# Week 1

## Supervised Learning: (Given "right answer" for each example in the data)

Mainly 2 types of problems:

(1) Classification: discrete-valued output

(2) Regression: real-valued output (continuous)

## Unsupervised Learning:

(1) clustering

(2) Non-clustering

---

## Linear Regression with 1 variable

### (1) The Model

training set: examples

housing data:

| size in feet$^2$ (x) | price ($) in 1000's (y) |
|---|---|
| 2104 | 460 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |
| ⋮ | ⋮ |

} m

Notation:

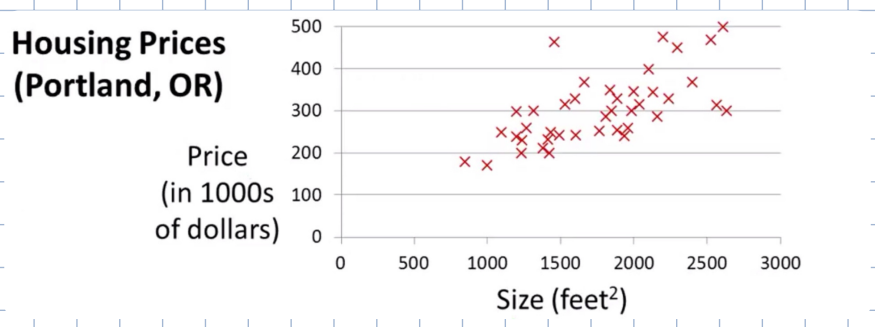
m = no. of training examples

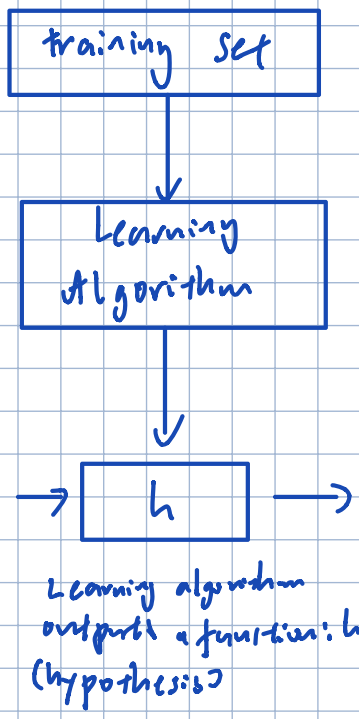
x's = "input" variable / features

y's = "output" variable / "target" variable

(x, y) = a single training example

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

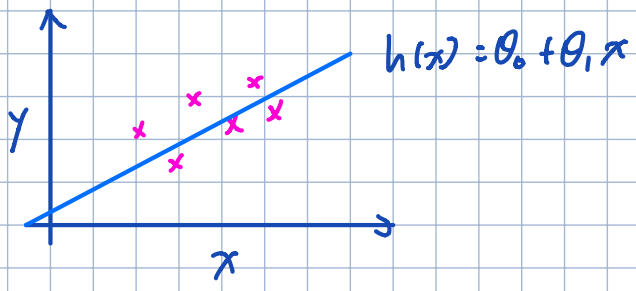





How do we represent $h$?

We choose:

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




This is linear regression with 1 variable

OR Univariate linear regression.

So $h$ is a function maps from $x \to y$

→ Supervised Learning Goal:

given a training set, to learn a function $h: X \to Y$ so that $h(x)$ is a "good" predictor

$X$: space of input value

$Y$: space of output value

## ② Cost Function

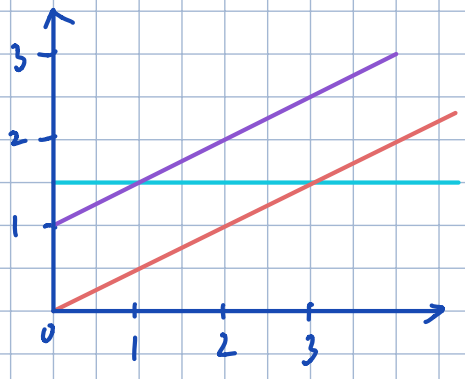
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

$\theta_i$'s: Parameters of the model

How to choose $\theta_i$'s?

Idea:

Choose $\theta_0, \theta_1$ so that $h_\theta(x)$ is close to $y$ for our training examples $(x, y)$




① $\theta_0 = 1.5, \theta_1 = 0$

$h(x) = 1.5 + 0 \cdot x$

② $\theta_0 = 0, \theta_1 = 0.5$

$h(x) = 0 + 0.5x$

③ $\theta_0 = 1, \theta_1 = 0.5$

$h(x) = 1 + 0.5x$

We want to solve a minimisation problem:

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

$\theta_0, \theta_1$: find value of $\theta_0, \theta_1$ to minimise

$\frac{1}{2m}$: maths easier for easier minimisation

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

By convention, we define a <u>cost function</u>:

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

squared error function or mean squared error

$y^{(i)}$: $\hat{y}_i$

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

i.e. it is $\frac{1}{2}\bar{x}$, where $\bar{x}$: mean of the square of $h_\theta(x^{(i)}) - y^{(i)}$

* mean is halved as a convenience for the computation of the gradient descent.

