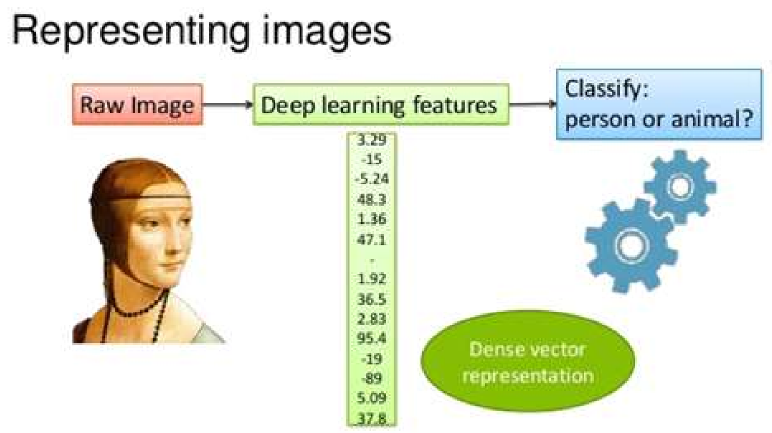
The data points $(\mathbf{x}_1, y_1)$are drawn from some (unknown) distribution $P(x, y)$. Ultimately we would like to learn a function h such that for a new pair,$(x, y) \sim P$, we have h(x)=y with high probability (or h(x)≈y). We will get to this later. For now let us go through some examples of X and Y.

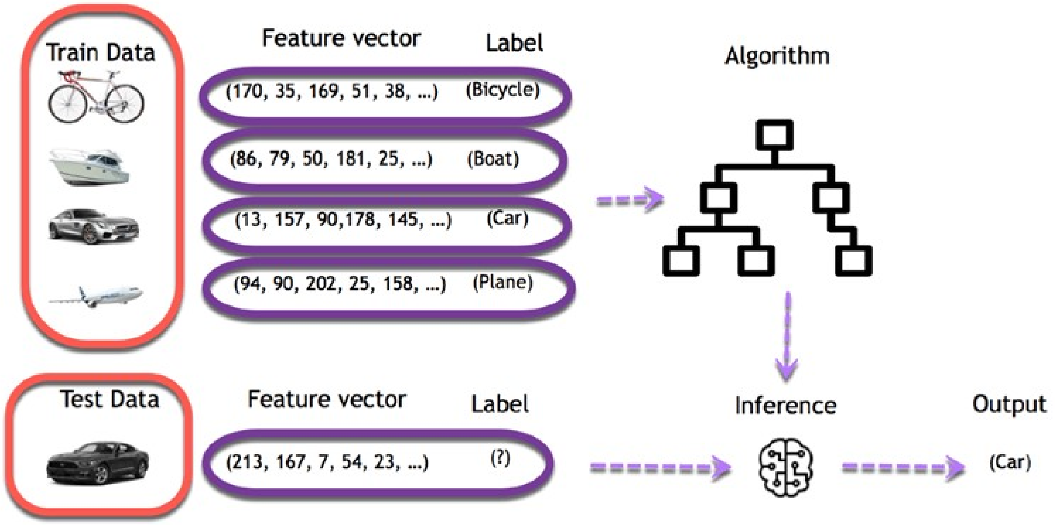
Supervised Learning Setup

There are multiple scenarios for the label space $\boldsymbol{C}$:

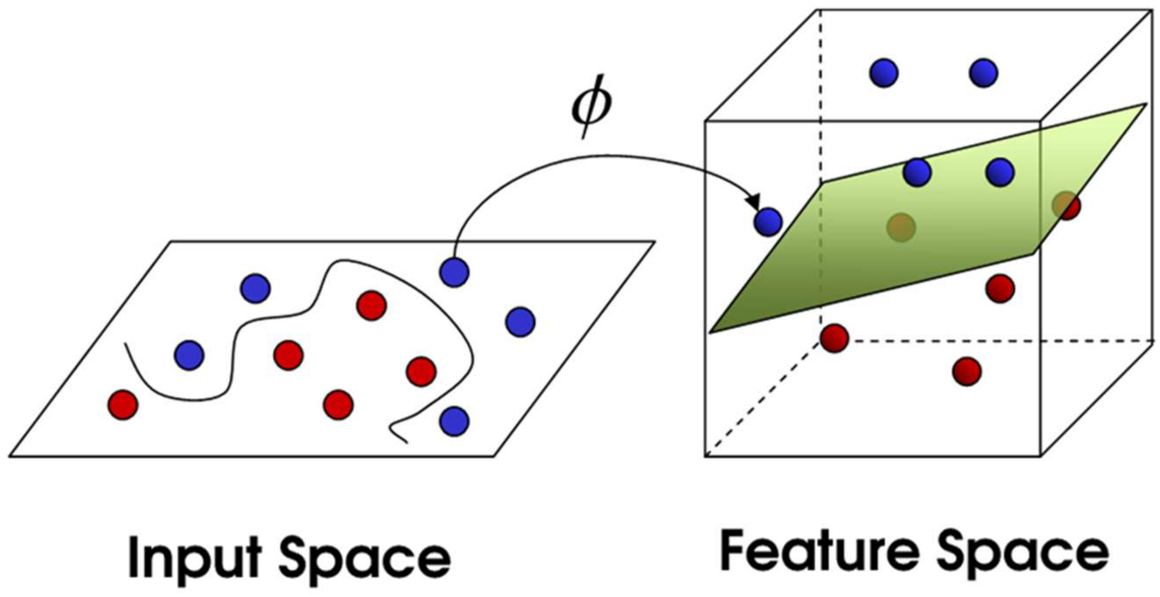
Binary Classification	$C = \{\mathbf{0}, \mathbf{1}\}\ \boldsymbol{or}\ \{+\mathbf{1}, -\mathbf{1}\}$	spam filtering. An email is either spam (+1+1), or not (−1−1).
Multi-class classification	$C = \{1,2,...K)\ \ K \geq 2$	face classification. A person can be exactly one of K identities
Regression	$C = \mathcal{R}$	predict future temperature or the height of a person.

Feature Space





Feature Space



Hypothesis classes and No Free Lunch

Before we can find a function h, we must specify what type of function it is that we are looking for. It could be an artificial neural network, a decision tree or many other types of classifiers. We call the set of possible functions the hypothesis class.

By specifying the hypothesis class, we are encoding important assumptions about the type of problem we are trying to learn.

The No Free Lunch Theorem states that every successful ML algorithm must make assumptions. This also means that there is no single ML algorithm that works for every setting.

Supervised (Inductive) Learning

- Simplest form: learn a function from examples
  -
  - *f* is the target function
  - An example is a pair (x, $f(x)$)

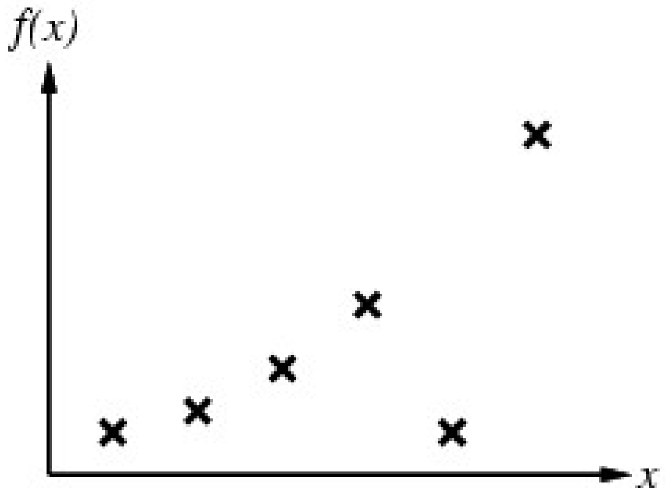
- Pure induction task:
  - **Given a collection of examples of *f*, return a function *h* that approximates *f*.**
  - find a hypothesis *h*, such that $h \approx f$, given a training set of examples

- This is a highly simplified model of real learning:
  - Ignores prior knowledge
  - Assumes examples are given

Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

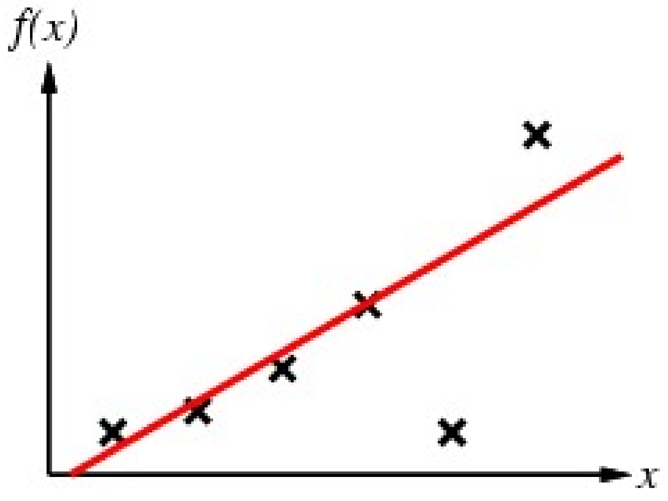
  e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

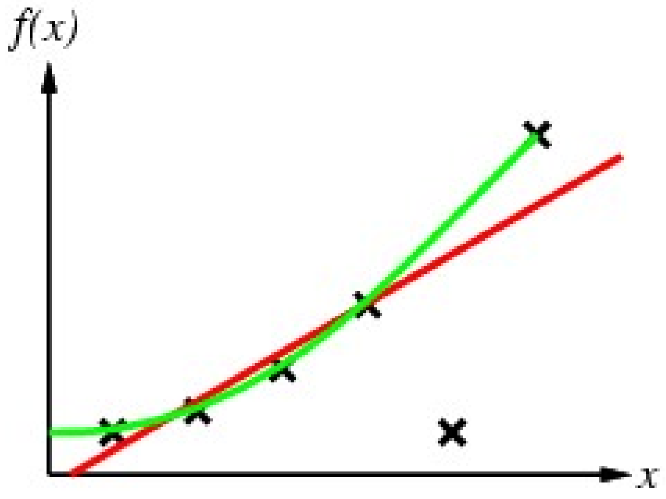
  e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

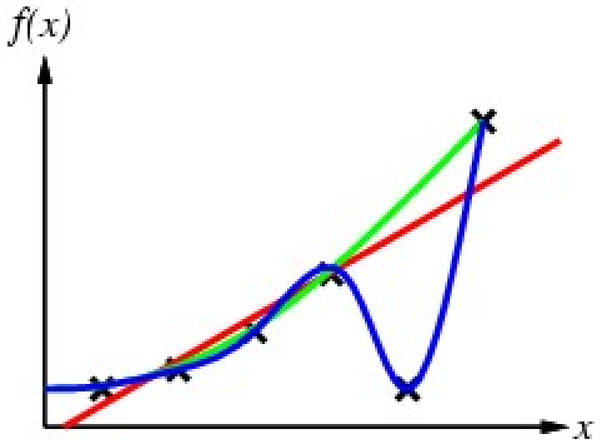
  e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

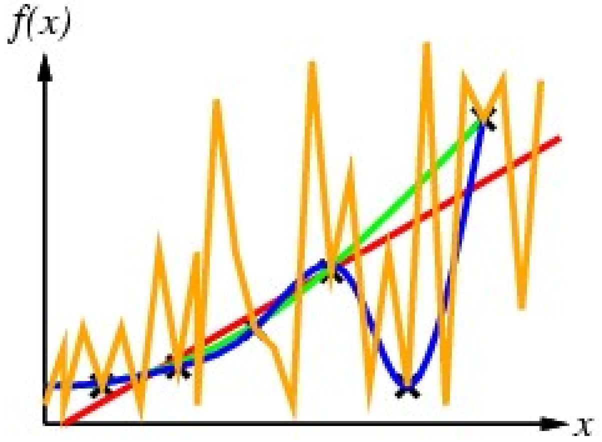
  e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

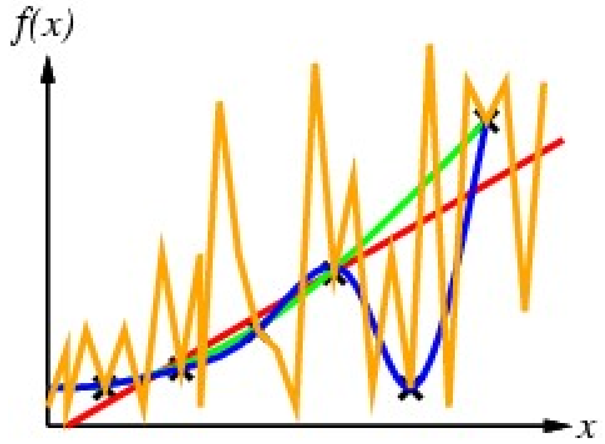
  e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

  e.g., curve fitting:



