COURSERA

MACHINE LEARNING

Andrew Ng, Stanford University

Course Materials: http://cs229.stanford.edu/materials.html

WEEK 1

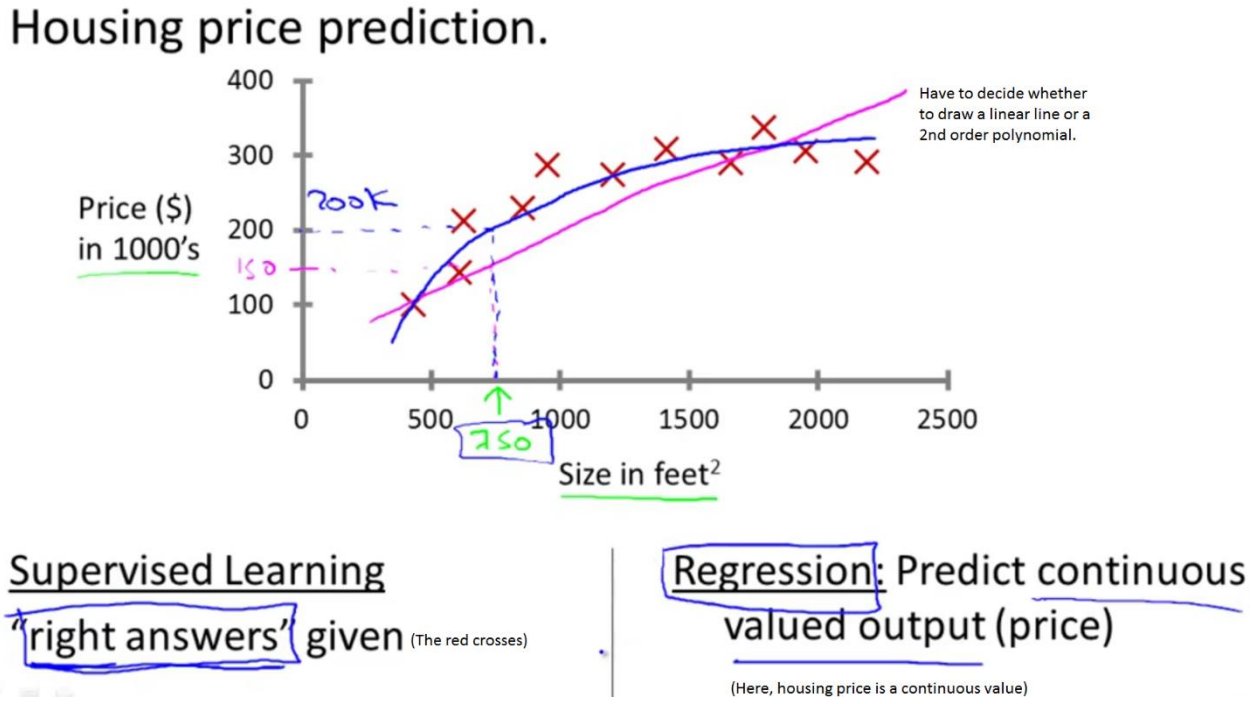
What is Machine Learning?

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Supervised Learning

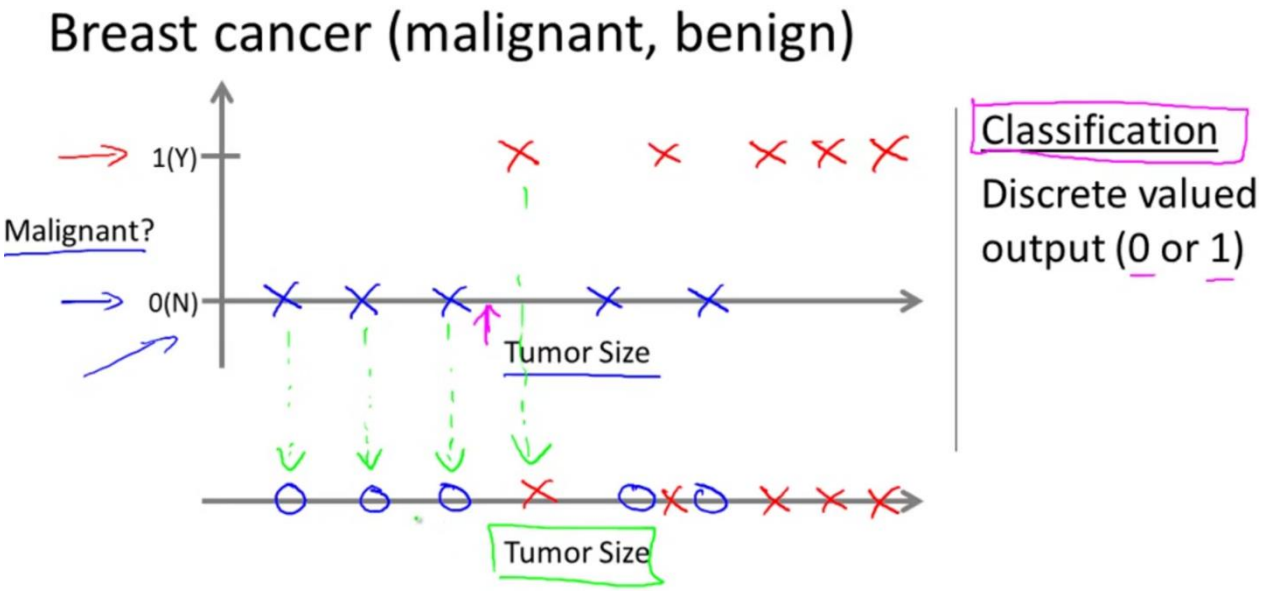
In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Example 1 - Regression

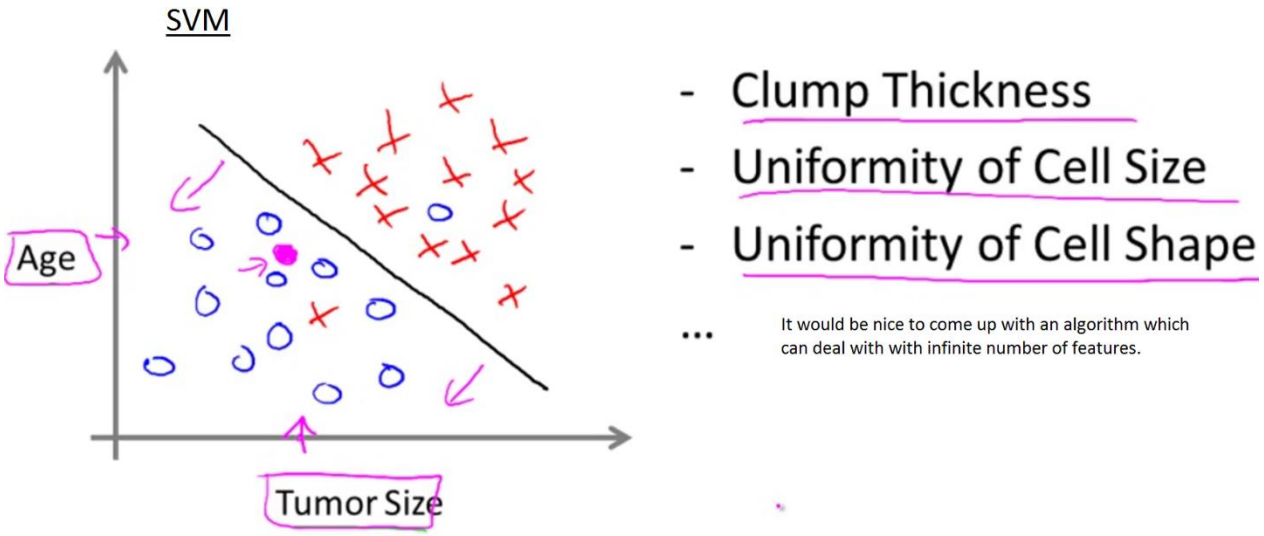


Example 2 - Classification

a) Single Feature:



b) Multiple Features:



Unsupervised Learning

Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables. We can derive this structure by clustering the data based on relationships among the variables in the data. With unsupervised learning there is no feedback based on the prediction results.

Linear Regression

We'll use $x^{(i)}$ to denote the "input" variables (features), and $y^{(i)}$ to denote the "output" or target variable that we are trying to predict. A pair $(x^{(i)}, y^{(i)})$ is called a training example, a list of m training examples $(x^{(i)}, y^{(i)})$ is called a training set.

Our goal is, given a training set, to learn a function h : X → Y so that h(x) is a "good" predictor for the corresponding value of y.

Training set of housing prices (Portland, OR)

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

m = 47

Notation:

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

(x, y) – one training example

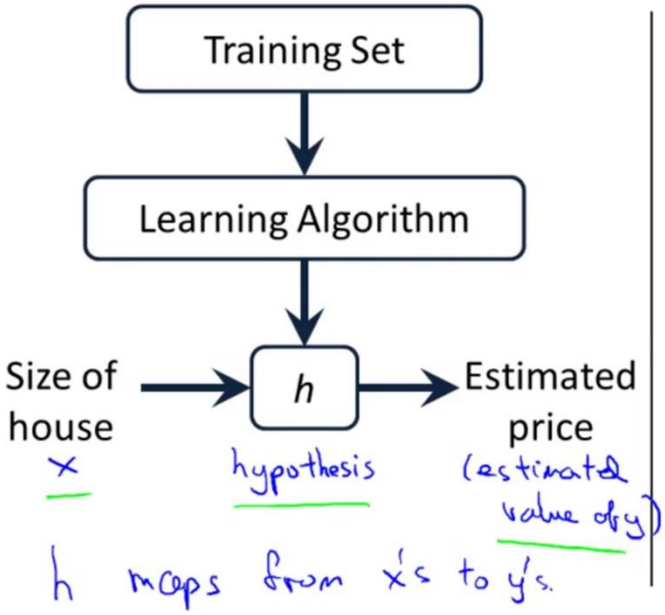
$(x^{(i)}, y^{(i)})$ – i^{th} training example

$x^{(1)} = 2104$

$x^{(2)} = 1416$

$y^{(1)} = 460$

Andrew

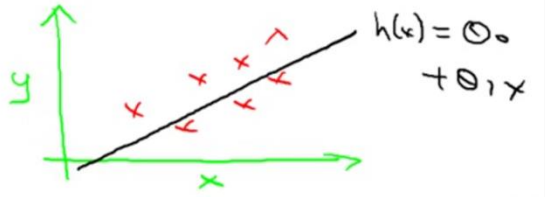


How do we represent *h* ?