We can measure the accuracy of hypothesis function by using a cost function.

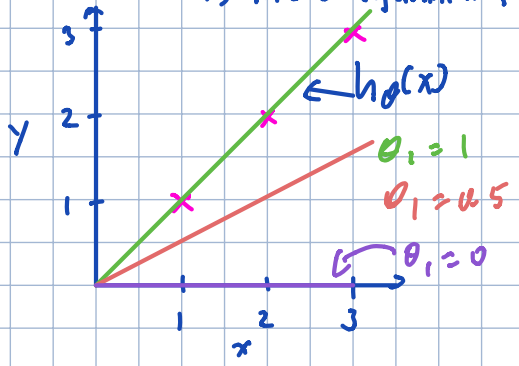
## A Simplified Model

$h_\theta(x) = \theta_1 x$ i.e. $\theta_0 = 0$

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

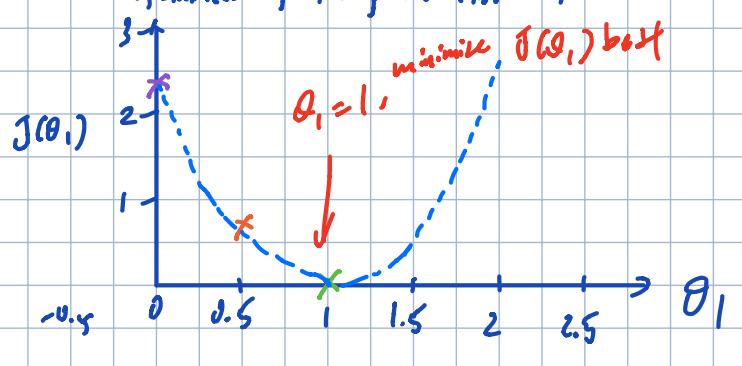
hypothesis function $h_\theta(x)$

for fixed $\theta_1$, this is a function of $x$




cost function $J(\theta_1)$

function of the parameter $\theta_1$



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

$\theta_1 = 1$

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

$$= \frac{1}{2m}(0^2 + 0^2 + 0^2)$$

$$= 0^2$$

i.e. $J(1) = 0$

$$J(\theta_1 = 0.5) = \frac{1}{2m}[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2]$$

$$= \frac{1}{2\times 3}[3.5]$$

$$\approx 0.58$$

i.e. $J(0.5) = 0.58$

$$J(0) = \frac{1}{2\times 3}[1^2 + 2^2 + 3^2]$$

$$= \frac{1}{6}[14]$$

$$= 7/3$$

* Each value of $\theta_1$ corresponds to a different hypothesis function

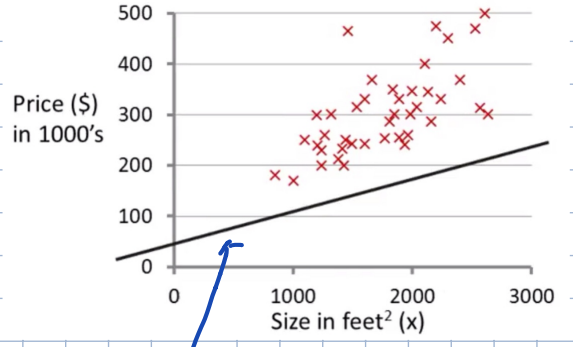
**Now with $\theta_0$ and $\theta_1$**

$h_\theta(x)$

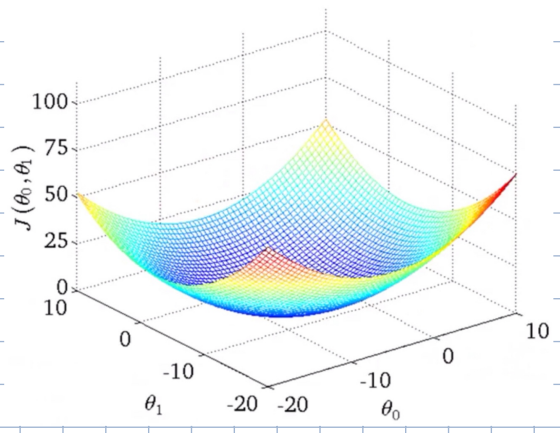
for fixed $\theta_0, \theta_1$, this is a function of $x$