- Ockham's razor: prefer the simplest hypothesis consistent with data

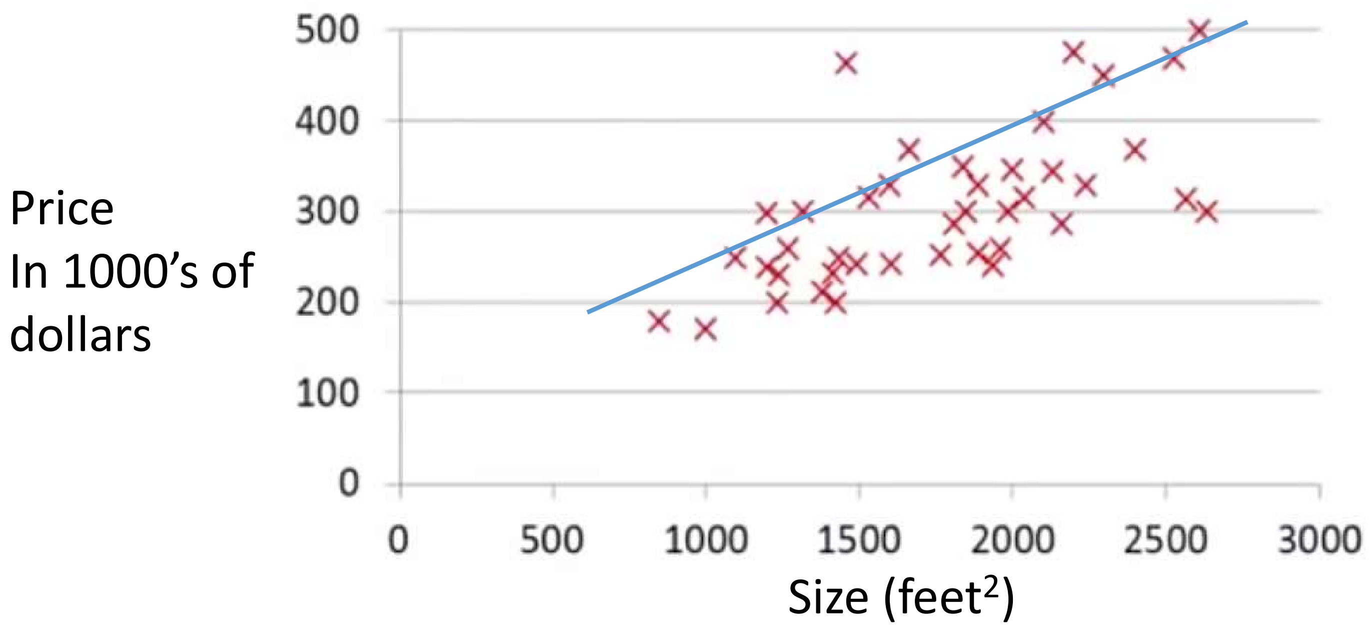
Supervised Learning

- Learning a continuous function: **Regression**

- Learning a discrete function: **Classification**
  - Boolean classification:
    - Each example is classified as true(positive) or false(negative).

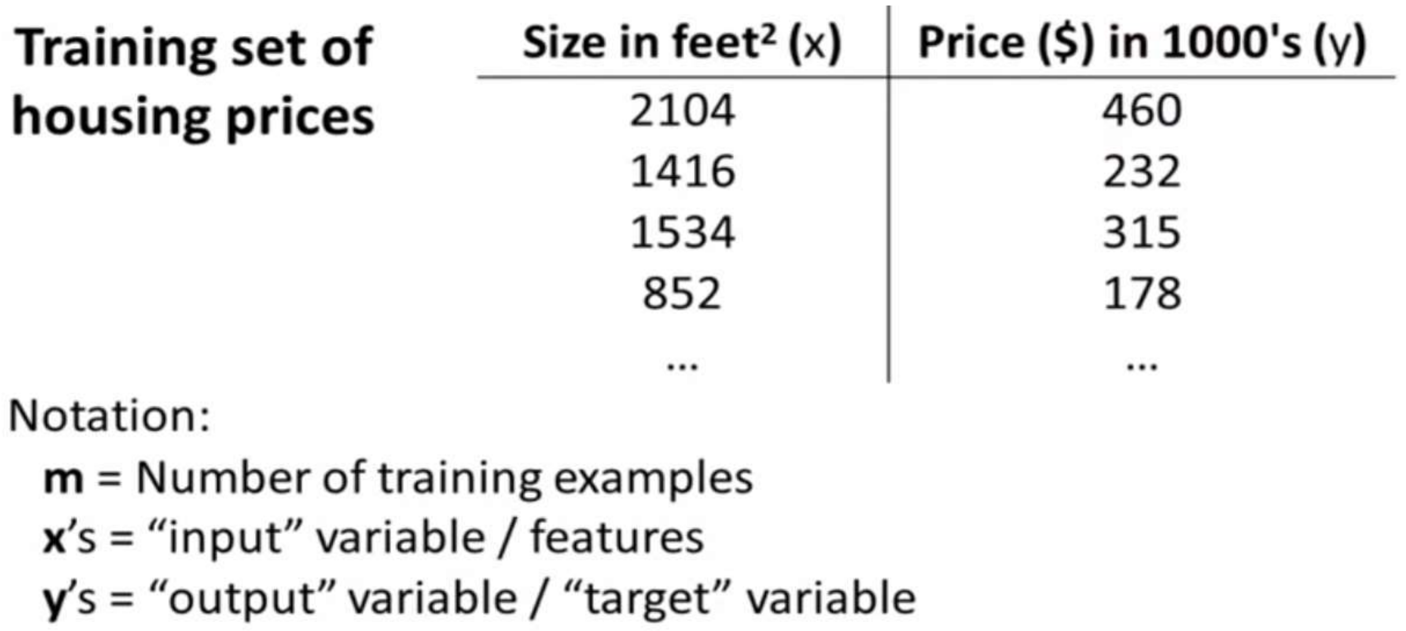
Supervised Learning: Linear Regression

Assumption: Hypothesis function is linear



- Given the right answer for each example of the data
  - Regression: Predict real valued data

Supervised Learning: Linear Regression



Training set of housing prices

Size in feet^2 (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

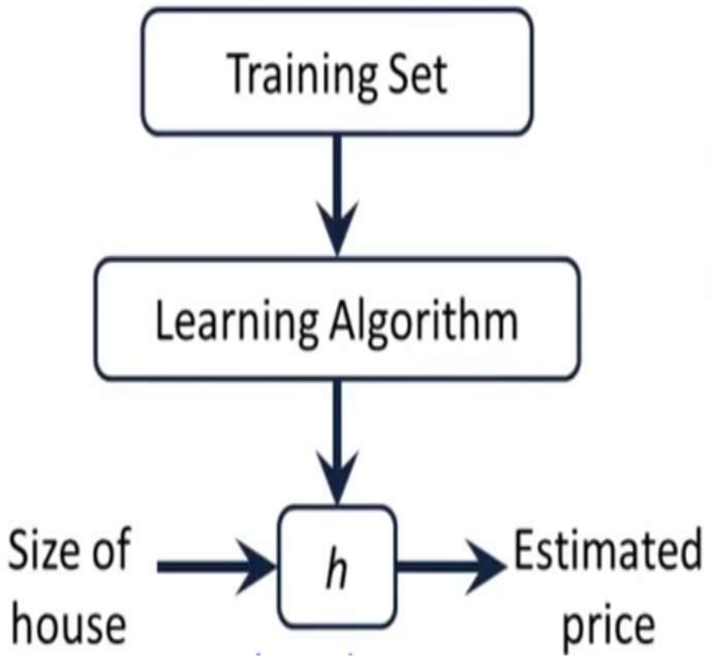
Notation:

- **m** = Number of training examples
- **x**'s = "input" variable / features
- **y**'s = "output" variable / "target" variable

$(x,y) \rightarrow$ one training example $\qquad x^{(i)} = 2104$

$(x^{(i)},y^{(i)}) \rightarrow$ ith training example $\qquad y^{(i)} = 460$

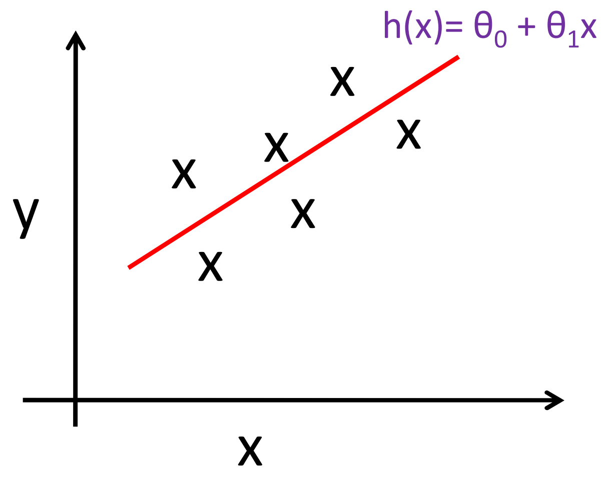
Supervised Learning: Linear Regression



x hypothesis Estimated value of y

How do we represent h

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



Univariate linear regression: linear regression with one variable

Linear Regression: Cost Function

Training Set

Size in feet^2 (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

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

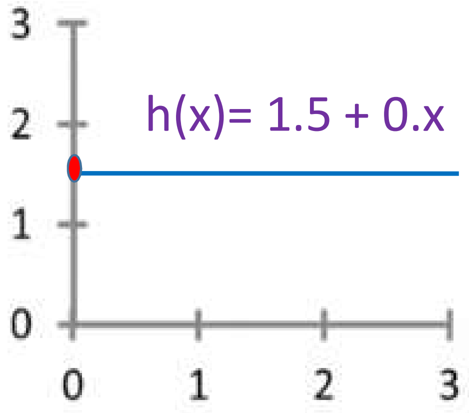
θ_1's $\rightarrow$ Parameters

How to choose θ_1's

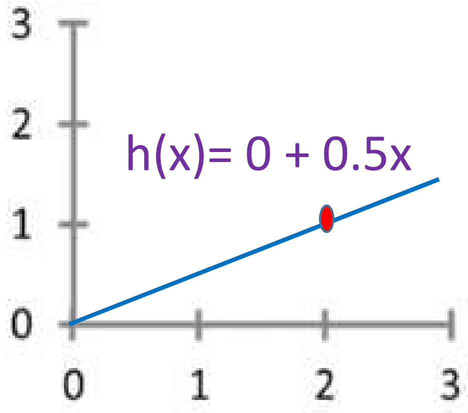
Cost Function

Hypothesis Function:

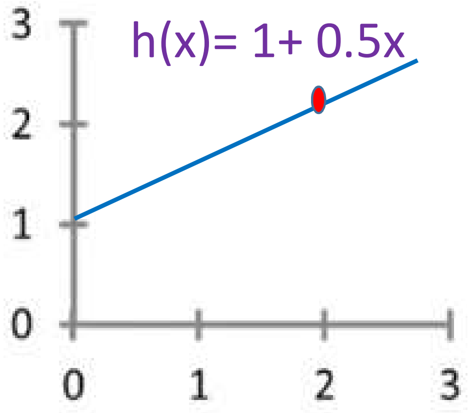
$h_\theta(x) = \theta_0 + \theta_1 x$



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

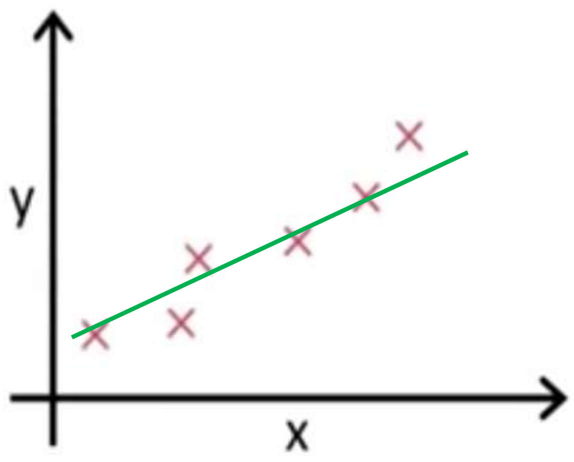


$\theta_0 = 0$
$\theta_1 = 0.5$



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

Cost Function



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

Parameters: θ_0, θ_1

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

Squared error function

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

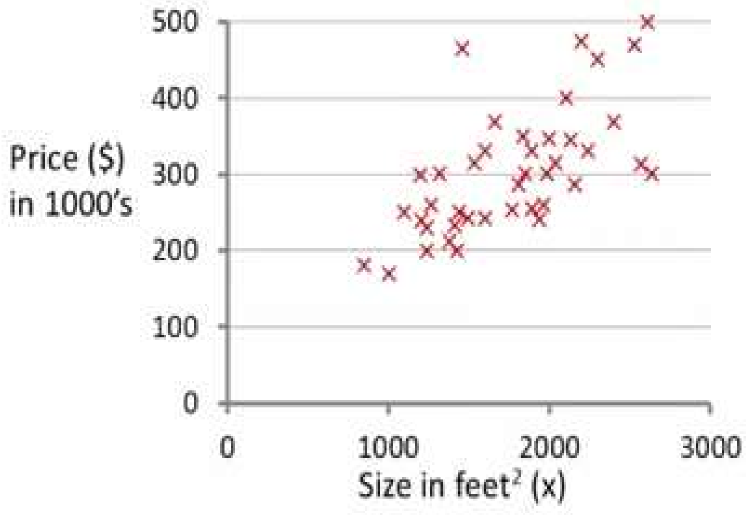
Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

m = No. of training samples

Cost Function

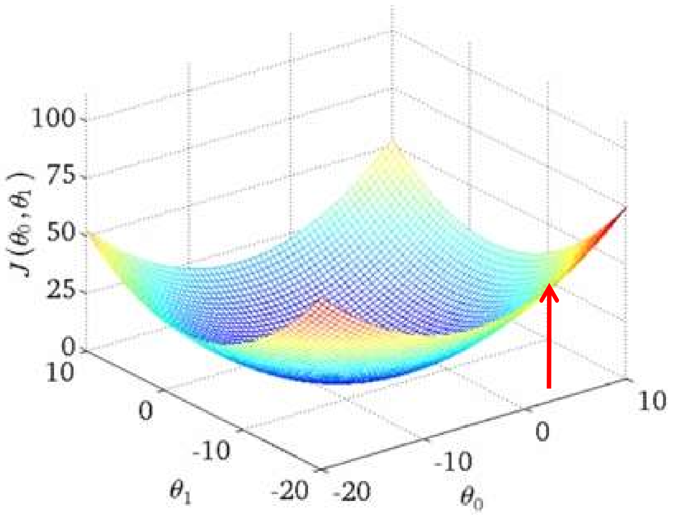
$h_\theta(x)$

(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)