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

θ_i's: Parameters

How to choose θ_i's ?

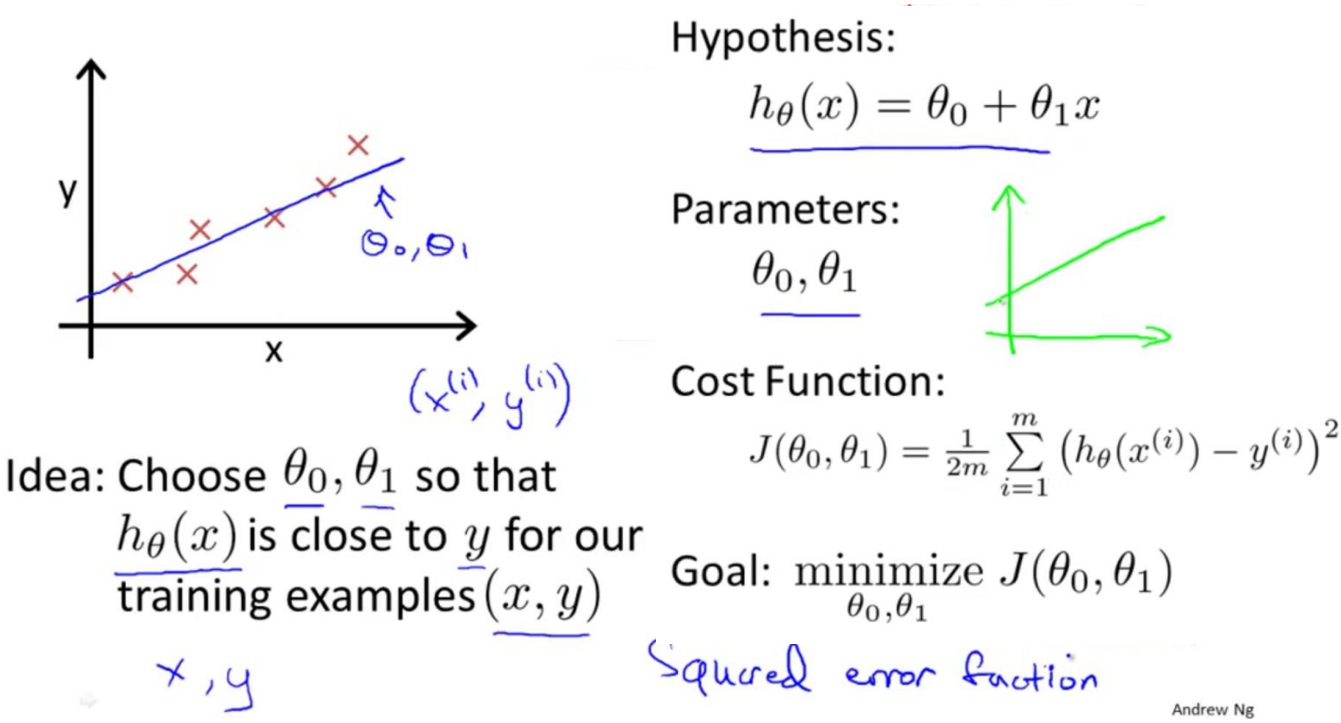


Linear regression with one variable. (x)

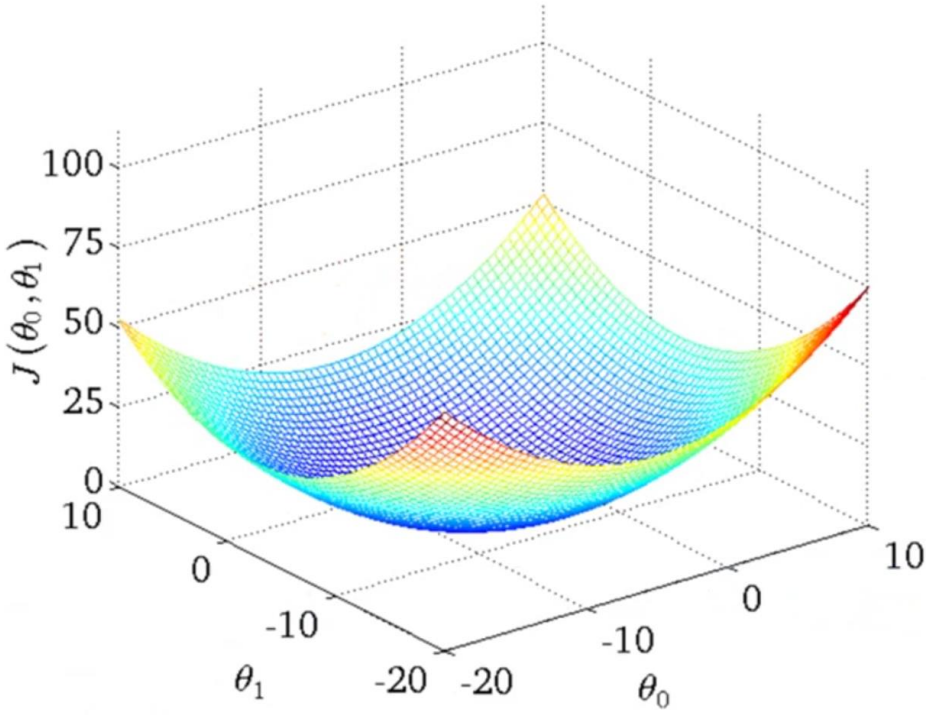
Univariate linear regression.

Cost Function

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average difference of all the results of the hypothesis with inputs from x's and the actual output y's.



The mean is halved as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the (½) term.



We need an algorithm to automatically find the set of parameters θ_0 and θ_1 that minimizes the cost function $J(\theta_0,\theta_1)$, which is always a **convex** function.

Gradient Descent

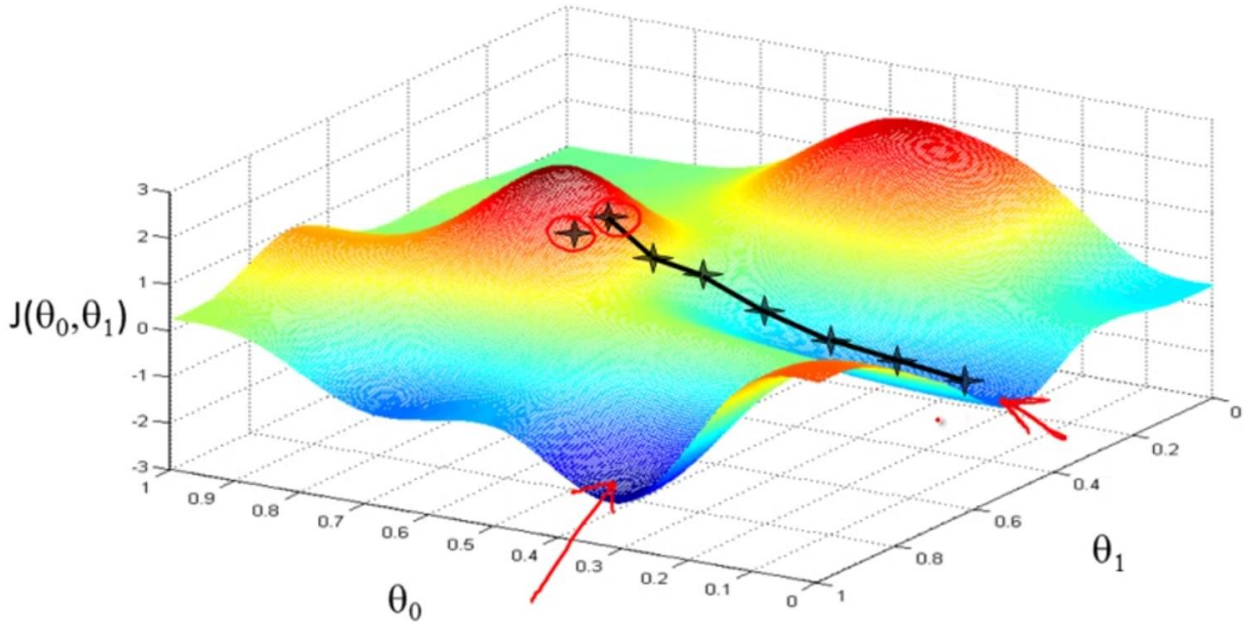
Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Starting point may lead to different local minimum values.

Outline:

- Start with some θ_0, θ_1 (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 algorithm

Assignment: $a := b$, $a := a+1$ | Truth assertion: $a = b$, $a = a+1$ ✗

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

learning rate

Simultaneously update θ_0 and θ_1

Correct: Simultaneous update

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

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

$\theta_0 := \text{temp0}$

$\theta_1 := \text{temp1}$

Incorrect:

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

$\theta_0 := \text{temp0}$

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

$\theta_1 := \text{temp1}$

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Linear Regression Model

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

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

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

Gradient descent algorithm

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

repeat until convergence {

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

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

}

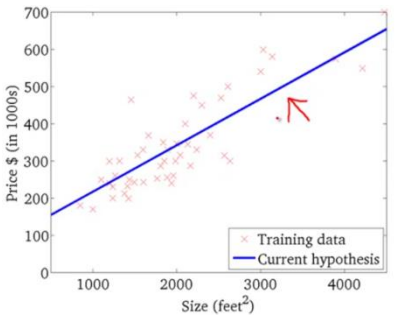
update θ_0 and θ_1 simultaneously

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

This is simply gradient descent on the original cost function J. This method looks at every example in the entire training set on every step, and is called **batch gradient descent**. Note that, while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here for linear regression has only one global and no other local optima (J is a convex quadratic function); thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum.