$J(\theta_0, \theta_1)$

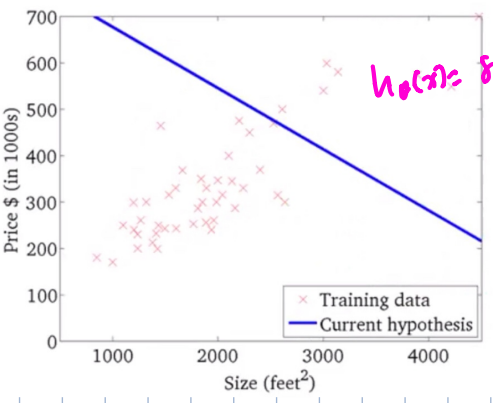
function of parameters $\theta_0, \theta_1$



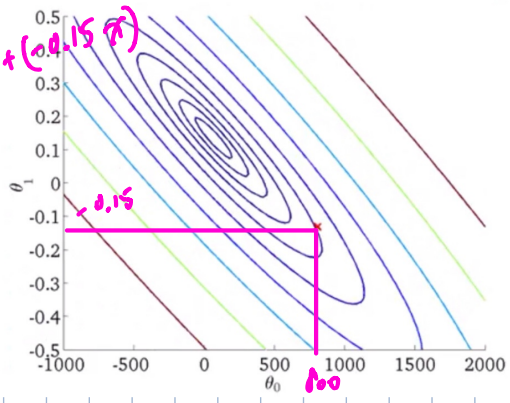

$h_\theta(x) = 50 + 0.06x$



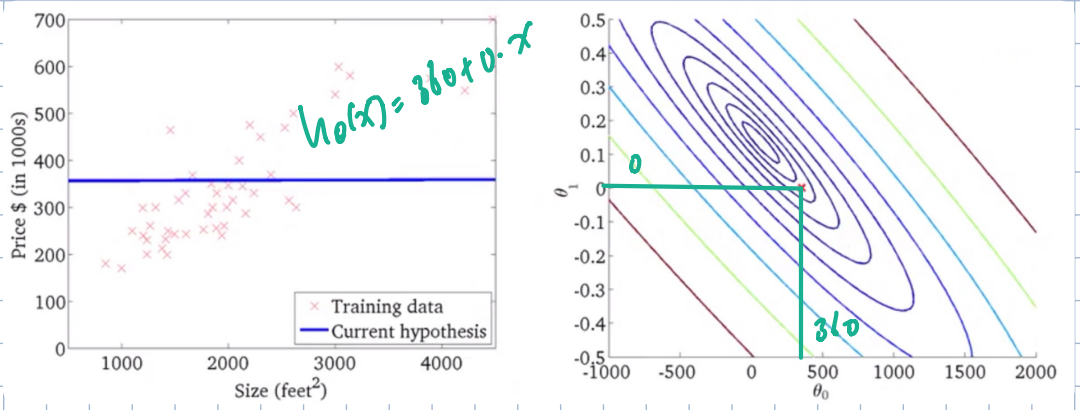

e.g.



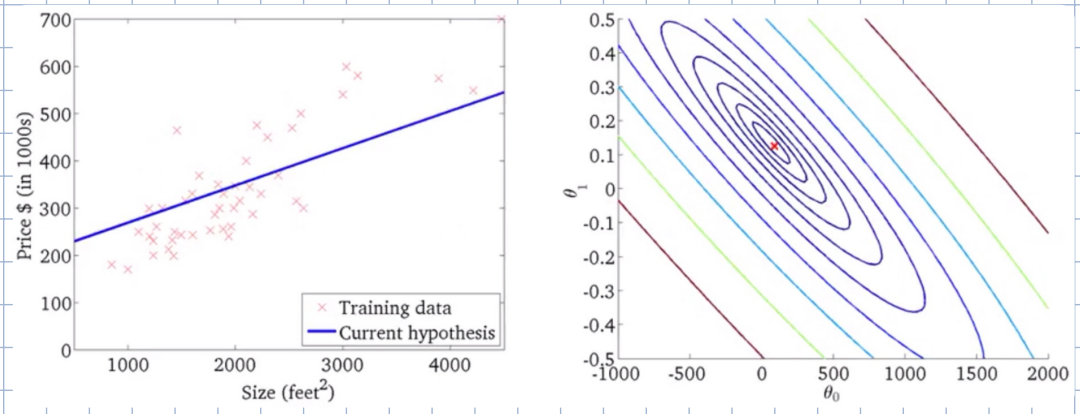

$h_\theta(x) = 800 + (-0.15x)$



e.g.




e.g.




we want an algorithm that finds $\theta_0, \theta_1$ that minimise $J$.