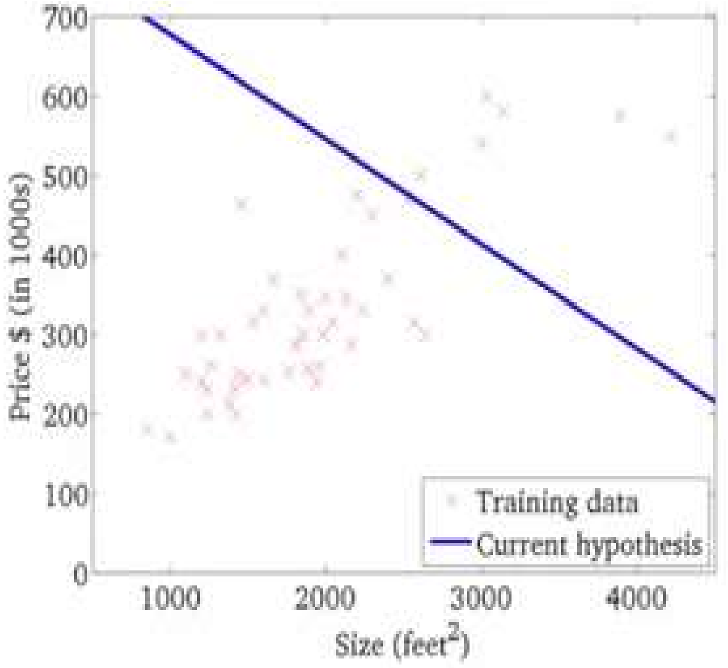


$J(\theta_0, \theta_1)$= value of the height of the surface

Contour Plots / Figures

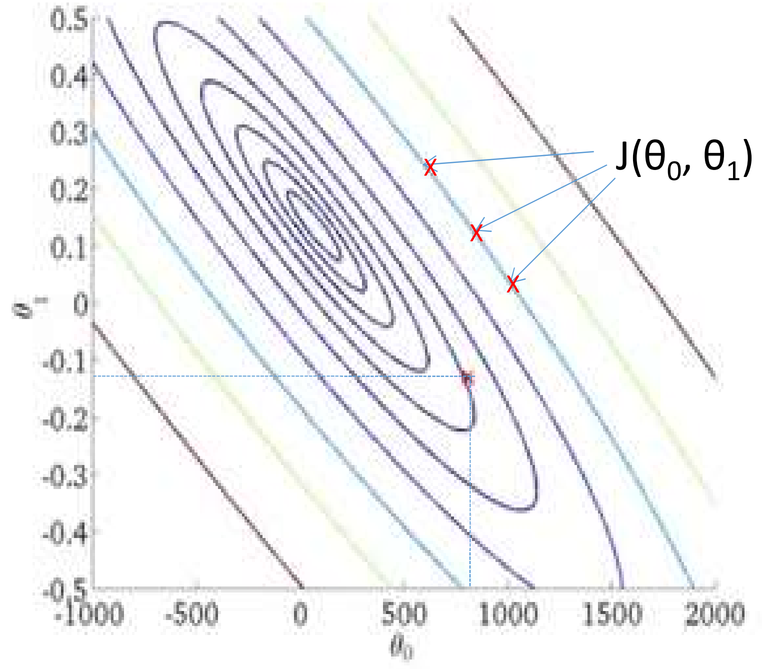
$h_\theta(x)$

(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)

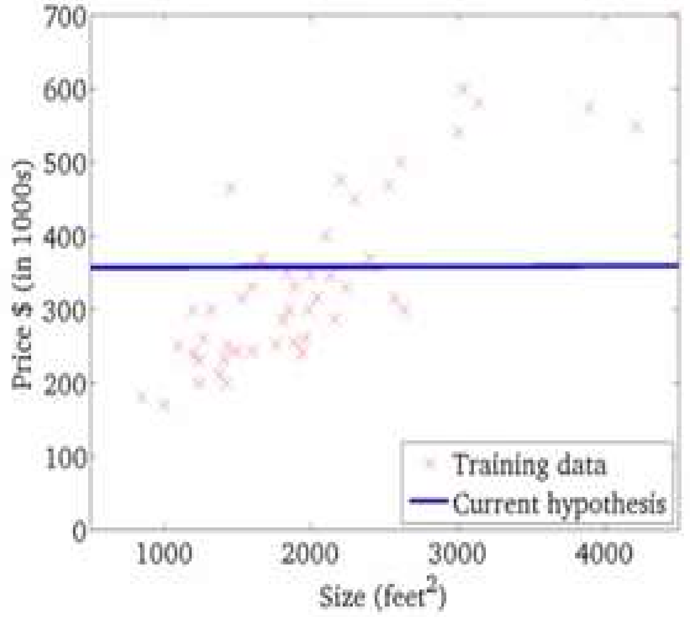


$(\theta_0, \theta_1) = (800, -0.125)$

Contour Plots / Figures

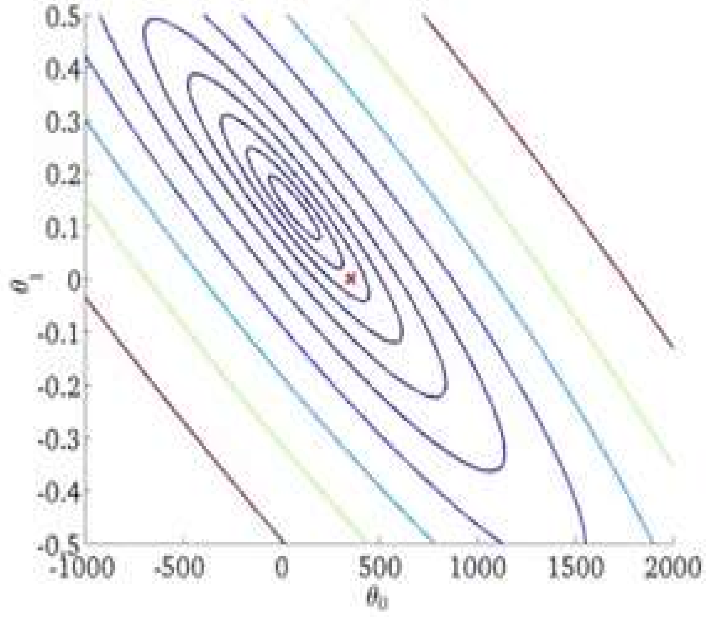
$h_\theta(x)$

(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)

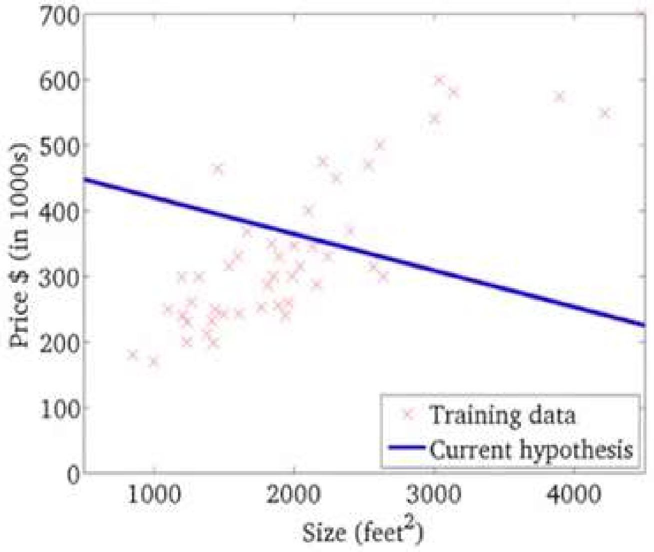


$(\theta_0, \theta_1) = (360, 0)$

Contour Plots / Figures

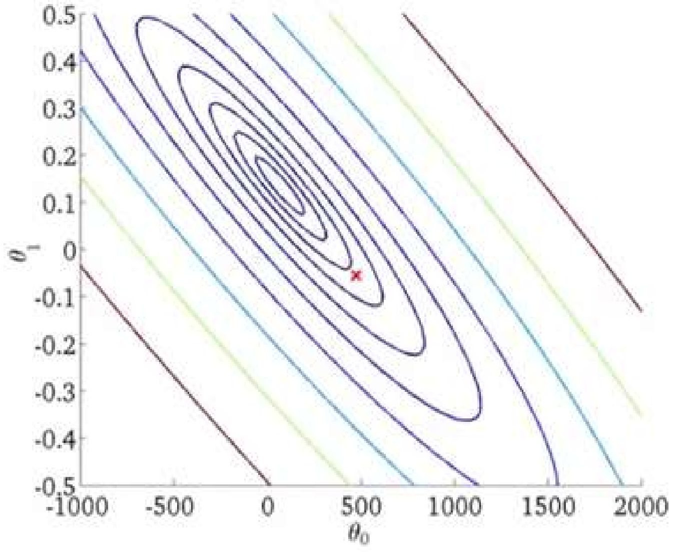
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

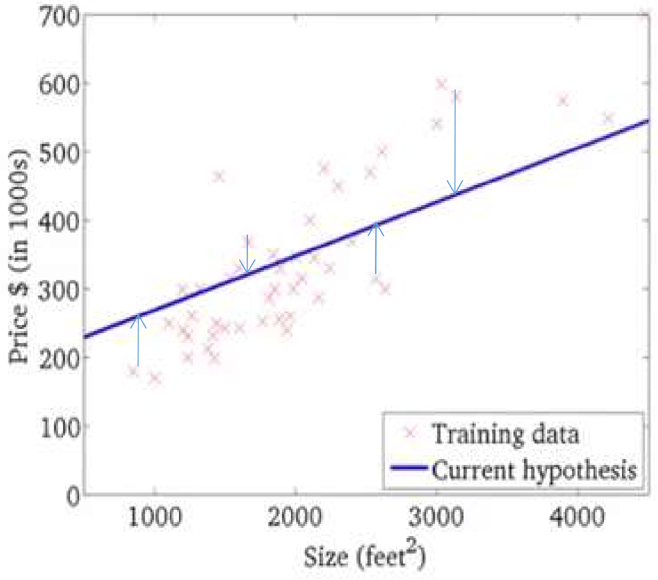
(function of the parameters θ_0, θ_1)



Contour Plots / Figures

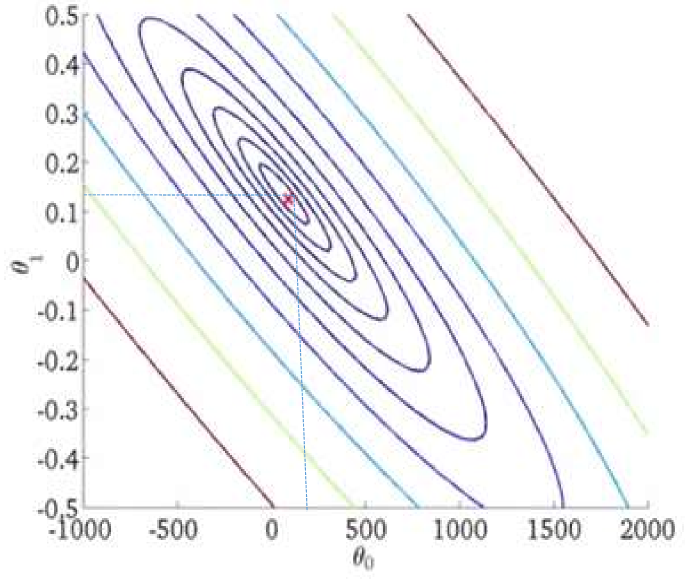
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