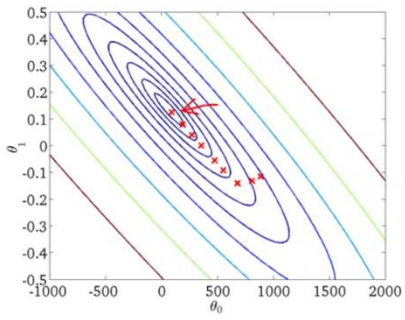
$h_\theta(x)$

(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)



WEEK 2

Multivariate Linear Regression

Multiple features (variables).

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$m = 47$

$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$

Notation:

- n = number of features $\quad n = 4$
- $x^{(i)}$ = input (features) of i^{th} training example.
- $x_j^{(i)}$ = value of feature j in i^{th} training example.

$x_3^{(2)} = 2$

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

For convenience of notation, define $x_0 = 1$. $(x_0^{(i)} = 1)$

$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$

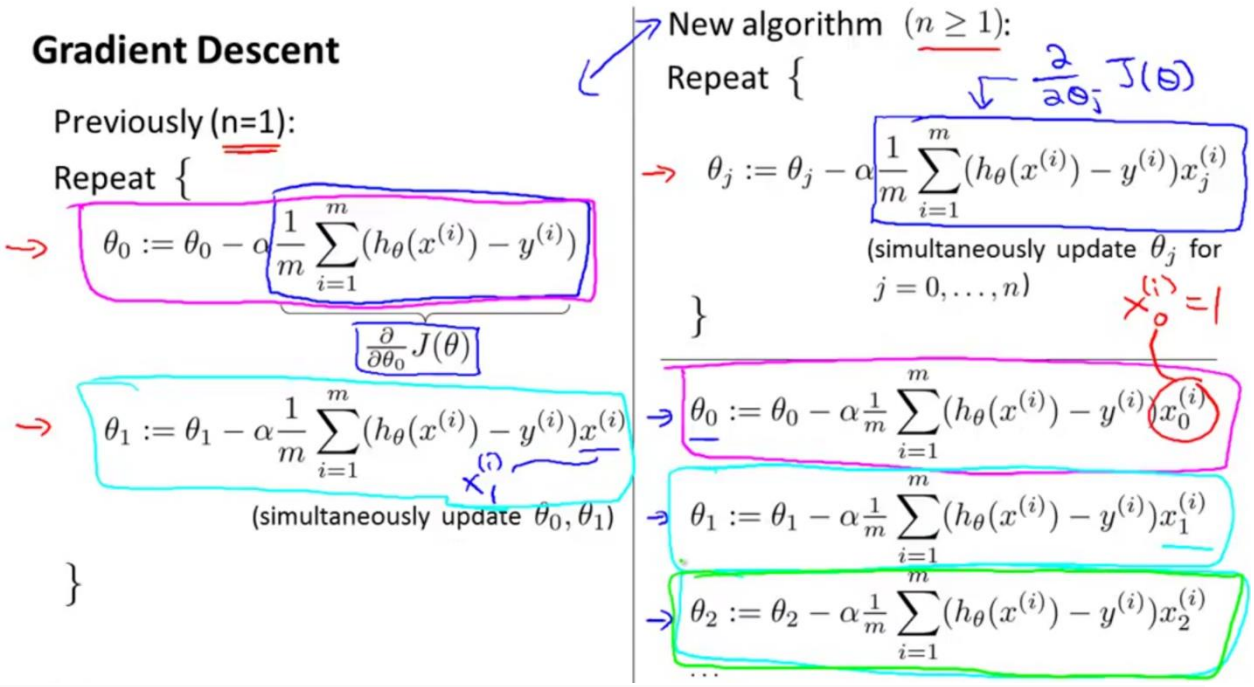
$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n \quad (x_0 = 1)$

$= \theta^T x.$

$[\theta_0 \; \theta_1 \cdots \theta_n] = \theta^T$, $(n+1) \times 1$ matrix

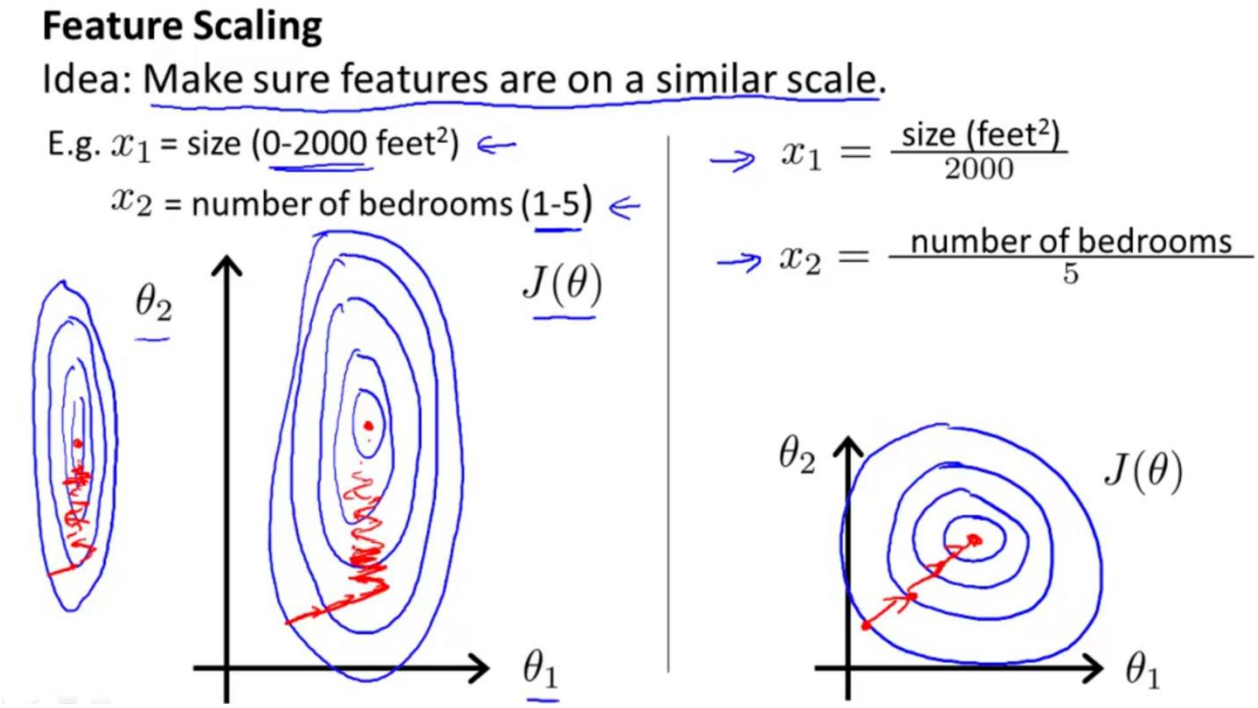
$\theta^T x$

x



We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

Two techniques to help with this are **feature scaling** and **mean normalization**. Feature scaling involves dividing the input values by the range of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero.



Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

E.g. $x_1 = \frac{size - 1000}{2000}$

Average size = 100

$x_2 = \frac{\#bedrooms - 2}{5}$

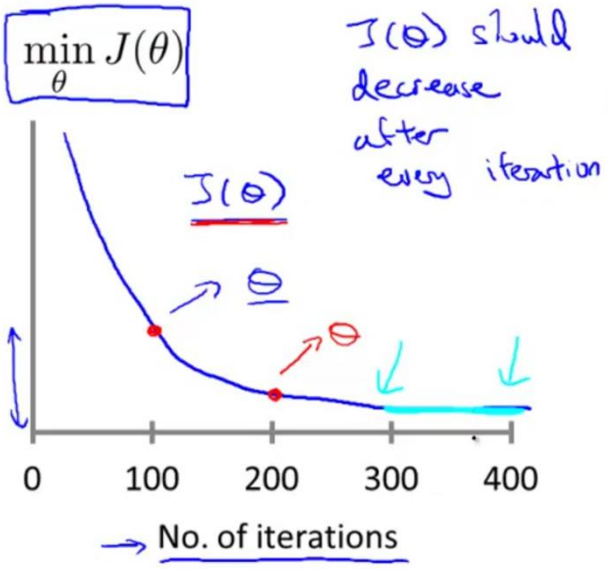
1-5 bedrooms

$-0.5 \leq x_1 \leq 0.5$, $-0.5 \leq x_2 \leq 0.5$

$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$ (μ_1: avg value of x_1 in training set; s_1: range (max - min))

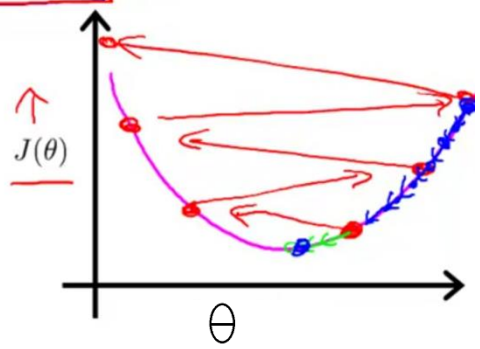
$x_2 \leftarrow \frac{x_2 - \mu_2}{s_2}$

Making sure gradient descent is working correctly.



Gradient descent not working.

Use smaller α.



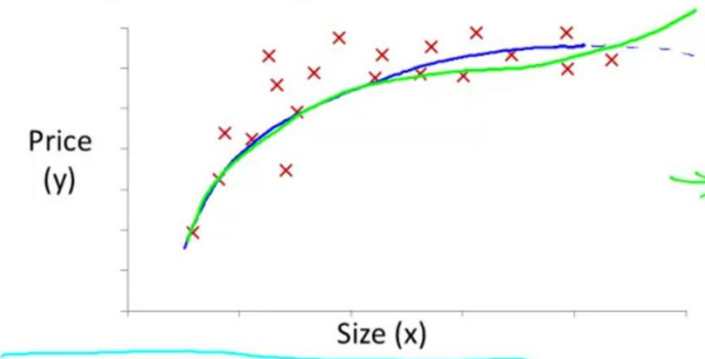
It has been proven that if α is sufficiently small, then J(θ) will decrease on every iteration.