## ③ Gradient Descent

It is a general technique used in all ML, can be used to minimise cost function $J$ for linear regression.

more generally: $J(\theta_0, \theta_1, \theta_2, \dots \theta_n)$

$\min_{\theta_0 \dots \theta_n} J(\theta_0 \dots \theta_n)$

**Problem set up:**

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

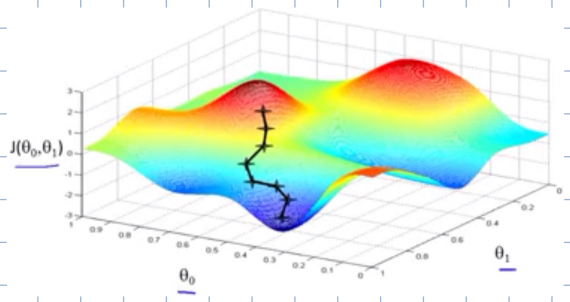
Want to come up with an algorithm for $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

**Out line:**

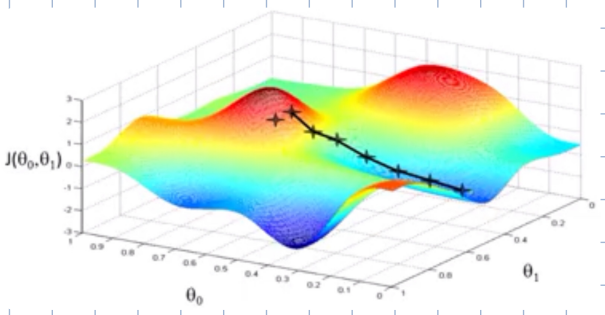
1. Start with some $\theta_0, \theta_1$ (say, initialise to $\theta_0 = \theta_1 = 0$)
2. Keep changing $\theta_0, \theta_1$ to reduce $J(\theta_0, \theta_1)$, until end up a minimum

Graphically:

initial point 1




initial point 2




taking derivatives to cost function (tangent)
→ slope of the tangent is the derivative at that point → gives us a direction

## Gradient Descent Algorithm

repeat until convergence {

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

}

assignment operator

learning rate

* Simultaneously update $\theta_0$ and $\theta_1$
i.e. compute the new $\theta_0$ and $\theta_1$ and update them together

Correct: Simultaneous update

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

Incorrect

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

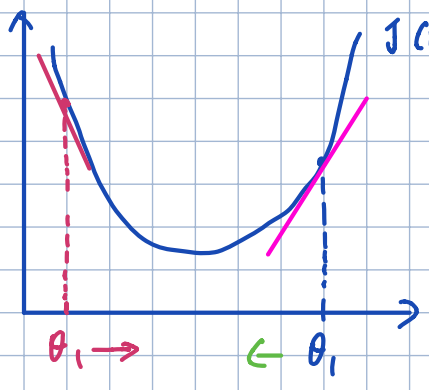
e.g. $\theta_0 = 1$, $\theta_1 = 2$, $\theta_j := \theta_j + \sqrt{\theta_0 \theta_1}$

$$\theta_0 := \theta_0 + \sqrt{\theta_0 \theta_1} = 1 + \sqrt{2}$$
$$\theta_1 := \theta_1 + \sqrt{\theta_0 \theta_1} = 2 + \sqrt{2}$$

## Simplified example

1 parameter

$$\min_{\theta_1} J(\theta_1) \qquad \theta_1 \in \mathbb{R}$$




$J(\theta_1)$ $(\theta_1 \in \mathbb{R})$ only 1 variable → total derivative

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

$\theta_1 := \theta_1 - \alpha \cdot$ (positive number)

$\theta_1 := \theta_1 - \alpha \cdot$ (negative number/slope)

* if $\alpha$ is too small, gradient descent can be slow
  If $\alpha$ is too large, gradient descent can overshoot the minimum
  ↳ might fail to converge, even diverge.

Gradient descent can converge to a local minimum, even when the learning rate $\alpha$ fixed.

$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$ ∵ $\frac{d}{d\theta_1} J(\theta_1)$ decrease when approaching min.

## Use gradient descent back to linear regression problem

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 $j=1$)

}

Linear Regression model

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

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

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

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

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

for $j=0$ (i.e. $\theta_0$): $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$

for $j=1$ (i.e. $\theta_1$): $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

repeat until convergence {

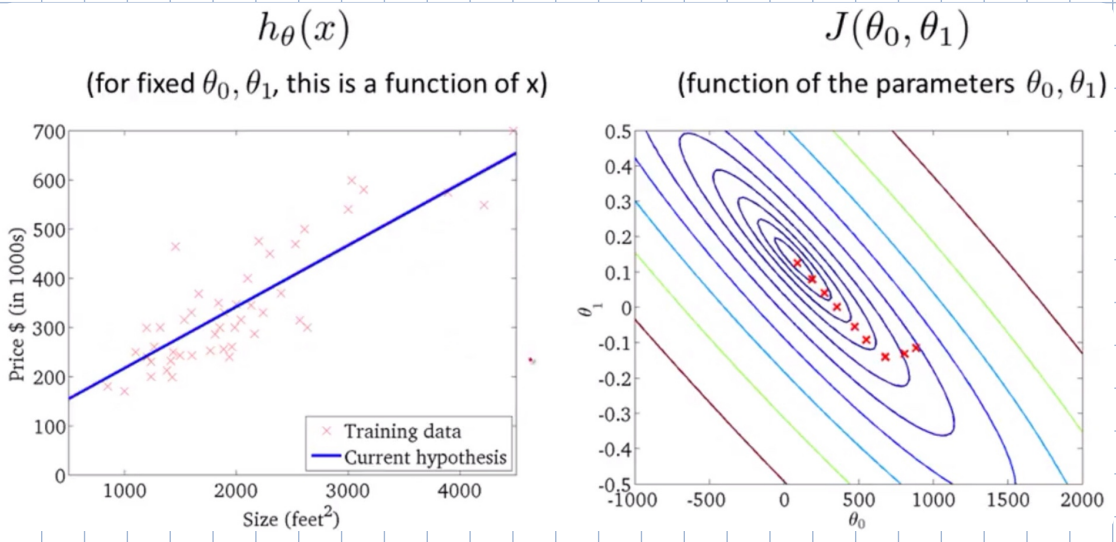
$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