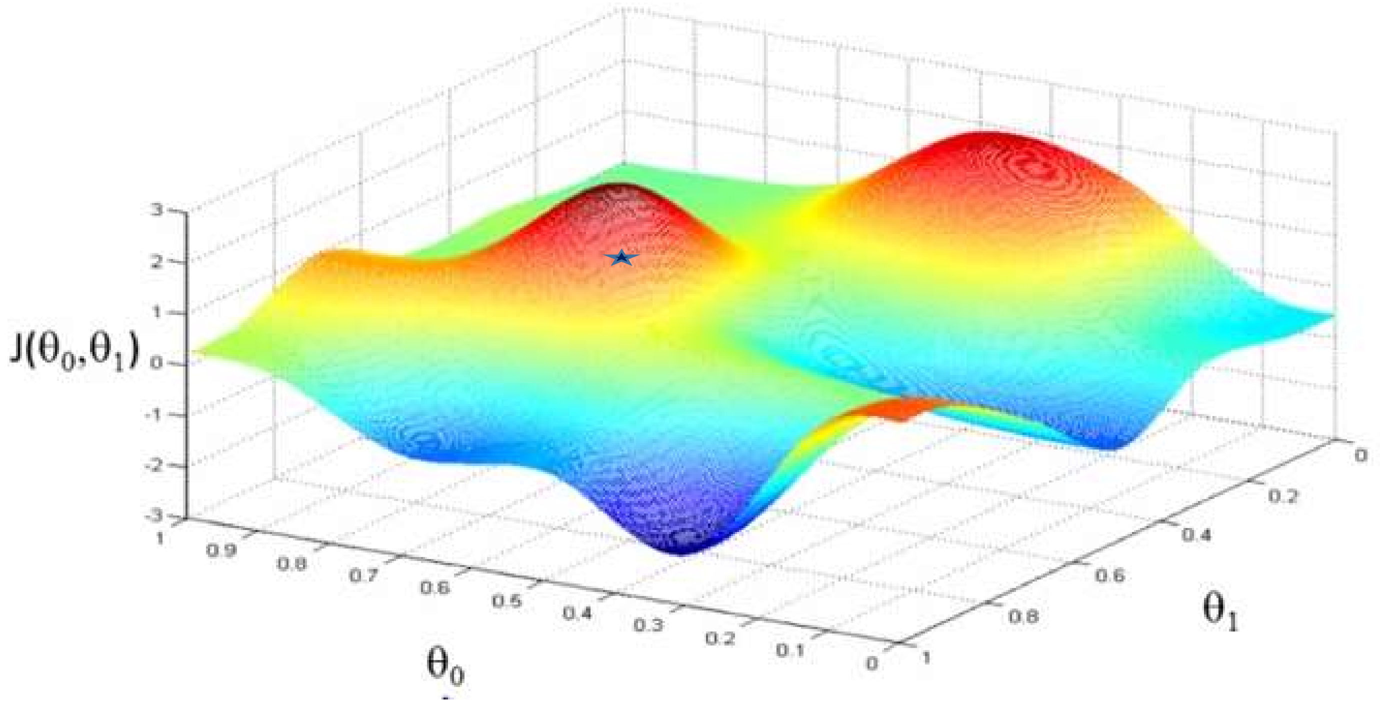
(function of the parameters θ_0, θ_1)



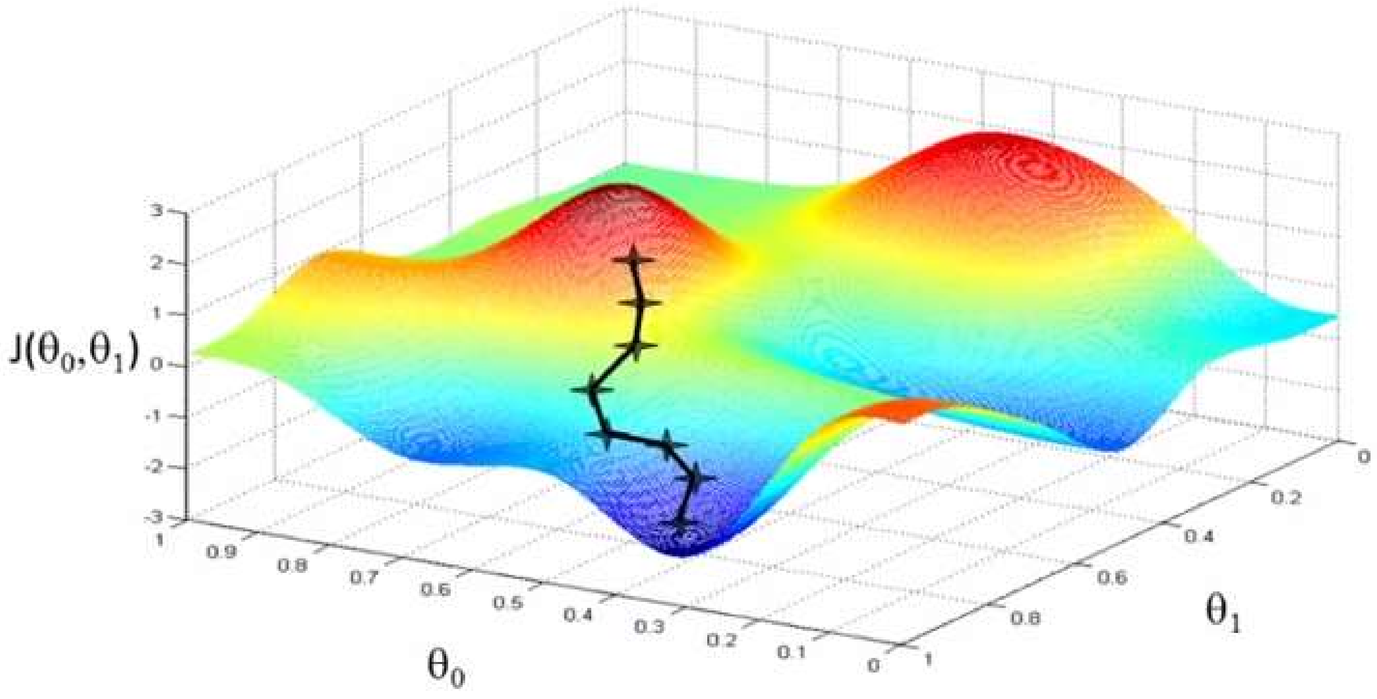
Gradient Descent

- Let some function $J(\theta_0, \theta_1)$
- We have to find $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$
- Start with some (θ_0, θ_1) (let say $\theta_0=0$, $\theta_1=0$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum

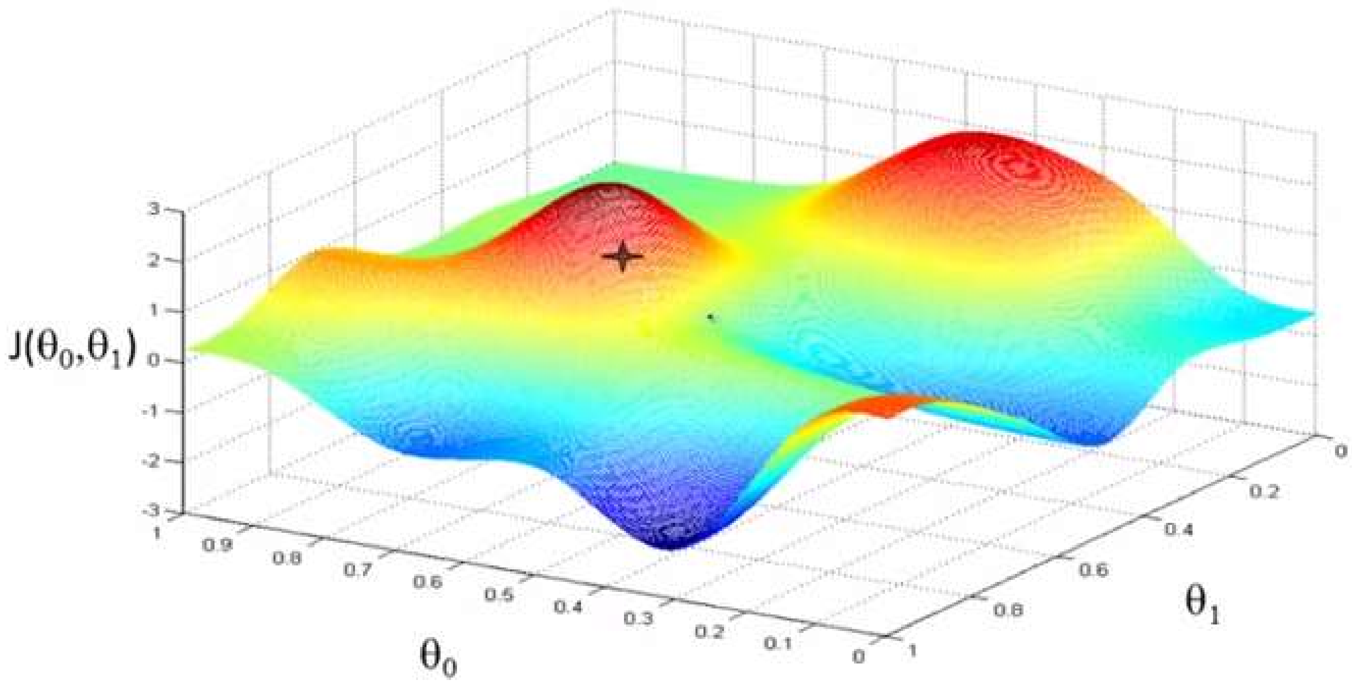
Gradient Descent



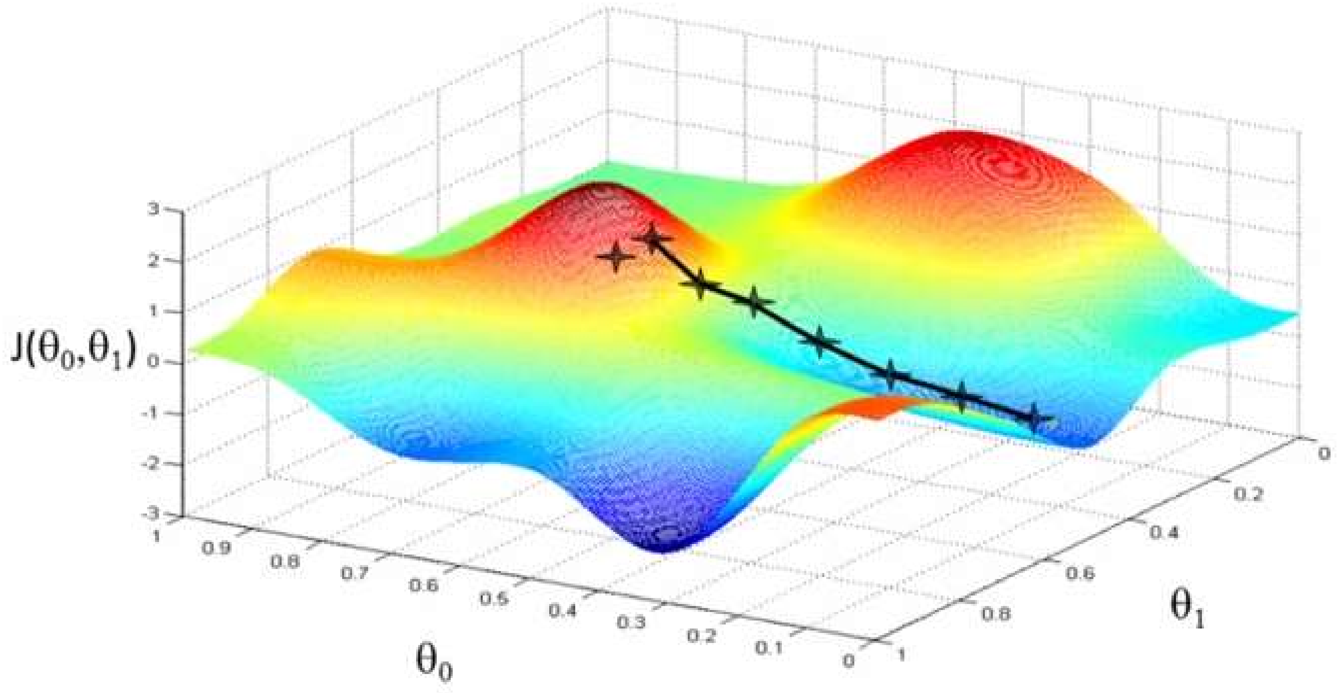
Gradient Descent



Gradient Descent



Gradient Descent



Gradient Descent Algorithm

```
repeat until convergence {
    $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$    (for $j = 0$ and $j = 1$)
}
```

α = learning rate

Implication of α = it controls how bigger steps we are taking over gradient descent

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \tfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \tfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

$$\text{temp0} := \theta_0 - \alpha \tfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \tfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

Gradient Descent Algorithm

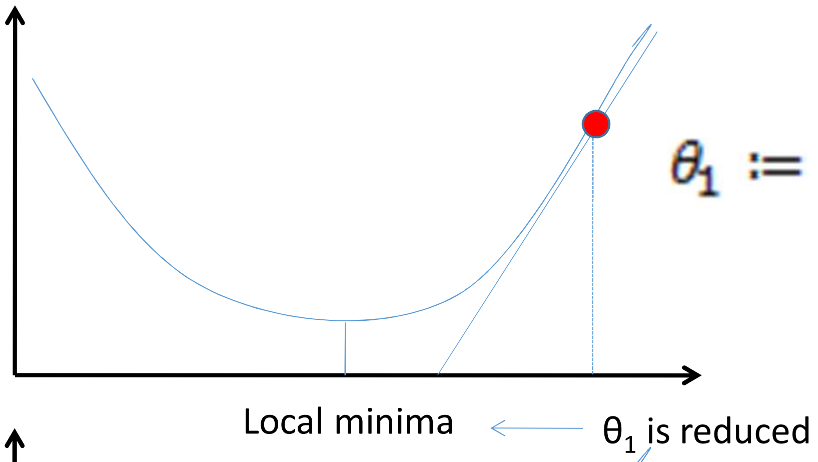
- Let take a single variable
- we have to minimize $\min_{\theta_1} J(\theta_1)$

where $\theta_1 \in R$

- So the GD algorithm becomes

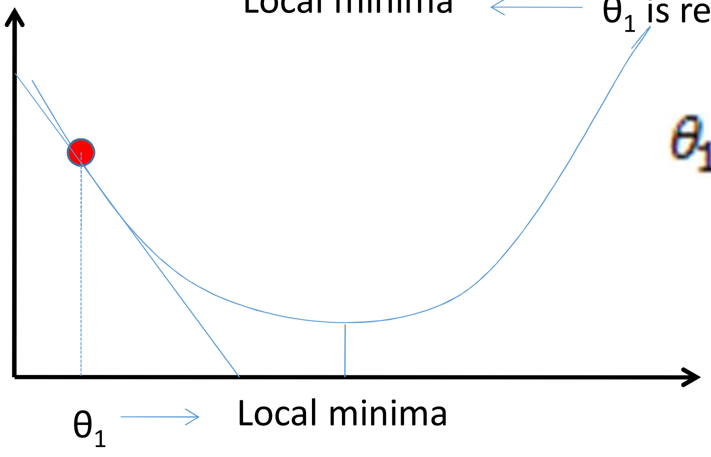
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