To choose α, try

$\ldots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \ldots$

(≈3x, 3x, ≈3x)

Polynomial regression



$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

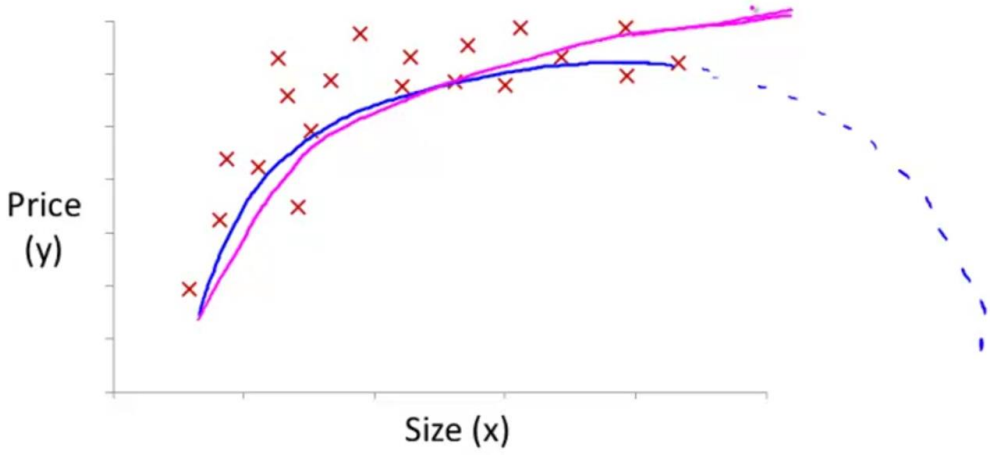
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1(size) + \theta_2(size)^2 + \theta_3(size)^3$$

$\rightarrow x_1 = (size)$
$\rightarrow x_2 = (size)^2$
$\rightarrow x_3 = (size)^3$

Size: 1 - 1000
$Size^2$: 1 - 1000,000
$Size^3$: $1 - 10^9$

scaling becomes very important

Choice of features



$\rightarrow h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$

$\rightarrow h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$

Normal Equation

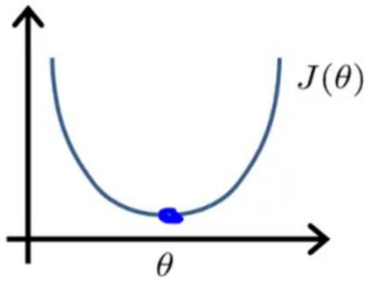
Gradient descent gives one way of minimizing J. Let's discuss a second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm. In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θj's, and setting them to zero. This allows us to find the optimum theta without iteration.

Intuition: If 1D $(\theta \in \mathbb{R})$

$J(\theta) = a\theta^2 + b\theta + c$

$\frac{d}{d\theta} J(\theta) = \cdots \overset{\text{set}}{=} 0$

Solve for θ



$\theta \in \mathbb{R}^{n+1}$ $\quad J(\theta_0, \theta_1, \ldots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots \overset{\text{set}}{=} 0$ (for every j)

Solve for $\theta_0, \theta_1, \ldots, \theta_n$

m **examples** $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$ **;** n **features**

$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$

X (design matrix) $= \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$ $m \times (n+1)$

E.g. If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$ $\quad X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix}$ $m \times 2$ $\quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$

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

Examples: $m = 4$.

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \quad m \times (n+1)$$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \quad m\text{-dimensional vector}$$

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

If $X^T X$ is **noninvertible,** the common causes might be having:

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization".

Solutions to the above problems include deleting a feature that is linearly dependent with another or deleting one or more features when there are too many features.

m **training examples,** n **features.**

Gradient Descent	Normal Equation
• Need to choose α.	• No need to choose α.
• Needs many iterations.	• Don't need to iterate.
• Works well even when n is large.	• Need to compute $(X^T X)^{-1}$ $n \times n$ $O(n^3)$
$n = 10^6$	• Slow if n is very large.
	$n = 100$
	$n = 1000$
	$n = 10000$

WEEK 3

Classification

Logistic Regression Model

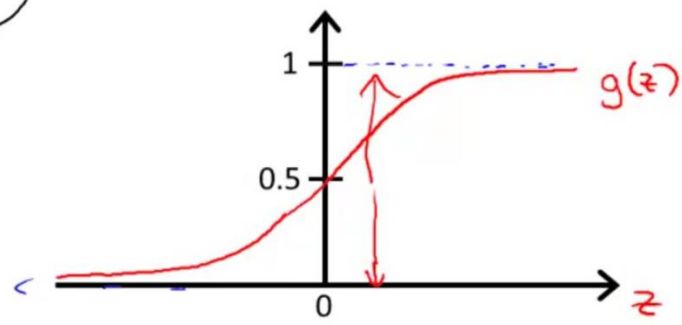
Want $0 \leq h_\theta(x) \leq 1$

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

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

$z = \theta^T x$

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



→ Sigmoid function

→ Logistic function

The function g(z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

Interpretation of Hypothesis Output $h_\theta(x)$

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

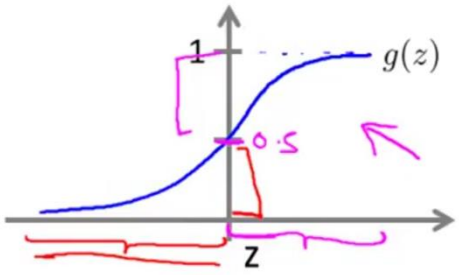
Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$h_\theta(x) = 0.7$ $\quad y = 1$

Tell patient that 70% chance of tumor being malignant

$h_\theta(x) = P(y=1 \mid x; \theta)$ "probability that y = 1, given x, parameterized by θ"

Logistic regression



$h_\theta(x) = g(\theta^T x) = P(y=1 \mid x; \theta)$

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

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

$\theta^T x \geq 0$

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

$h_\theta(x) = g(\theta^T x) \geq 0.5$ whenever $\theta^T x \geq 0$ (z)

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

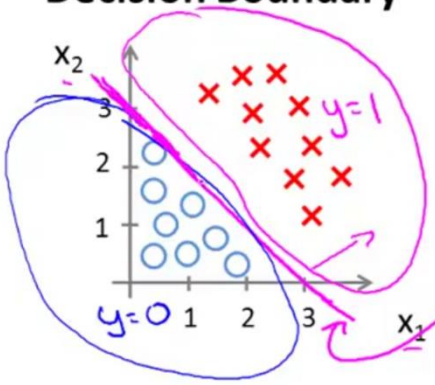
$h_\theta(x) = g(\theta^T x)$

$g(z) < 0.5$

$\theta^T x < 0$

Decision Boundary

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



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

$-3 \quad 1 \quad 1$

Decision boundary

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

$\theta^T x$

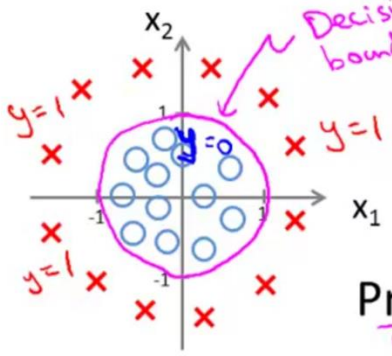
$x_1 + x_2 \geq 3$

x_1, x_2 → $h_\theta(x) = 0.5$

$x_1 + x_2 = 3$

$x_1 + x_2 < 3$ → $y = 0$

Non-linear decision boundaries



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

$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

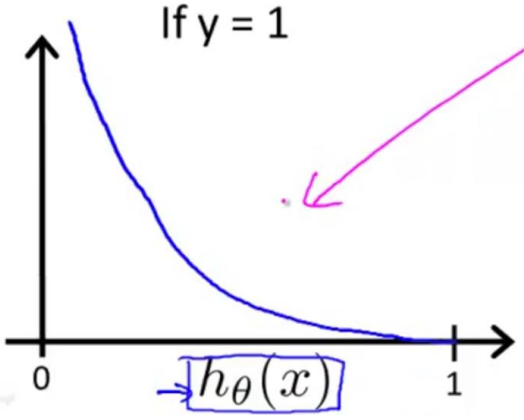
Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

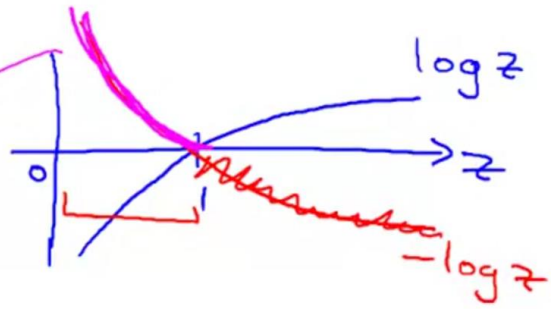
$x_1^2 + x_2^2 = 1$

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

Logistic regression cost function

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





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

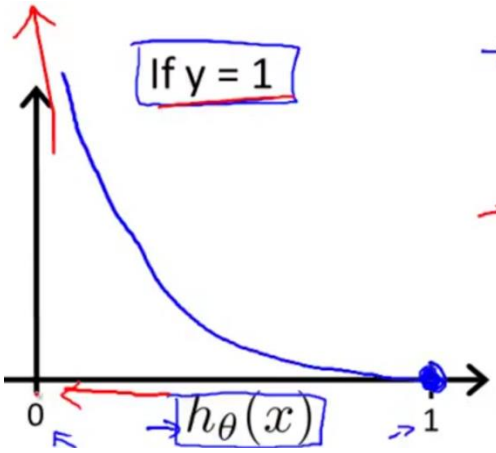
- **Note that writing the cost function in this way guarantees that J(θ) is convex for logistic regression.**

Simplified form:

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

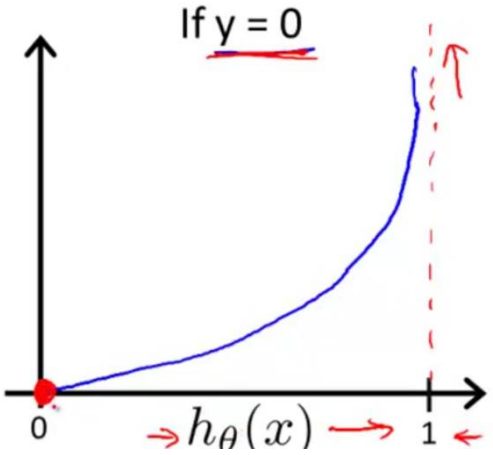
If $y = 1$: $\text{Cost}(h_\theta(x), y) = -\log h_\theta(x)$

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



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

$\rightarrow$ 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.