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

$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$ $\quad \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

}

* Cost function for linear Regression is always a bowl-shaped function
  - ↳ convex function
    - ↳ no local optima, only global optima.
      - ↳ Gradient descent always converge to global optima




* "Batch" Gradient Descent

  Batch: each step of gradient descent uses all the training examples.

  (i.e. $\sum_{i=1}^{m}$ )

There exist a method to numerically solving the min. of J w/o iteration.

- ↳ normal equations method
  - ↳ but gradient descent scales better with large data sets.

# Linear Algebra Review

## Matrix

Rectangular array of numbers

$$\begin{bmatrix} 1902 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

$4 \times 2$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$2 \times 3$

or $\mathbb{R}^{2\times 3}$

Dimension of matrix: no. of rows x no. of columns

## matrix elements

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

$A_{ij}$ = "$i,j$" entry in the $i^{th}$ row, $j^{th}$ column

e.g. $A_{11} = 1402$

## Vector

special case of matrix: $n \times 1$ matrix

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

e.g. $n = 4 \Rightarrow 4 \times 1$

also 4-dimensional vector or $\mathbb{R}^4$

$y_i = i^{th}$ element

e.g. $y_1 = 460$

### 1-indexed vs 0-indexed

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

## Matrix Addition

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0.5 \\ 4 & 10 \\ 3 & 2 \end{bmatrix}$$

$3 \times 2 \qquad 3 \times 2 \qquad 3 \times 2$

## Scaler Multiplication

$$3 \times \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 6 & 15 \\ 9 & 3 \end{bmatrix}$$

## Matrix - Vector Multiplication

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

$3 \times 2 \qquad 2 \times 1 \qquad 3 \times 1$

## Matrix - Matrix Multiplication

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}$$

$2 \times 3 \qquad 3 \times 2 \qquad 2 \times 2$

## Matrix Inverse

If $A$ is square matrix, and if it has an inverse:

$$A A^{-1} = A^{-1} A = I$$

e.g. $\begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix} \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix} = I$

## Matrix Transpose

$A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}$ $\quad A^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$

Let $A$ be an $m \times n$ matrix, and let $B = A^T$

then $B$ is an $n \times m$ matrix, and

$B_{ij} = A_{ji}$

---

# Week 2

## Multivariate Linear Regression

| 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 |
| ... | ... | ... | ... | ... |

} $m$

$n$ = number of features

$x^{(i)}$ = input (features) of $i^{th}$ training example

$x_j^{(i)}$ = value of feature $j$ in $i^{th}$ training example

e.g. $x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix} \in \mathbb{R}^4$

$x_3^{(2)} = 2$

## Hypothesis

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

In general:

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

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

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

now:

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

## Gradient Descent for multiple variables

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

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

$\theta$ is $n+1$ dimensional vector

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

↳ $J(\theta)$

Gradient descent:

Repeat {

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

($J(\theta)$)

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

}

Repeat {

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

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

}

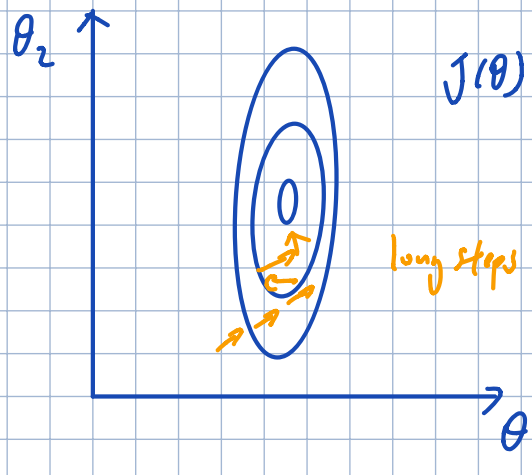
# Gradient Descent in practice – Feature Scaling

Idea: make sure features are on a similar scale

It allows gradient descent to converge more quickly

e.g. $x_1$ = size (0 – 2000 feet$^2$)