Gradient Descent Algorithm



$$\theta_1 := \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)}$$

≥ 0
(slope is positive)



$$\theta_1 := \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)}$$

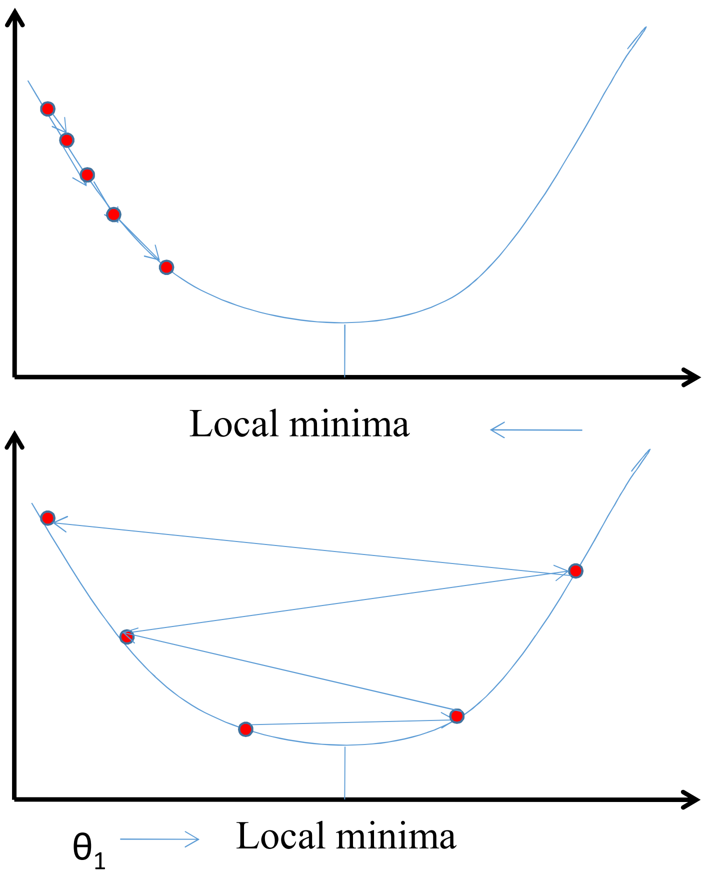
≤ 0
(slope is negative)

Gradient Descent Algorithm

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.

If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Multivariate Linear Regression

Univariate Hypothesis function:

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

Multivariate Hypothesis function:

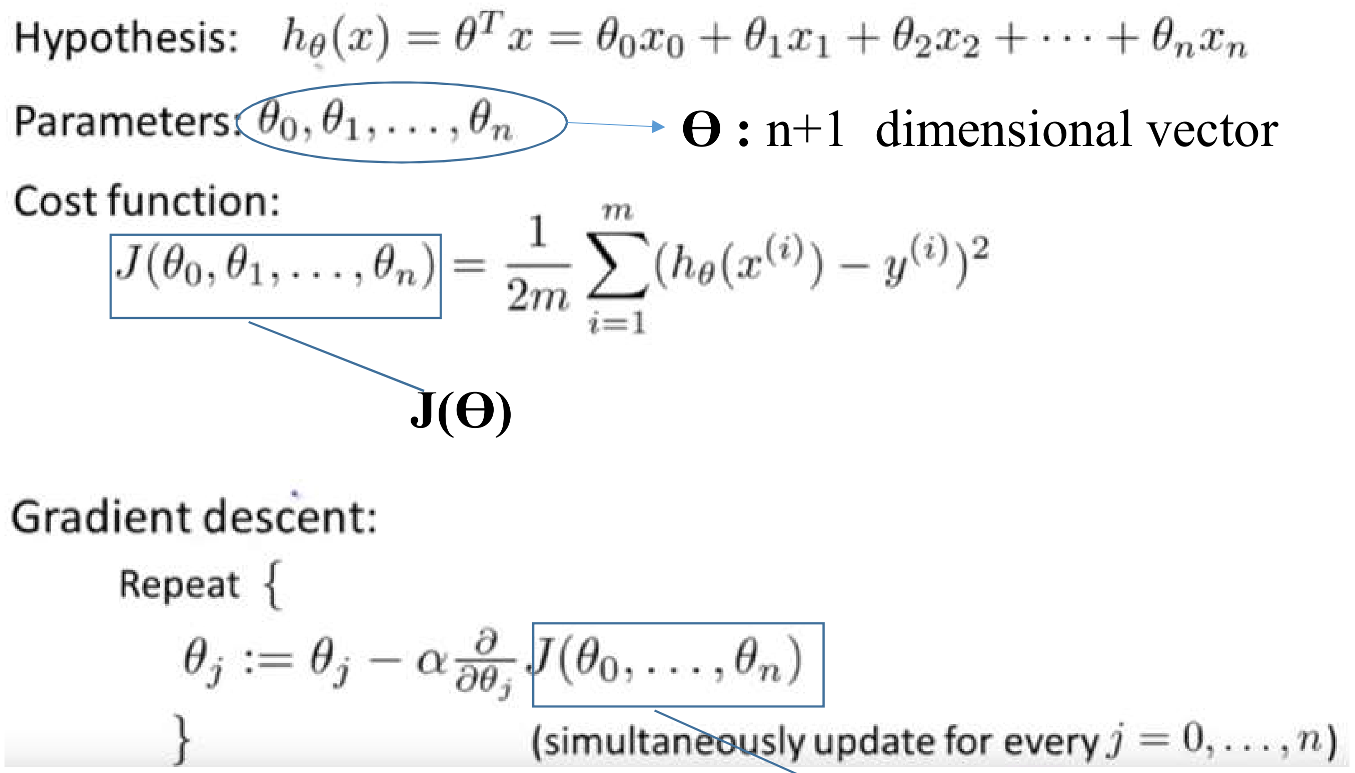
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

$$\mathrm{X} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} € \mathbb{R}^{n+1} \quad \Theta = \begin{bmatrix} \Theta_0 \\ \Theta_1 \\ \Theta_2 \\ \vdots \\ \Theta_n \end{bmatrix} € \mathbb{R}^{n+1} \qquad \Theta^T x = \begin{bmatrix} \Theta_0 & \Theta_1 & \cdots & \Theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

where $x_0 = 1$

$$\boxed{h_\Theta(\mathrm{x}) = \Theta^T x}$$

Multivariate Gradient Descent



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

Parameters: $\boxed{\theta_0, \theta_1, \ldots, \theta_n} \rightarrow$ **Θ :** n+1 dimensional vector

Cost function:

$$\underbrace{\boxed{J(\theta_0, \theta_1, \ldots, \theta_n)}}_{\mathbf{J(\Theta)}} = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

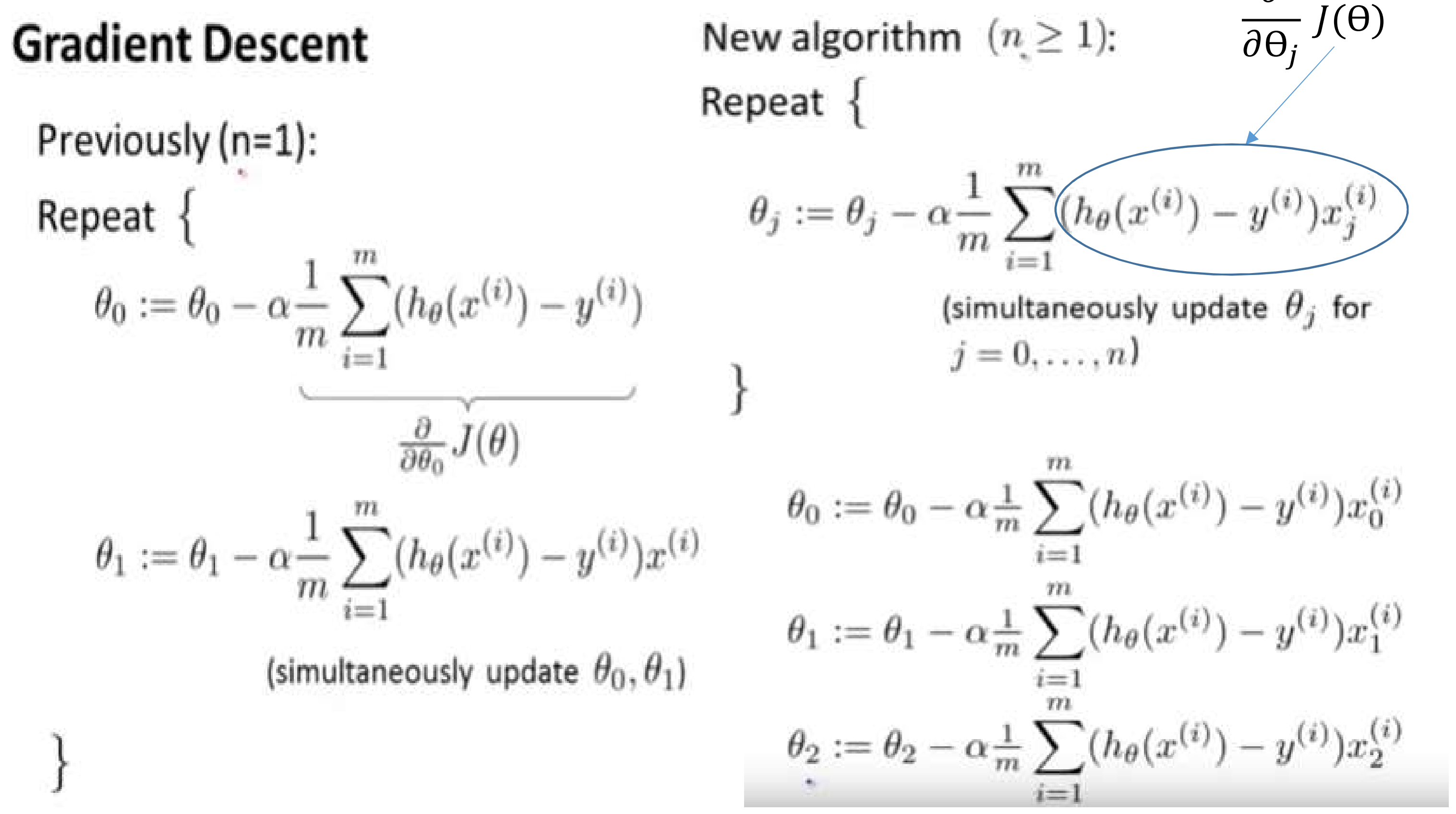
Gradient descent:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \underbrace{\boxed{J(\theta_0, \ldots, \theta_n)}}_{\mathbf{J(\Theta)}}$$

} (simultaneously update for every $j = 0, \ldots, n$)

Multivariate Gradient Descent



Gradient Descent

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial\theta_0}J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

(simultaneously update θ_0, θ_1)

}

New algorithm $(n \geq 1)$:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}\overbrace{(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}}^{\frac{\partial}{\partial\Theta_j}J(\Theta)}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

}

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

Feature Scaling

Feature Scaling

Idea: Make sure features are on a similar scale.