If our correct answer 'y' is 0, then the cost function will be 0 if our hypothesis function also outputs 0. If our hypothesis approaches 1, then the cost function will approach infinity.

Gradient Descent

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

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

Repeat {

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

}

Algorithm looks identical to linear regression!

Multiclass Classification

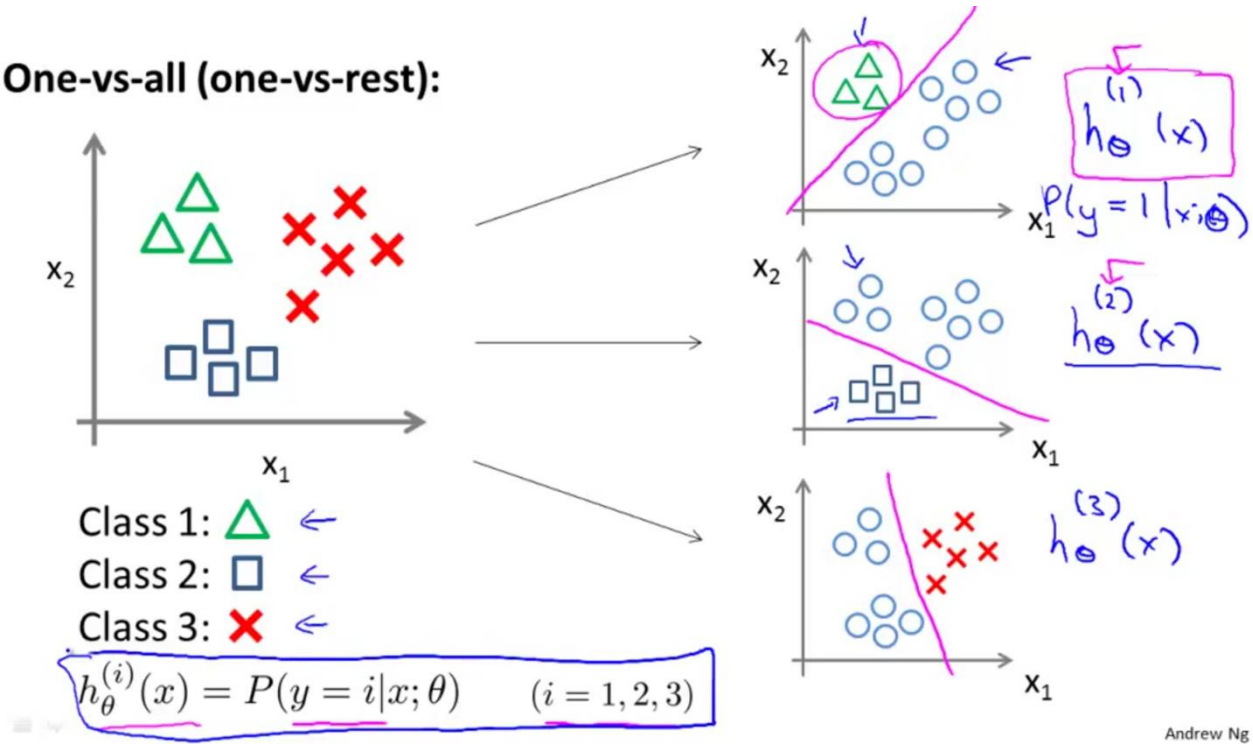
One-vs-all

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

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

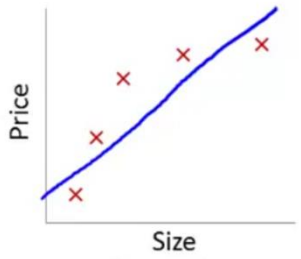
One-vs-all (one-vs-rest):



Andrew Ng

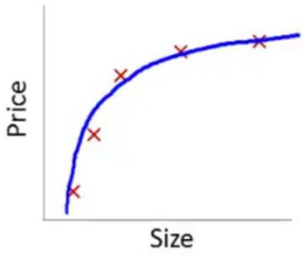
The Problem of Overfitting

Example: Linear regression (housing prices)



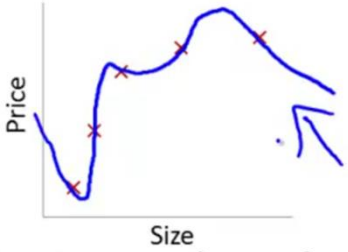
$\rightarrow \theta_0 + \theta_1 x$

"Underfit" "High bias"



$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"

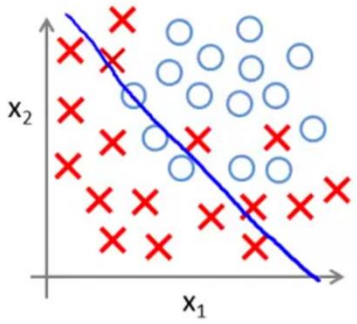


$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

"Overfit" "High variance"

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

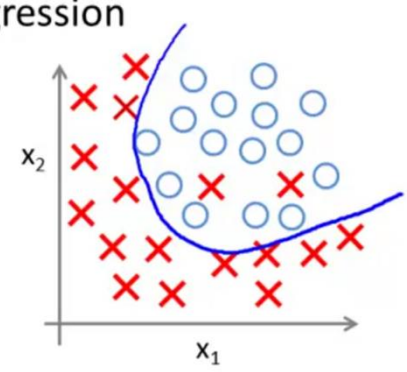
Example: Logistic regression



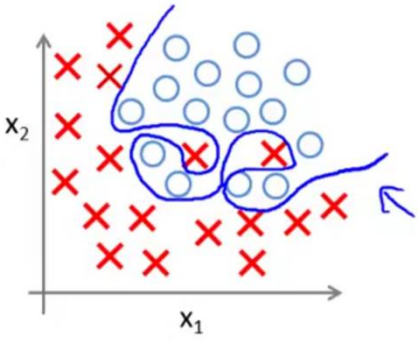
$\rightarrow h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

"Underfit"



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

"Overfit"

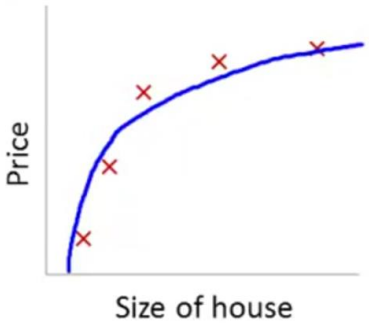
How to Address Overfitting

Options:

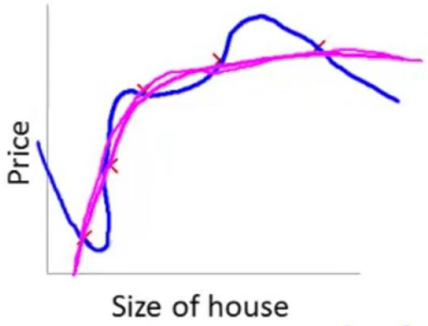
1. Reduce number of features.
   - — Manually select which features to keep.
   - — Model selection algorithm (later in course).
2. Regularization.
   - — Keep all the features, but reduce magnitude/values of parameters θ_j.
   - — Works well when we have a lot of features, each of which contributes a bit to predicting y.

Regularization

Intuition



$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Suppose we penalize and make θ_3, θ_4 really small.

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

$\theta_3 \approx 0 \qquad \theta_4 \approx 0$

Regularization.

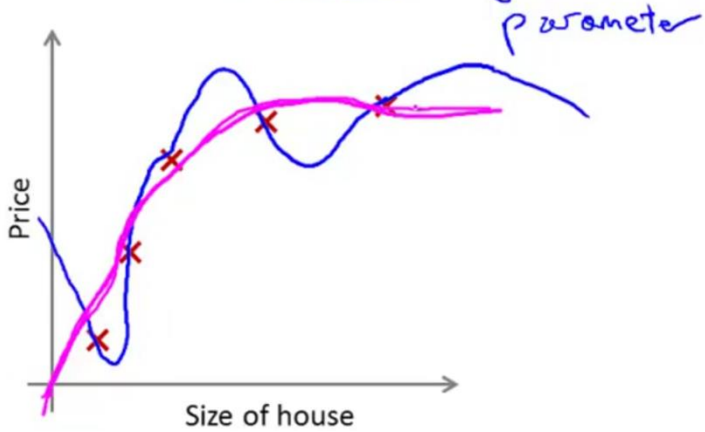
Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$
- "Simpler" hypothesis
- Less prone to overfitting

Housing:
- Features: $x_1, x_2, \ldots, x_{100}$
- Parameters: $\theta_0, \theta_1, \theta_2, \ldots, \theta_{100}$

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

regularization parameter

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



What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?

$h_\theta(x)$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$\theta_1, \theta_2, \theta_3, \theta_4$

$\theta_1 \approx 0, \theta_2 \approx 0$

$\theta_3 \approx 0, \theta_4 \approx 0$

$h_\theta(x) = \theta_0$

Regularized Linear Regression

Gradient descent

Repeat $\{$

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

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \quad (j = 1, 2, 3, \ldots, n)$$

$\}$

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

$1 - \alpha \frac{\lambda}{m} < 1$

Normal equation

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

invertible.