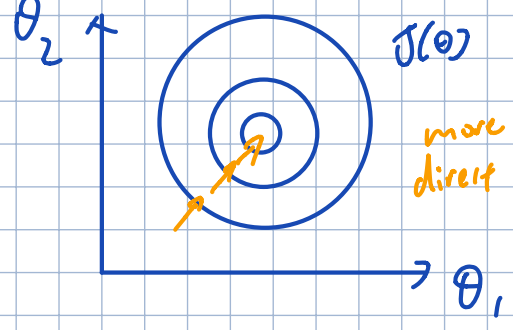
$x_2$ = no. of bedroom (1–5)




very tall, skinny, skewed contour of $J(\theta)$

$\because \theta_2$ can take on large value

If $x_1 = \dfrac{\text{size (feet}^2)}{2000}$, $x_2 = \dfrac{\text{no. of bedroom}}{5}$




$0 \le x_1 \le 1$ $\quad 0 \le x_2 \le 1$

## Feature Scaling

Get every feature to approximately $-1 \le x_i \le 1$ range

## Mean normalisation

using $x_i - \mu_i$ to make features have approximately zero mean

(Do NOT apply to $x_0 = 1$)

e.g. $x_1 = \dfrac{\text{size} - 1000}{2000}$, $x_2 = \dfrac{\text{no. of bedrooms} - 2}{5}$

$-0.5 \le x_1 \le 0.5$, $-0.5 \le x_2 \le 0.5$

* $x_1 \leftarrow \dfrac{x_1 - \mu_1}{s_1}$ ← $\mu_1$: avg value of $x_1$ in training set.

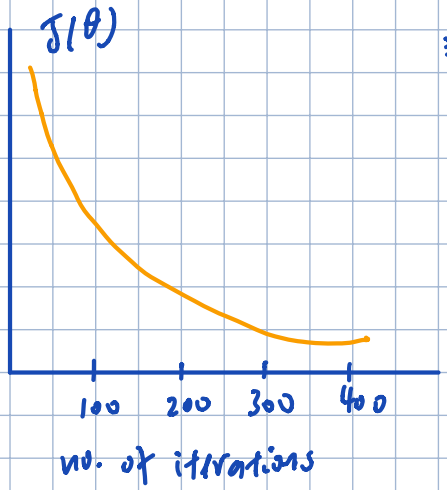
$s_1$ ← range or std deviation (max − min)

$\vdots$

$x_n \leftarrow \dfrac{x_n - \mu_n}{s_n}$ (for 1 feature)

## Gradient Descent in practice — Learning Rate

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



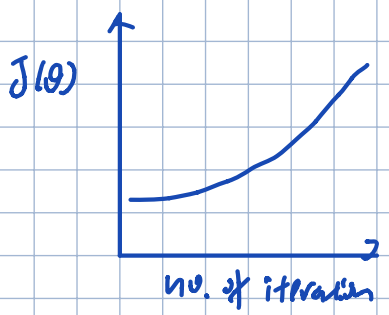

* plot $J(\theta)$ as gradient descent runs.

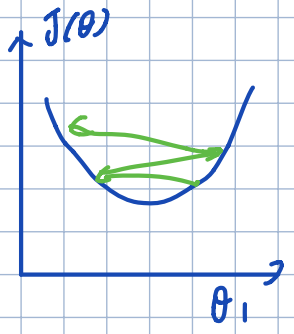
$J(\theta)$ should decrease after every iteration

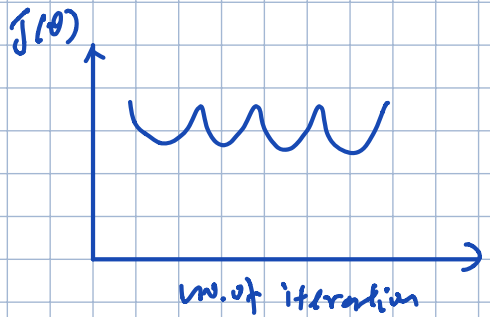
example of automatic convergence test:

Declare convergence if $J(\theta)$ decreases by less than $10^{-3}$ in 1 iteration.

If learning rate is too big



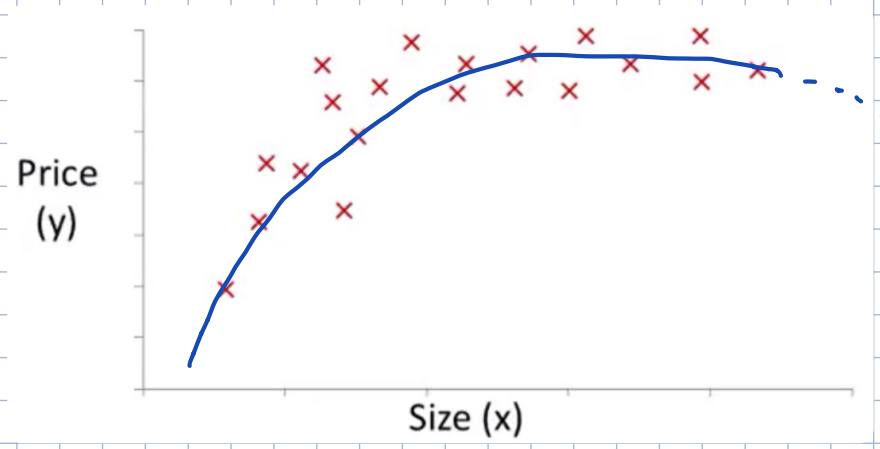




* For sufficiently small $\alpha$, $J(\theta)$ should decrease on every iteration