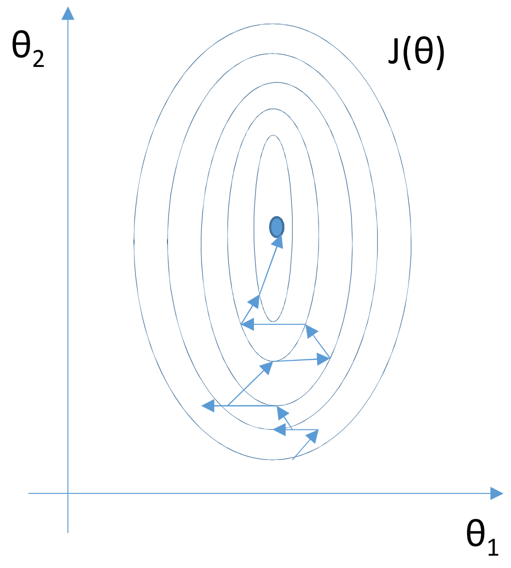
E.g. x_1 = size (0-2000 feet^2)

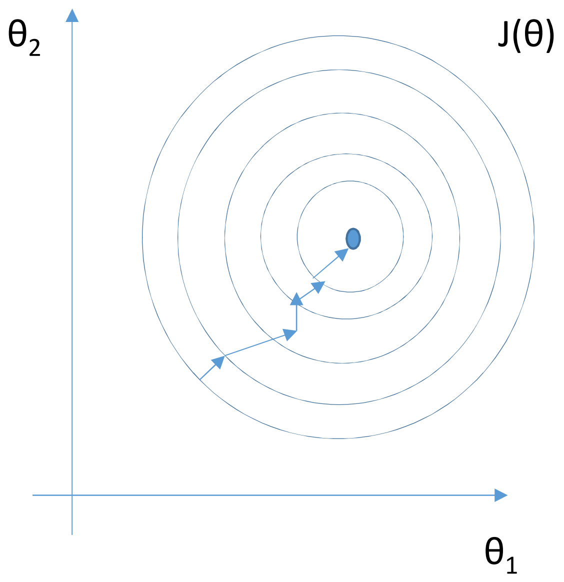
x_2 = number of bedrooms (1-5)

$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$

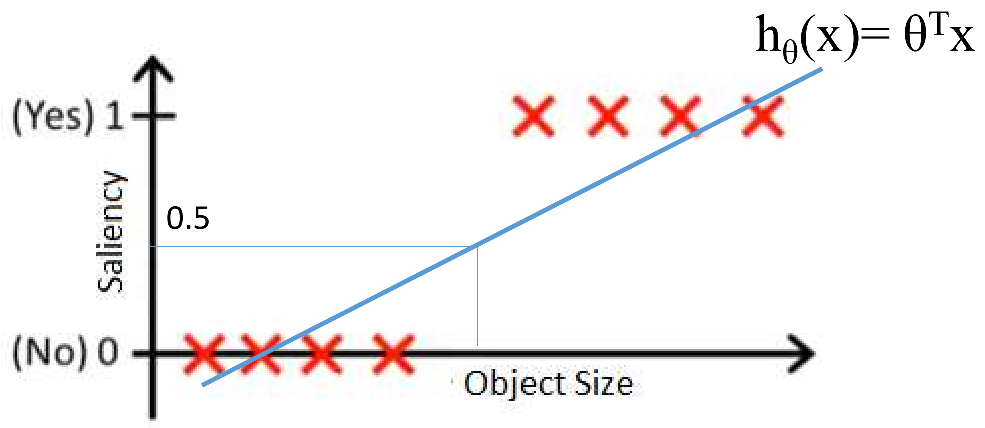
$0 \leq x_1 \leq 1$ $\quad$ $0 \leq x_2 \leq 1$



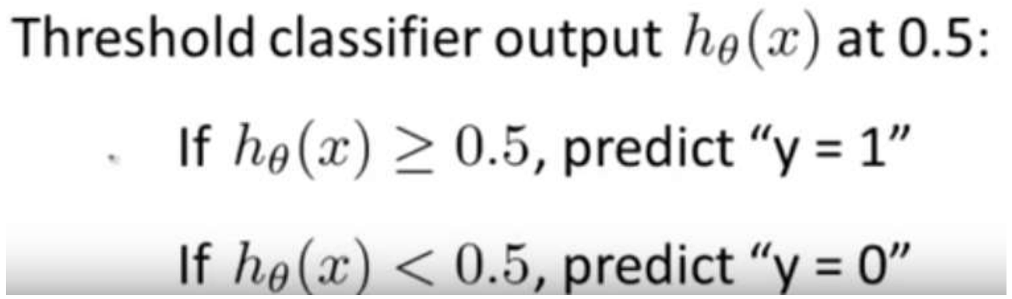


Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

Logistic Regression: Classification



$h_\theta(x)= \theta^T x$

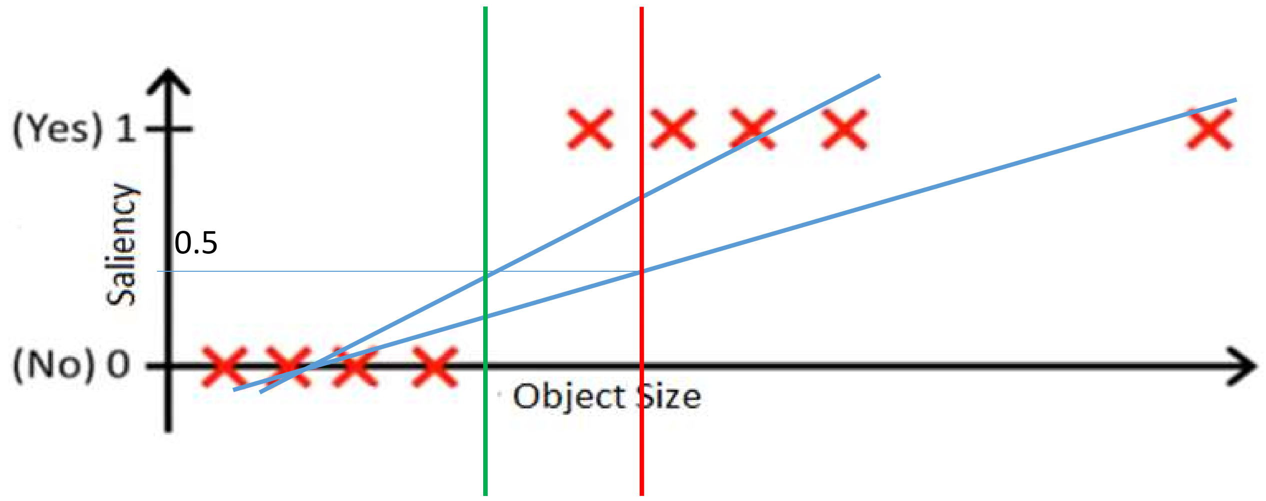


Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"

Logistic Regression



Linear regression for classification problem is not always good

Classification: y = 0 or 1

$h_\theta(x)$ can be > 1 or < 0

Logistic Regression: $0 \le h_\theta(x) \le 1$

Logistic Regression Model

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

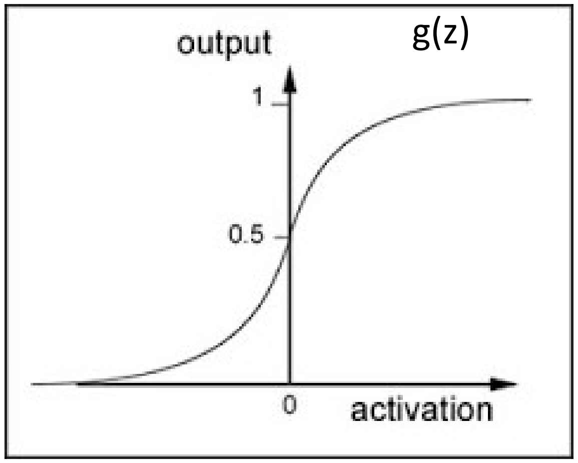
Linear Regression: $h_\theta(x) = \theta^T x$

Logistic Regression:

$$h_\Theta(x) = g(\Theta^T x) \qquad h_\Theta(x) = \frac{1}{1 + e^{-\Theta^T x}}$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid Function or Logistic function



Hypothesis Representation

$h_\Theta(x) =$ estimated probability that y=1 on input x

Example: if $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ Object\ Size \end{bmatrix}$

$h_\Theta(x) = 0.7$

There is 70% chance that the object is salient

$h_\Theta(x) =$ p(y=1|x, Θ)

i.e. “probability that y=1, given x, parameterized by Θ”

p(y=0|x; Θ) + p(y=1|x; Θ) = 1

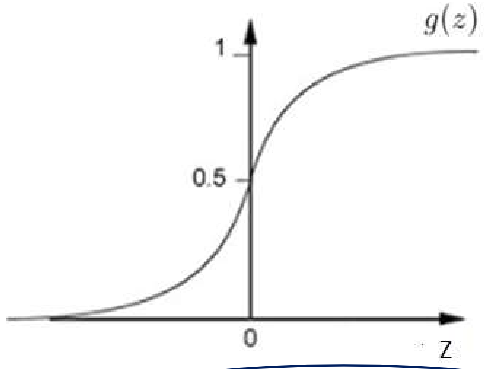
p(y=0|x; Θ) = 1 - p(y=1|x; Θ)

Decision Boundary

Logistic regression

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$



Suppose predict "$y = 1$" if $h_\theta(x) \geq 0.5$

$g(z) \geq 0.5 \quad when \quad z \geq 0$

i.e. $\Theta^T x \geq 0$

$h_\Theta(x) = g(\Theta^T x) \geq 0.5$

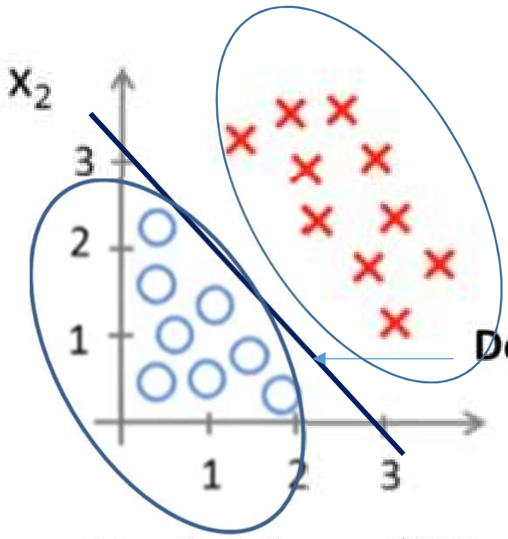
whenever $\underbrace{\Theta^T x}_{\boxed{z}} \geq 0$

predict "$y = 0$" if $h_\theta(x) < 0.5$

$h_\Theta(x) = g(\Theta^T x)$

i.e. $\Theta^T x < 0$

Decision Boundary



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

$$\Theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

$$x_1 + x_2 = 3$$

Decision Boundary

$$h_\Theta(x) = g(\Theta^T x) = 0.5$$

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

i.e. $\Theta^T x \geq 0$

$$x_1 + x_2 \geq 3$$

Predict "y = 0" if

$$x_1 + x_2 < 3$$

Decision boundary is a property of hypothesis function NOT of a data set

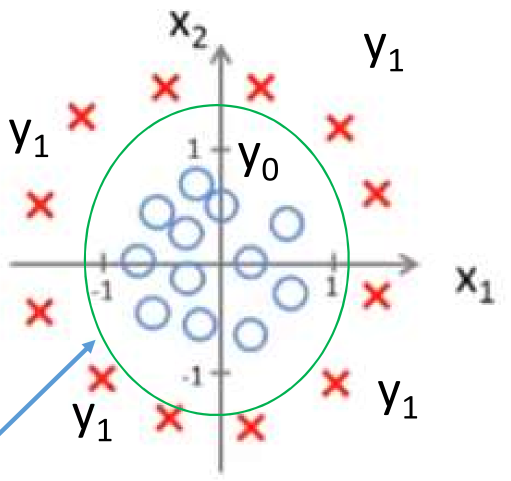
Non-Linear Decision Boundary

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Let $\Theta^T = [-1 \quad 0 \quad 0 \quad 1 \quad 1]$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

$$x_1^2 + x_2^2 \geq 1$$



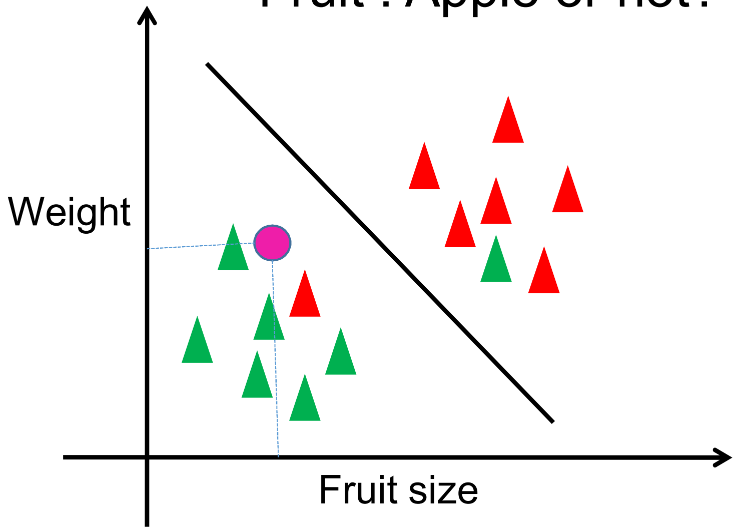
$$x_1^2 + x_2^2 = 1$$

Decision Boundary

Again, decision boundary is a property of hypothesis function NOT of a data set

Supervised Learning: Classification

Fruit : Apple or not?



Two features
examples

- Features:
  - color
  - texture
  - cost
  - …..