Regularized Logistic Regression

Cost Function

Recall that our cost function for logistic regression was:

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

We can regularize this equation by adding a term to the end:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1-y^{(i)})\ \log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Gradient descent

Repeat $\{$

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

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad (j = 1, 2, 3, \ldots, n)$$

$\theta_1 \ldots \theta_n$

$\}$

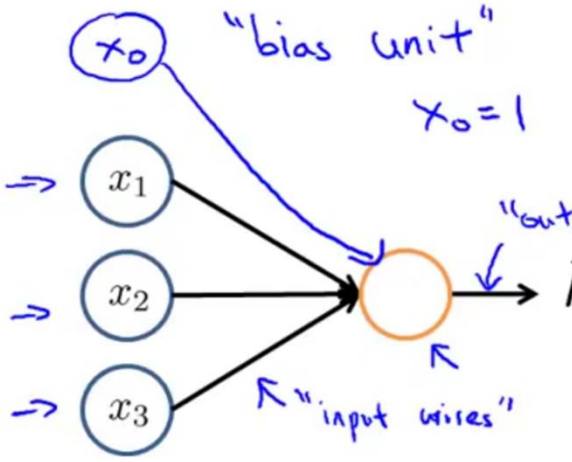
$\frac{\partial}{\partial\theta_j}J(\theta)$

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

WEEK 4

Neural Networks

Neuron model: Logistic unit



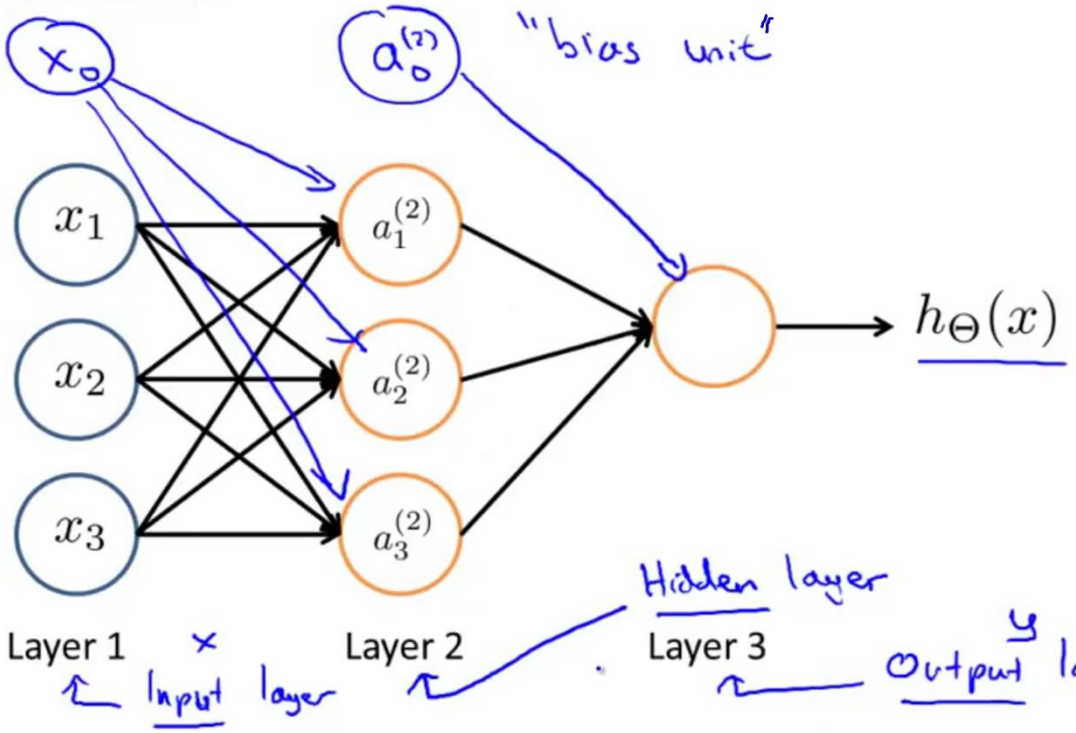
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

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

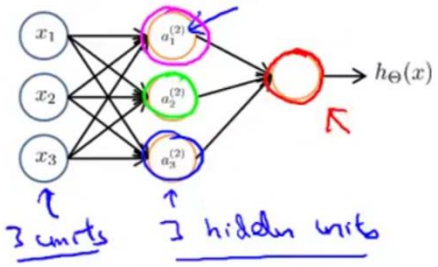
Sigmoid (logistic) activation function.

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

Neural Network



Neural Network



3 units 3 hidden units $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

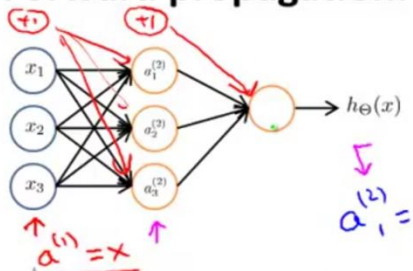
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

Forward propagation: Vectorized implementation



$a^{(1)} = x$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

$a_1^{(2)} = g(z_1^{(2)})$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)}a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

Add $a_0^{(2)} = 1$. → $a^{(2)} \in \mathbb{R}^4$

$$z^{(3)} = \Theta^{(2)}a^{(2)}$$

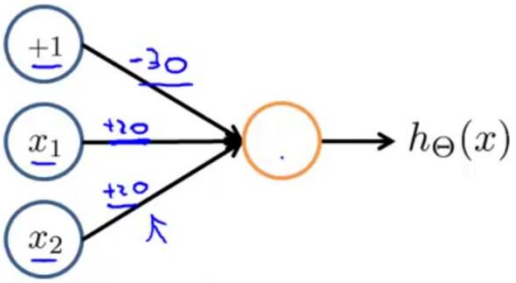
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

$a_3^{(2)} = g(z_3^{(2)})$ $z^{(3)}$

Simple example: AND

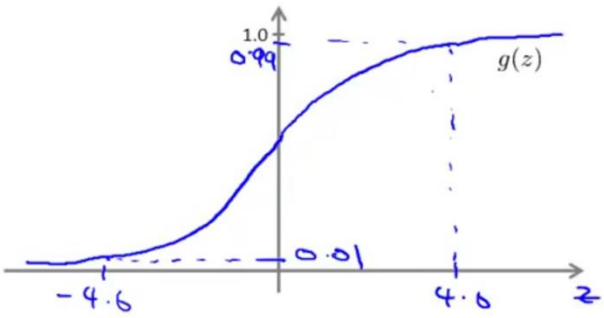
$x_1, x_2 \in \{0, 1\}$

$y = x_1$ AND x_2



$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

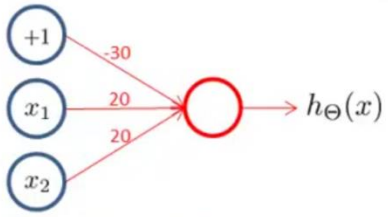
$\Theta^{(1)}_{10}$ $\Theta^{(1)}_{11}$ $\Theta^{(1)}_{12}$



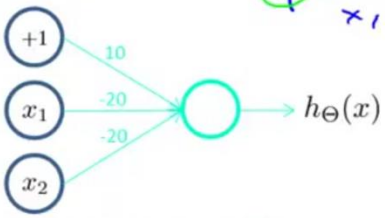
x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

$h_\Theta(x) \approx x_1$ AND x_2

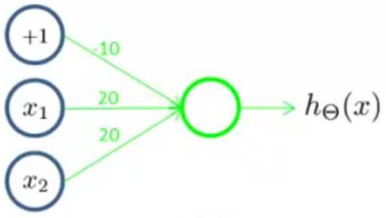
Putting it together: x_1 XNOR x_2



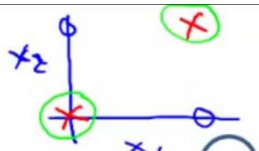
→ x_1 AND x_2

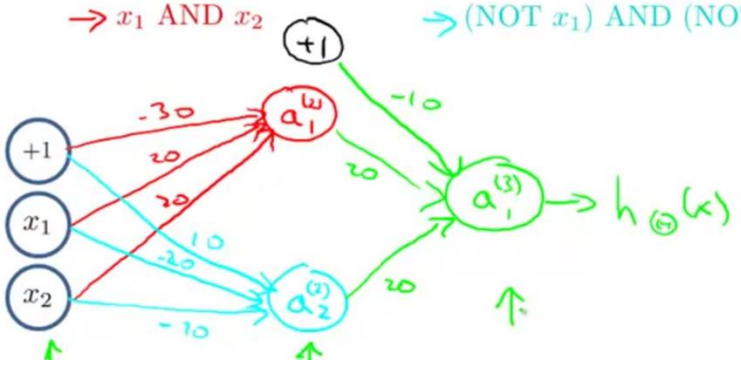


→ (NOT x_1) AND (NOT x_2)



→ x_1 OR x_2

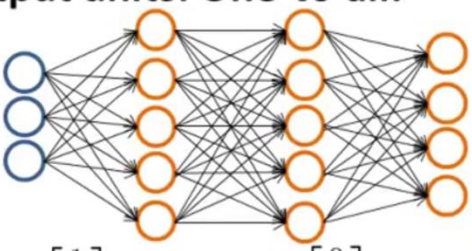




x_1	x_2	$a^{(2)}_1$	$a^{(2)}_2$	$h_\Theta(x)$
→ 0	0	0	1	1 ←
0	1	0	0	0
1	0	0	0	0
→ 1	1	1	0	1 ←

Multiclass Classification

Multiple output units: One-vs-all.



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

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

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

Previously $y \in \{1, 2, 3, 4\}$

$(x^{(i)}, y^{(i)})$

$h_\Theta(x^{(i)}) \approx y^{(i)}$

$\in \mathbb{R}^4$

Cost function

$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th}$ output

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

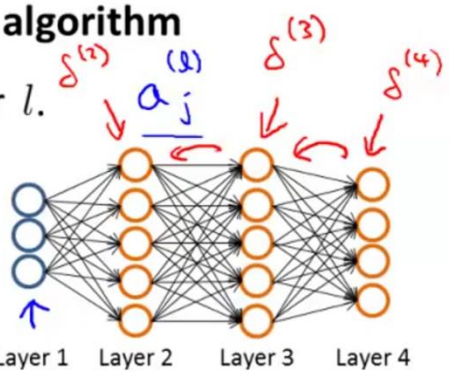
- L = total number of layers in the network
- s_l = number of units (not counting bias unit) in layer l
- K = number of output units/classes

In the regularization part, after the square brackets, we must account for multiple theta matrices. The number of columns in our current theta matrix is equal to the number of nodes in our current layer (including the bias unit). The number of rows in our current theta matrix is equal to the number of nodes in the next layer (excluding the bias unit). As before with logistic regression, we square every term.