But if $\alpha$ is too small → slow to converge

# Polynomial Regression




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

$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$

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

feature scaling!

we can also try: $h_\theta(x) = \theta_0 + \theta_1 (size) + \theta_2 \sqrt{(size)}$

---

# Normal Equation

A method to solve for $\theta$ analytically.

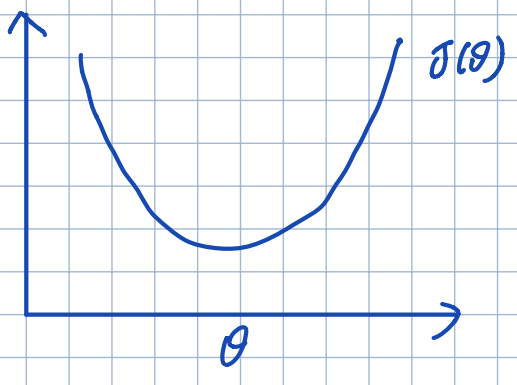
i.e. instead of iterative finding the optima, 1 step will do.

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

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

for $\theta$ is real number!

$\frac{d}{d\theta} J(\theta) = 0$, solve for $\theta$



for regression problem

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

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

$$\frac{\partial}{\partial \theta_j} J(\theta) = 0$$

solve for $\theta_0, \dots, \theta_n$

e.g. $m = 4$ (4 training examples)

new feature $x_0$

| $x_0$ | 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 |

Construct matrix $X$, contains all features

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

$m \times (n+1)$

Construct vector $y$, contains target

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$m \times 1$
m-dimensional vector

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

In the general case, say we have $m$ examples:

i.e. $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$

and $n$ features:

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

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

(design matrix)

e.g. only 1 feature

$$x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix} \qquad X = \begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ \vdots & \\ 1 & x^{(m)} \end{bmatrix} \quad m \times 2$$

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

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

**Normal Equation**

this gives the optimum value of $\theta$ that minimize $J(\theta)$

octave: `pinv(X' * X) * X' * y`

* no need for feature scaling

e.g. $0 \le x_1 \le 1$

$0 \le x_2 \le 1000$

Comparison: $m$ training examples, $n$ features

| Gradient Descent | Normal Equation |
|---|---|
| → need to choose $\alpha$ | → no need to choose $\alpha$ |
| → need many iterations | → don't need to iterate |
| → works well even when $n$ is large | → need to compute $(X^TX)^{-1}$ $n \times n$ → $O(n^3)$ |
| → $O(kn^2)$ | → slow if $n$ is very large ($n > 10k$) |

What if $X^TX$ is non-invertible? (singular / degenerate)

Octave: `pinv` (pseudo-inv) ← works even if $X^TX$ is not invertible

`inv` (inv)

2 most common causes for $X^TX$ not invertible

① Redundant features (linearly dependent)

e.g. $x_1$ = size in feet$^2$

$x_2$ = size in m$^2$

② too many features (e.g. $m \leq n$)

↳ delete some features

↳ use regularisation

---

# week 3

## Classification problems

discrete prediction: Email: Spam / not spam

Online transaction: Fraudulent (Y/N)

Tumor: Malignant / Benign

prediction: $y \in \{0, 1\}$

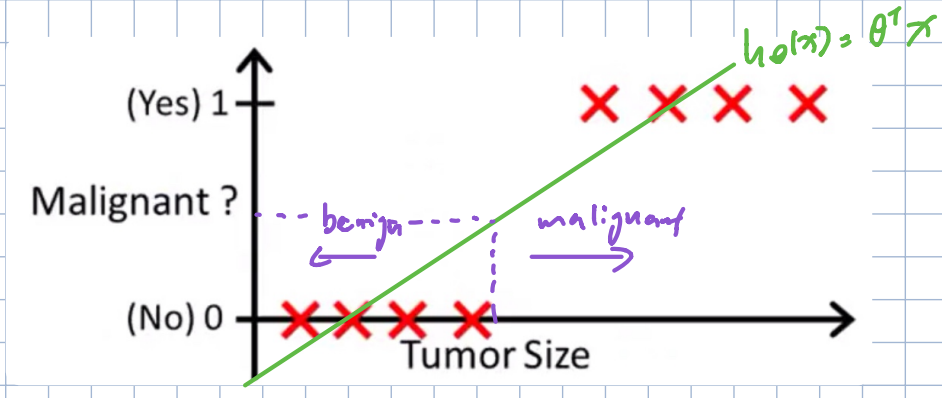
0: "-ve class" (e.g. benign tumor)