- Given the right answer for each example of the data
  - Classification: discrete no. of outputs

Classification—A Two-Step Process

- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label
  - The set of tuples used for model construction is training set
  - The model is represented as **classification rules**, **decision trees**, or **mathematical formulae**

- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - **Test set is independent of training set**, otherwise over-fitting will occur
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known

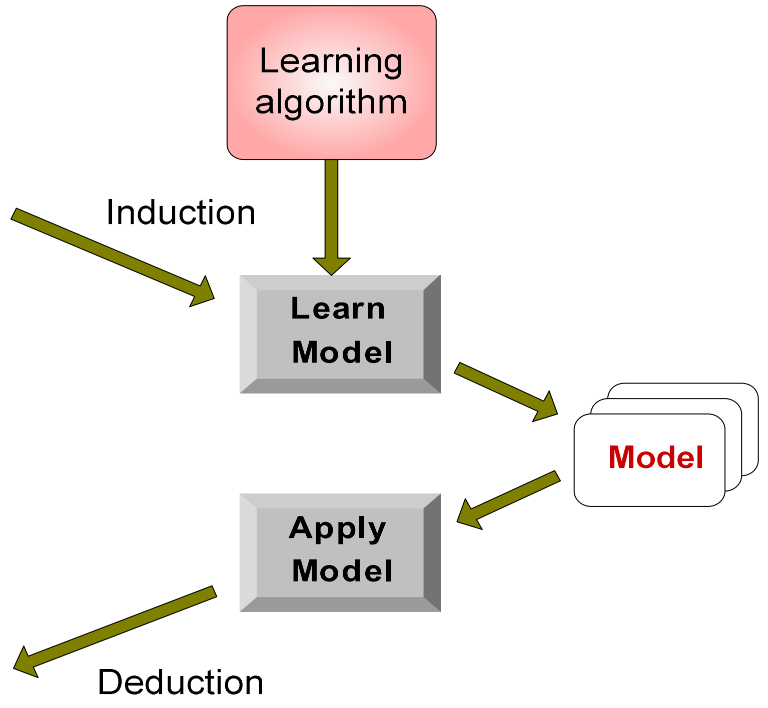
Illustrating Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Loss Function

- There are typically two steps involved in learning a hypothesis function h().
- First, we select the type of machine learning algorithm that we think is appropriate for this particular learning problem. This defines the hypothesis class $\mathcal{H}$, i.e. the set of functions we can possibly learn.
- The second step is to find the best function within this class, h$\in \mathcal{H}$. This second step is the actual learning process and often, but not always, involves an optimization problem. Essentially, we try to find a function h within the hypothesis class that makes the fewest mistakes within our training data.
- If there is not a single function we typically try to choose the "simplest" by some notion of simplicity - but we will cover this in more detail in a later class.
- How can we find the best function? For this we need some way to evaluate what it means for one function to be better than another. This is where the loss function (aka risk function) comes in.
- A loss function evaluates a hypothesis h$\in \mathcal{H}$ on our training data and tells us how bad it is. The higher the loss, the worse it is - a loss of zero means it makes perfect predictions. It is common practice to normalize the loss by the total number of training samples, n, so that the output can be interpreted as the average loss per sample (and is independent of n).

Loss Function: Example

- Zero-one loss: The simplest loss function is the zero-one loss.

- It literally counts how many mistakes an hypothesis function h makes on the training set. For every single example it suffers a loss of 1 if it is mispredicted, and 0 otherwise.

- The normalized zero-one loss returns the fraction of misclassified training samples, also often referred to as the training error.

- The zero-one loss is often used to evaluate classifiers in multi-class/binary classification settings but rarely useful to guide optimization procedures because the function is non-differentiable and non-continuous. Formally, the zero-one loss can be stated has:

$$\mathcal{L}_{0/1}(h) = \frac{1}{n}\sum_{i=1}^{n} \delta_{h(\mathbf{x}_i)\neq y_i}, \text{ where } \delta_{h(\mathbf{x}_i)\neq y_i} = \begin{cases} 1, & \text{if } h(\mathbf{x}_i) \neq y_i \\ 0, & \text{o.w.} \end{cases}$$

- This loss function returns the error rate on this data set D. For every example that the classifier misclassifies (i.e. gets wrong) a loss of 1 is suffered, whereas correctly classified samples lead to 0 loss.

Squared Loss

- The squared loss function is typically used in regression settings. It iterates over all training samples and suffers the loss $(h(\mathbf{x}_i) - y_i)^2$

- The squaring has two effects:
  - the loss suffered is always nonnegative;
  - the loss suffered grows quadratically with the absolute mis-predicted amount.

- The latter property encourages no predictions to be really far off (or the penalty would be so large that a different hypothesis function is likely better suited).

Squared Loss

- On the flipside, if a prediction is very close to be correct, the square will be tiny and little attention will be given to that example to obtain zero error.
- For example, if $|h(\mathbf{x}_i) - y_i| = 0.001$, the squared loss will be even smaller, 0.000001, and will likely never be fully corrected.
- If, given an input x, the label y is probabilistic according to some distribution $P(y|\mathbf{x})$ then the optimal prediction to minimize the squared loss is to predict the expected value, i.e. $h(\mathbf{x}) = \mathbf{E}_{P(y|\mathbf{x})}[y]$
- Formally the squared loss is:

$$\mathcal{L}_{sq}(h) = \frac{1}{n}\sum_{i=1}^{n}(h(\mathbf{x}_i) - y_i)^2.$$

Absolute loss

- Similar to the squared loss, the absolute loss function is also typically used in regression settings.
- It suffers the penalties $|h(\mathbf{x}_i) - y_i|$. Because the suffered loss grows linearly with the mis-predictions it is more suitable for noisy data (when some mis-predictions are unavoidable and shouldn't dominate the loss).
- If, given an input x, the label y is probabilistic according to some distribution $P(y|\mathbf{x})$ then the optimal prediction to minimize the absolute loss is to predict the median value, i.e. $h(\mathbf{x}) = \text{MEDIAN}_{P(y|\mathbf{x})}[y]$
- Formally, the absolute loss can be stated as:

$$\mathcal{L}_{abs}(h) = \frac{1}{n}\sum_{i=1}^{n} |h(\mathbf{x}_i) - y_i|.$$

Generalization

Given a loss function, we can then attempt to find the function h that minimizes the loss:

$$h = \text{argmin}_{h \in \mathcal{H}} \mathcal{L}(h)$$

A big part of machine learning focuses on the question, how to do this minimization efficiently.

If you find a function h(·) with low loss on your data D, how do you know whether it will still get examples right that are not in D?

This is a very important issue of ML and we will get back on it later.

Summary

- We train our classifier by minimizing the training loss

$$\text{Learning: } h^*(\cdot) = \text{argmin}_{h(\cdot)\in\mathcal{H}} \frac{1}{|D_{\text{TR}}|} \sum_{(\mathbf{x},y)\in D_{\text{TR}}} \ell(\mathbf{x}, y|h(\cdot)),$$

  Where $\mathcal{H}$ is the hypothetical class (i.e., the set of all possible classifiers h(·)).
  In other words, we are trying to find a hypothesis h which would have performed well on the past/known data.

- We evaluate our classifier on the testing loss:

$$\text{Evaluation: } \epsilon_{\text{TE}} = \frac{1}{|D_{TE}|} \sum_{(\mathbf{x},y)\in D_{\text{TE}}} \ell(\mathbf{x}, y|h^*(\cdot)).$$

- If the samples are drawn i.i.d. from the same distribution P, then the testing loss is an unbiased estimator of the true **generalization loss**

$$\text{Generalization: } \epsilon = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}}[\ell(\mathbf{x}, y|h^*(\cdot))].$$

Cost Function

- Optimization objective of the cost function

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \ldots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters θ ?

Cost Function

Cost function

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

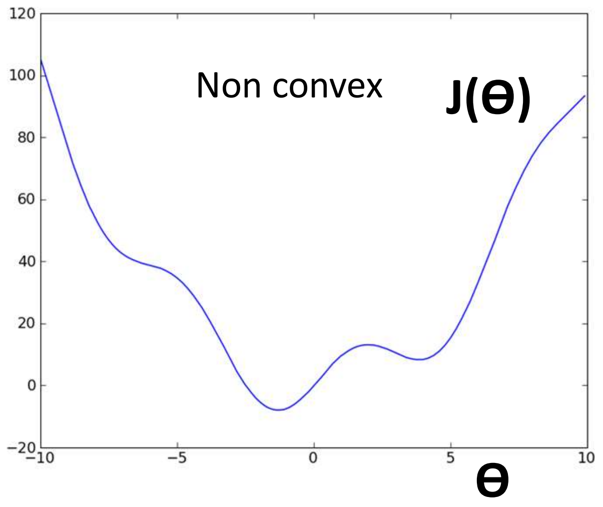
Let, $\text{Cost}\left(h_\Theta\left(x^{(i)}\right), y^{(i)}\right) = \frac{1}{2} \left(h_\Theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

So, $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \theta(x^{(i)}), y^{(i)})$

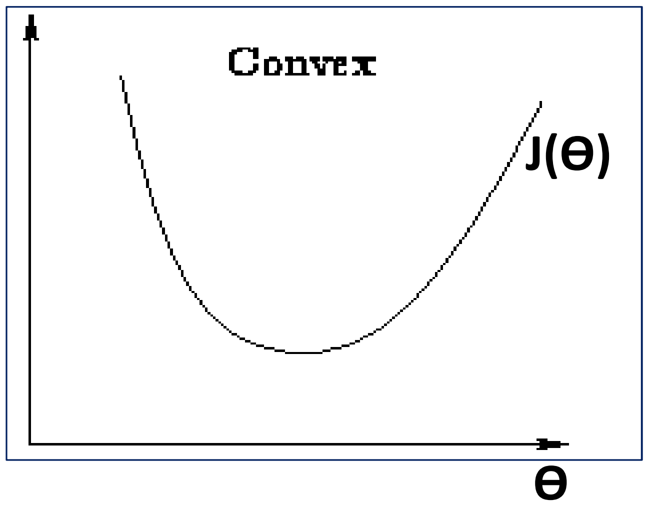
where, for logistic regression

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Cost Function



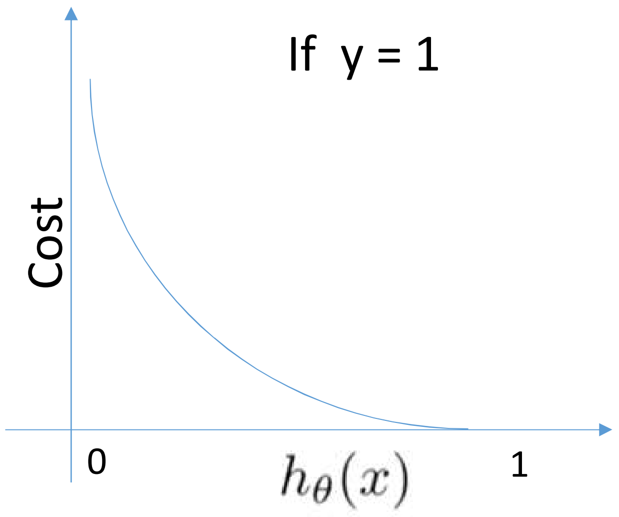
Logistic Regression



Linear Regression

Cost Function: Logistic Regression

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

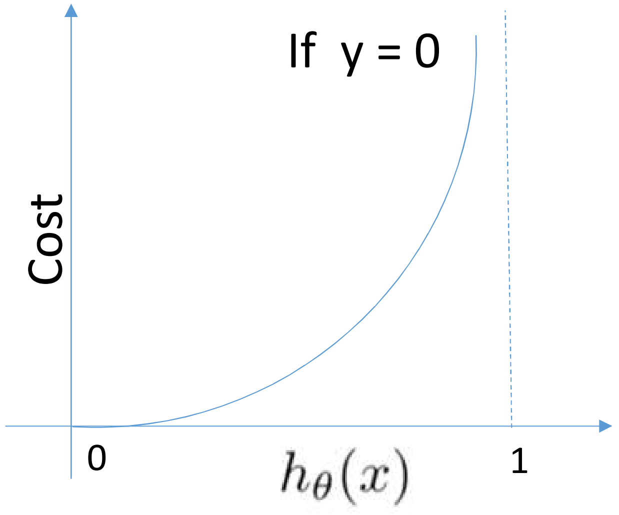


Cost $= 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \rightarrow 0$
$Cost \rightarrow \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x;\theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Cost Function: Logistic Regression

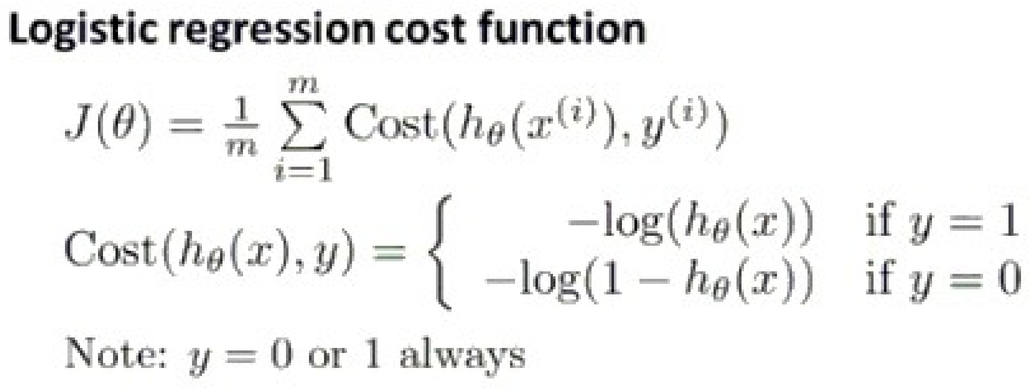
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



Cost =0 if y=0, $h_\Theta(x) = 0$
But as $h_\Theta(x) \rightarrow 1$
Cost $\rightarrow \infty$
Captures intuition that if $h_\Theta(x) = 1$,
(predict $P(y=0|x; \Theta) = 1$), but $y = 0$,
We will penalize learning algorithm
by a very large cost.