WEEK 5

Gradient computation: Backpropagation algorithm

Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l.



For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

(No $\delta^{(1)}$)

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\lambda = 0)$$

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$$

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j). (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to m ← $(x^{(i)}, y^{(i)})$

- Set $a^{(1)} = x^{(i)}$
- Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$
- Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$
- $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

MATLAB Implementation Notes

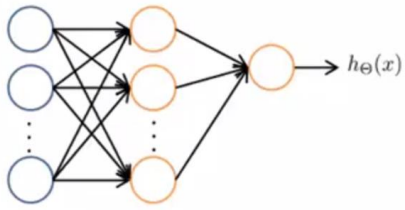
In order to use optimizing functions such as `fminunc()`, we will want to "unroll" all the elements and put them into one long vector:

Example

$s_1 = 10, s_2 = 10, s_3 = 1$

$\Theta^{(1)} \in \mathbb{R}^{10\times 11}, \Theta^{(2)} \in \mathbb{R}^{10\times 11}, \Theta^{(3)} \in \mathbb{R}^{1\times 11}$

$D^{(1)} \in \mathbb{R}^{10\times 11}, D^{(2)} \in \mathbb{R}^{10\times 11}, D^{(3)} \in \mathbb{R}^{1\times 11}$




```
thetaVec = [ Theta1(:); Theta2(:); Theta3(:)];
DVec = [D1(:); D2(:); D3(:)];

Theta1 = reshape(thetaVec(1:110),10,11);
Theta2 = reshape(thetaVec(111:220),10,11);
Theta3 = reshape(thetaVec(221:231),1,11);
```

Learning Algorithm

Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.

Unroll to get `initialTheta` to pass to

```
fminunc(@costFunction, initialTheta, options)
```

```
function [jval, gradientVec] = costFunction(thetaVec)
```

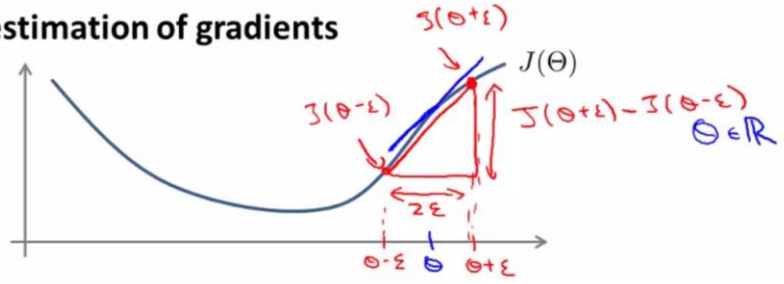
From `thetaVec`, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$. reshape

Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$.

Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get `gradientVec`.

Gradient Checking

Numerical estimation of gradients



$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$

$\varepsilon = 10^{-4}$

Implement:
```
gradApprox = (J(theta + EPSILON) - J(theta - EPSILON))
             /(2*EPSILON)
```

Parameter vector θ

$\theta \in \mathbb{R}^n$ (E.g. θ is "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)

$\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon, \theta_2, \theta_3, \ldots, \theta_n) - J(\theta_1-\epsilon, \theta_2, \theta_3, \ldots, \theta_n)}{2\epsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\epsilon, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2-\epsilon, \theta_3, \ldots, \theta_n)}{2\epsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n+\epsilon) - J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n-\epsilon)}{2\epsilon}$$

```
for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) = thetaPlus(i) + EPSILON;
  thetaMinus = theta;
  thetaMinus(i) = thetaMinus(i) - EPSILON;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))
                  /(2*EPSILON);
end;
```

$\frac{\partial}{\partial \theta_i} J(\theta)$

Check that `gradApprox` $\approx$ `DVec` ← From backprop

Random Initialization

Initializing all theta weights to zero does not work with neural networks. When we backpropagate, all nodes will update to the same value repeatedly. Instead we can randomly initialize our weights for our Θ matrices using the following method:

Random initialization: Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$
(i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

E.g.

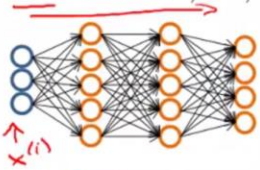
```
Theta1 =  rand(10,11)*(2*INIT_EPSILON)
          - INIT_EPSILON;

Theta2 =  rand(1,11)*(2*INIT_EPSILON)
          - INIT_EPSILON;
```

Training a neural network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

```
for i = 1:m {
```

→ Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).



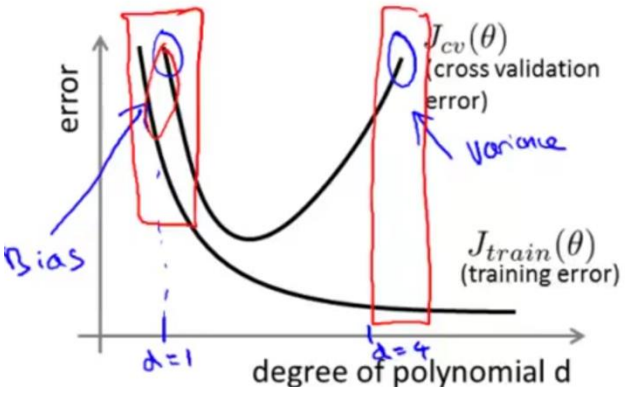
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.

→ Then disable gradient checking code.

6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

Diagnosing bias vs. variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Bias (underfit):

$J_{train}(\theta)$ will be high

$J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):

$J_{train}(\theta)$ will be low

$J_{cv}(\theta) \gg J_{train}(\theta)$

Choosing the regularization parameter λ

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

1. Try $\lambda = 0$ → $\min_\theta J(\theta)$ → $\theta^{(1)}$ → $J_{cv}(\theta^{(1)})$
2. Try $\lambda = 0.01$ → $\min_\theta J(\theta)$ → $\theta^{(2)}$ → $J_{cv}(\theta^{(2)})$
3. Try $\lambda = 0.02$ → $\theta^{(3)}$ → $J_{cv}(\theta^{(3)})$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$ → $\theta^{(5)}$ $J_{cv}(\theta^{(5)})$

⋮

12. Try $\lambda = 10$ → $\theta^{(12)}$ → $J_{cv}(\theta^{(12)})$

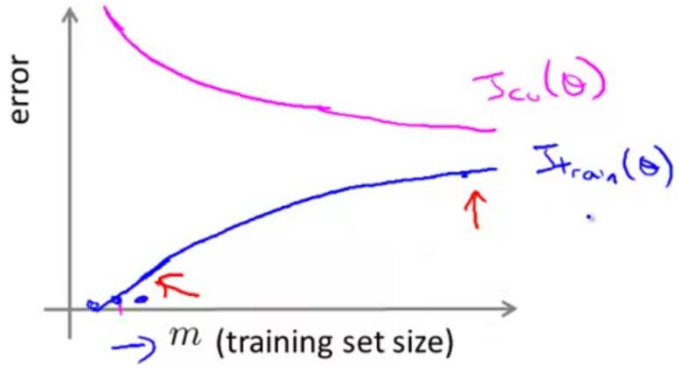
10.24

Pick (say) $\theta^{(5)}$. Test error: $J_{test}(\theta^{(5)})$

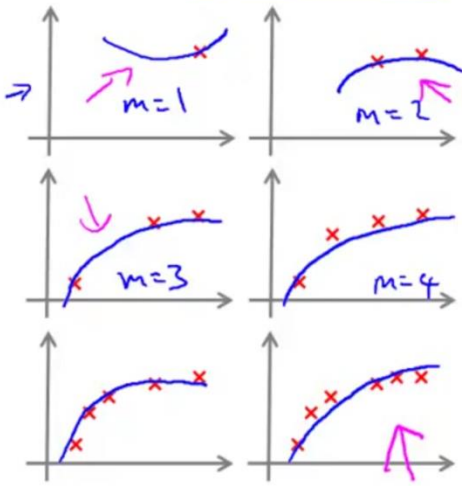
Learning curves

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

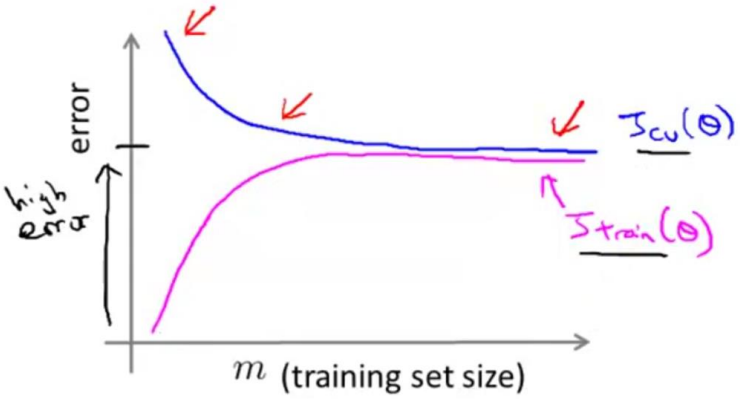
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