It can be shown that the overall cost function is convex function and local optimum free. But details of such convexity analysis is beyond of the scope of this course.

Cost Function: Logistic Regression



Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

$$\text{Cost }(h_\Theta(x), y) = -y\log\big(h_\Theta(x)\big) - (1-y)\log\big(1 - h_\Theta(x)\big)$$

$$\text{If y} = 1: \ \text{Cost }(h_\Theta(x), y) = -\log\big(h_\Theta(x)\big)\big)$$

$$\text{If y} = 0: \ \text{Cost }(h_\Theta(x), y) = -\log\big(1 - h_\Theta(x)\big)$$

Cost Function: Logistic Regression

Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Principle of Maximum Likelihood Estimation

To fit parameters θ:

Obtain $\min_\theta J(\theta)$

and get ϴ

To make a prediction given new x:

Output: $h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

For p(y=1|x; ϴ)

How to minimize J(ϴ) ?

Cost Function and Gradient Descent

Gradient Descent

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))\right]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\}$ (simultaneously update all θ_j)

$$\frac{\partial}{\partial \Theta_j} J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Cost Function and Gradient Descent

Gradient Descent

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))\right]$$

Want $\min_\theta J(\theta)$:

Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (\boxed{h_\theta(x^{(i)})} - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

}

For Linear Regression:

$$h_\Theta(\mathrm{x}) = \Theta^T x$$

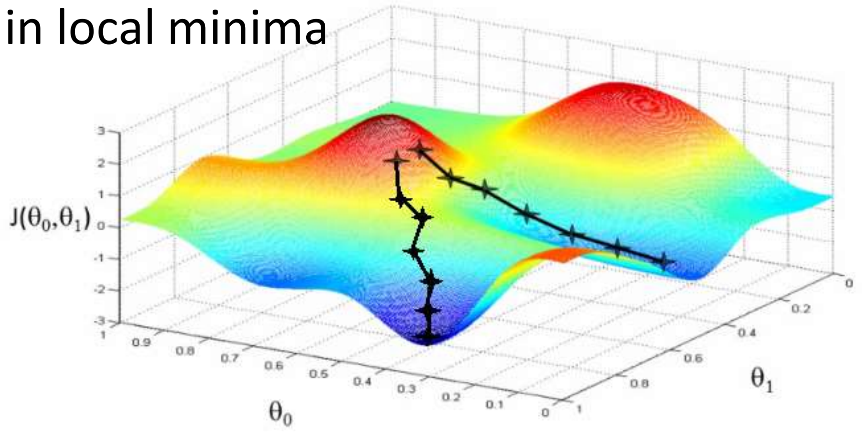
For Logistic Regression:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Algorithm looks identical to linear regression!

Gradient descent optimization

- Problems:
  - Choosing step size
    - too small $\rightarrow$ convergence is slow and inefficient
    - too large $\rightarrow$ may not converge
  - Can get stuck on “flat” areas of function
  - Easily trapped in local minima



Gradient Descent

- Gradient Descent is a popular optimization technique in Machine Learning and Deep Learning, and it can be used with most, if not all, of the learning algorithms.
- A gradient is the slope of a function. It measures the degree of change of a variable in response to the changes of another variable.
- Mathematically, Gradient Descent is a convex function whose output is the partial derivative of a set of parameters of its inputs. The greater the gradient, the steeper the slope.
- Starting from an initial value, Gradient Descent is run iteratively to find the optimal values of the parameters to find the minimum possible value of the given cost function.

Types of Gradient Descent

- Typically, there are three types of Gradient Descent:
    - Batch Gradient Descent
    - Stochastic Gradient Descent
    - Mini-batch Gradient Descent

Stochastic Gradient Descent (SGD)

- The word ‘*stochastic*‘ means a system or a process that is linked with a random probability.
- Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.
- In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration.
- In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset.
- Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big.
- Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

Stochastic Gradient Descent

- This problem is solved by Stochastic Gradient Descent.
- In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration.
- The sample is randomly shuffled and selected for performing the iteration.

Normal Gradient Descent

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{m} (\hat{y}^i - y^i) x_j^i$$

Stochastic Gradient Descent

```
$for\ i\ in\ range\,(m):$
    $\theta_j = \theta_j - \alpha\,(\hat{y}^i - y^i)\,x_j^i$
```

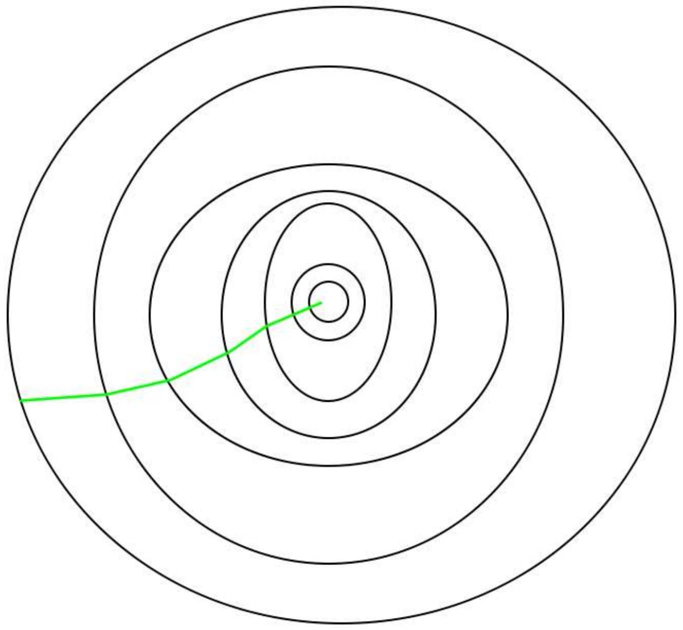
Stochastic Gradient Descent vs Typical Gradient Descent

- So, in SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples. Less expensive than normal GD.

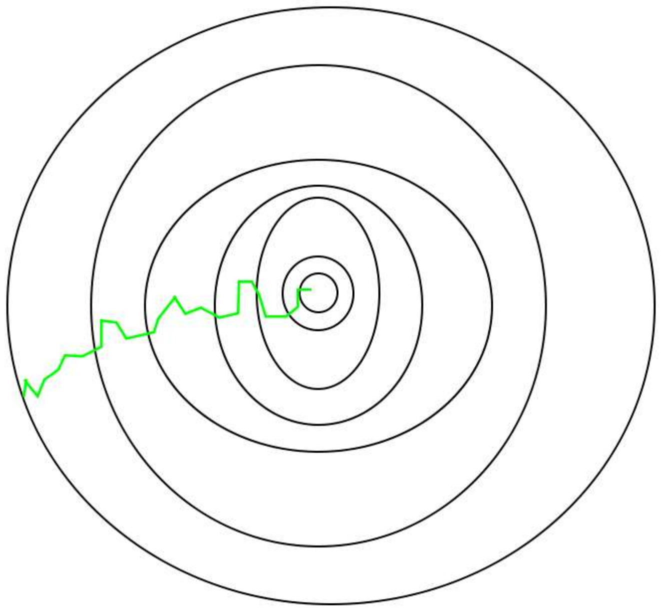
- In SGD, since only one sample from the dataset is chosen at random for each iteration, the path taken by the algorithm to reach the minima is usually noisier than your typical Gradient Descent algorithm.

- But that doesn’t matter all that much because the path taken by the algorithm does not matter, as long as we reach the minima and with significantly shorter training time.

Stochastic Gradient Descent vs Typical Gradient Descent

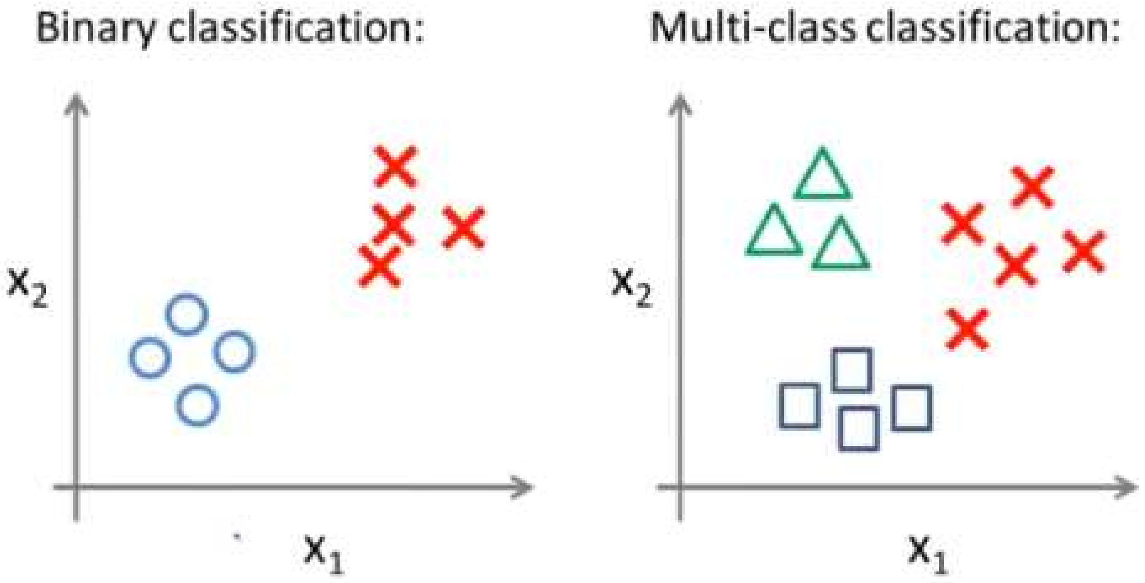


Path taken by Typical Gradient Descent

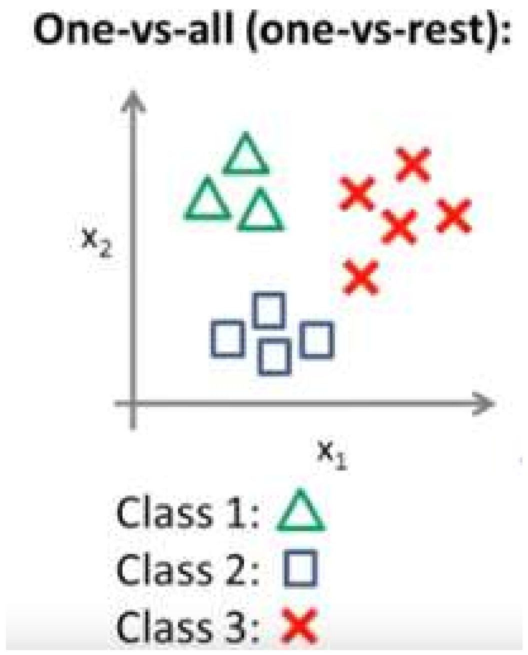


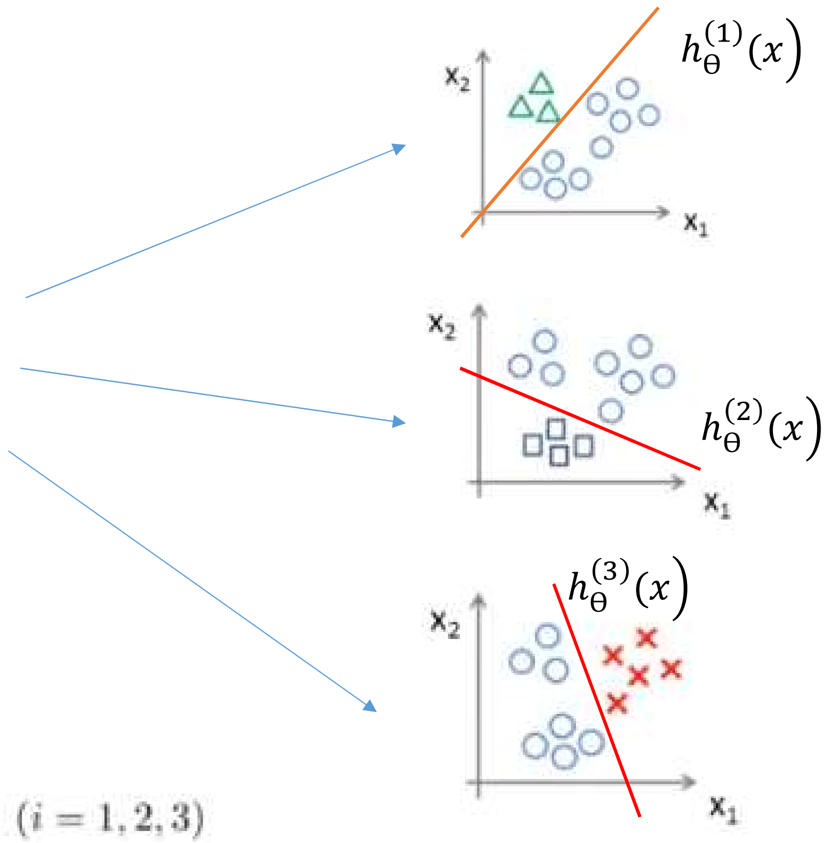
Path taken by Stochastic Gradient Descent

Multi Class Classification



One vs. All (One vs. Rest)





$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \qquad (i = 1, 2, 3)$$

One vs. All (One vs. Rest)

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes

$$\max_i h_\theta^{(i)}(x)$$

to continue...