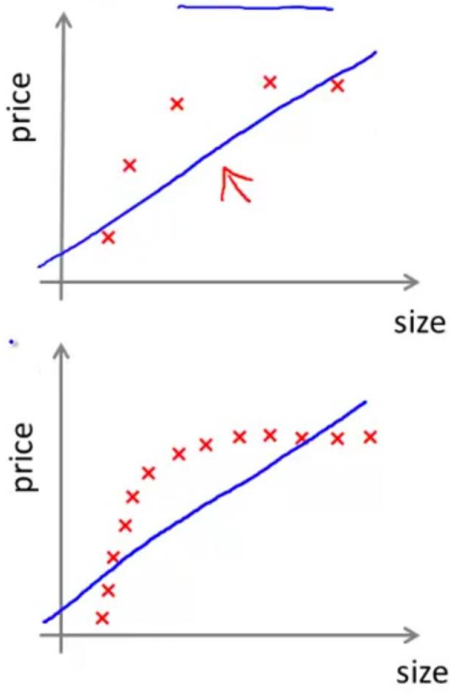


High bias



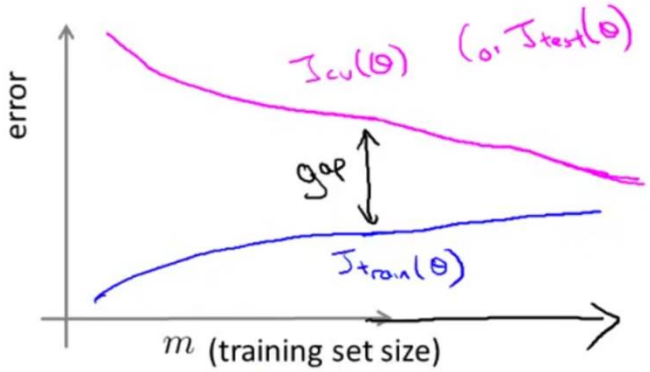
If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

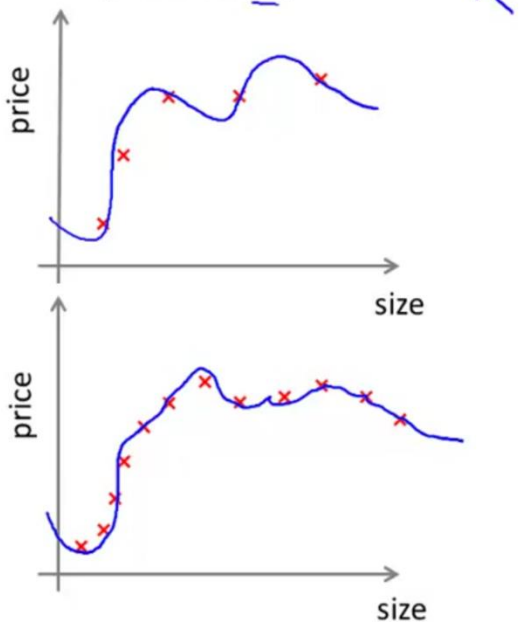
$$h_\theta(x) = \theta_0 + \theta_1 x$$



High variance

$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{100} x^{100}$
(and small λ)





If a learning algorithm is suffering from high variance, getting more training data is likely to help. ←

Prioritizing What to Work On

Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features $(x_1^2, x_2^2, x_1 x_2, \text{etc})$ → fixes high bias
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

Recommended approach

- Start with a simple algorithm that you can implement quickly. Implement it and test it on your cross-validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Error analysis: Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

How to compare precision/recall numbers?

	Precision(P)	Recall (R)	Average	F_1 Score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.51	0.0392

Predict y=1 all the time

Average: $\frac{P+R}{2}$

F_1 Score: $2\frac{PR}{P+R}$

Large data rationale

Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units). low bias algorithms. ←

→ $J_{train}(\theta)$ will be small.

Use a very large training set (unlikely to overfit) low variance ←

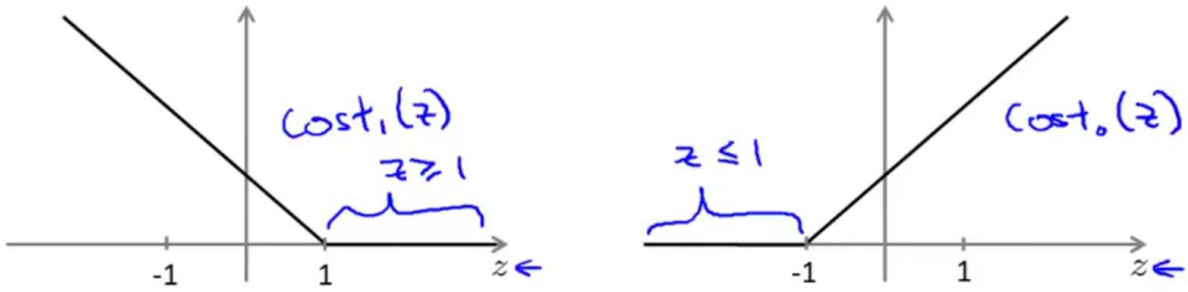
→ $J_{train}(\theta) \approx J_{test}(\theta)$

$J_{test}(\theta)$ will be small

WEEK 7

Support Vector Machine

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$



If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

If x if far from $l^{(1)}$:

$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$$

$l^{(1)} \rightarrow f_1$

$l^{(2)} \rightarrow f_2$

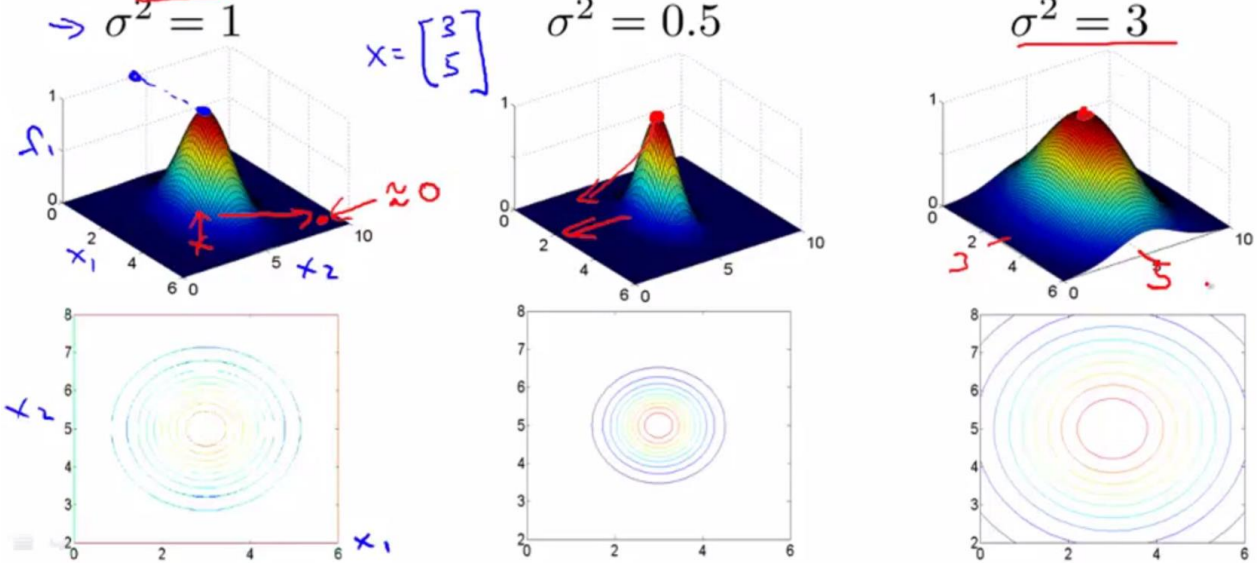
$l^{(3)} \rightarrow f_3$.

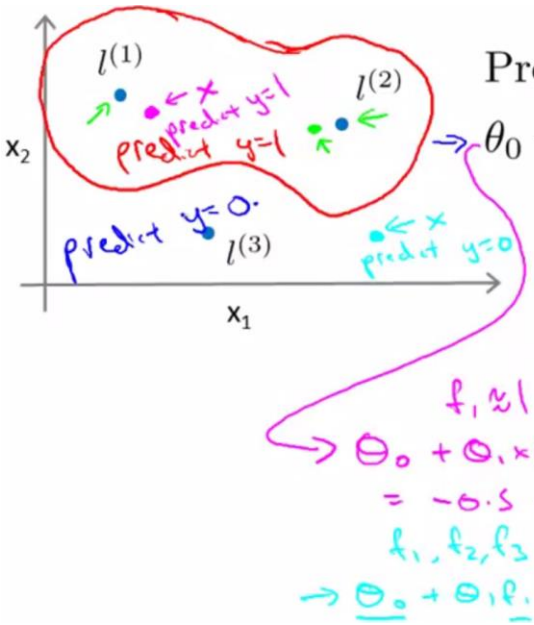
Gaussian Kernel

Example:

$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$, $f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

$\Rightarrow \sigma^2 = 1$ $\quad \sigma^2 = 0.5$ $\quad \sigma^2 = 3$





Predict "1" when

$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

$\theta_0 = -0.5, \quad \theta_1 = 1, \quad \theta_2 = 1, \quad \theta_3 = 0$

$f_1 \approx 1, \quad f_2 \approx 0, \quad f_3 \approx 0.$

$\theta_0 + \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 \times 0$

$= -0.5 + 1 = 0.5 \geq 0$

$f_1, f_2, f_3 \approx 0$

$\rightarrow \theta_0 + \theta_1 f_1 + \ldots \approx -0.5 < 0$

SVM with Kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$

→ Predict "y=1" if $\theta^T f \geq 0$

$\theta \in \mathbb{R}^{m+1}$

$\theta_0 f_0 + \theta_1 f_1 + \cdots + \theta_m f_m$

Training:

$n = m$

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{m=m} \theta_j^2$$

$\theta^T x^{(i)}$ $\theta^T f^{(i)}$

Need to specify:

→ Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel ("linear kernel")

Predict "y = 1" if $\theta^T x \geq 0$

$\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$

→ n large, m small

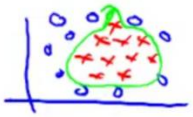
$x \in \mathbb{R}^{n+1}$

Gaussian kernel:

$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$.

Need to choose σ^2.

$x \in \mathbb{R}^n$, n small and/or m large



Kernel (similarity) functions:

$x^{(i)}$ $l^{(j)} = x^{(j)}$

f_i

```
function f = kernel(x1,x2)
    f = exp( -|| x1 - x2 ||^2 / (2σ^2) )
return
```

Note: Do perform feature scaling before using the Gaussian kernel.

$x \in \mathbb{R}^n$

$||x - l||^2$

$v = x - l$

$||v||^2 = v_1^2 + v_2^2 + \cdots + v_n^2$

$= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$

1000 feet^2 1-5 bedrooms