1: "+ve class" (e.g. malignant tumor)

→ binary classification problem

→ multiclass classification problem: $y \in \{0, 1, 2, 3\}$

Intuition for logistic regression:




linear regression to this dataset:

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

If we want to make prediction using the linear regression model:

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 $y = 0$

Linear regression is NOT good for classification problem

e.g. $y = 0$ or $y = 1$

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

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

# Logistic Regression

① hypothesis representation $h_\theta(x)$?

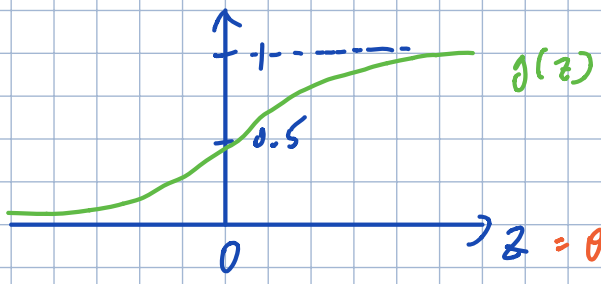
we want $0 \le h_\theta(x) \le 1$

recall linear regression: $h_\theta(x) = \theta^T x$

$\Rightarrow$ for logistic regression: $h_\theta(x) = g(\theta^T x)$ $\Big\}$ $h_\theta(x) = \dfrac{1}{1+e^{-\theta^T x}}$

$$g(z) = \frac{1}{1+e^{-z}} \qquad z \in \mathbb{R}$$

Sigmoid function or logistic function




$g(z)$ asymptotes at 0 and 1

as $z \to \pm\infty$ $\begin{cases} z \to \infty,\ e^{-\infty} \to 0,\ g(z) \to 1 \\ z \to -\infty,\ e^{\infty} \to \infty,\ g(z) \to 0 \end{cases}$

so given that hypothesis is $\boxed{h_\theta(x) = \dfrac{1}{1+e^{-\theta^T x}}}$

$\Rightarrow$ given a training set, we need to pick parameters $\theta$

$\Rightarrow$ then this hypothesis let us make prediction

② hypothesis output interpretation

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

e.g. if $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumor size} \end{bmatrix}$ ← one of the feature used

1 new sample with some $x_0, x_1$ $\Rightarrow h_\theta(x) = 0.7$

after training $\theta$ $\Rightarrow$ 70% chance of tumor being malignant

Formally: $h_\theta(x) = P(y=1 \mid x; \theta)$ ← this probability is parameterized by $\theta$

for classification task : $y = 0$ or $1$

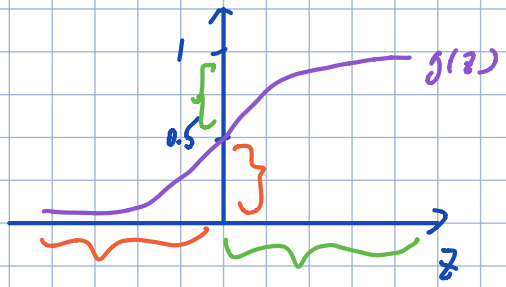
↳ $P(y=0 \mid x;\theta) + P(y=1 \mid x;\theta) = 1$ (for the same patient)

$P(y=0 \mid x;\theta) = 1 - P(y=1 \mid x;\theta)$

## ③ Decision boundary

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

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




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

↳ $h_\theta(x) = g(\theta^T x) \geq 0.5$ when $\theta^T x \geq 0$

predict $y=1$ when $\theta^T x \geq 0$

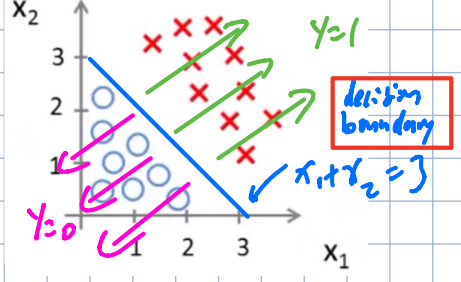
$g(z) < 0.5$ when $z < 0$

↳ $h_\theta(x) = g(\theta^T x) < 0.5$ when $\theta^T x < 0$

predict $y=0$ when $\theta^T x < 0$

---

Feature space




Suppose we have a training set like this and a hypothesis is (after training)

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

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

we know: predict $y=1$ if $-3 + x_1 + x_2 \geq 0$ ($\theta^T x \geq 0$)

↳ $x_1 + x_2 \geq 3$

predict $y=0$ if $-3+x_1+x_2<0$

↳ $x_1+x_2<3$

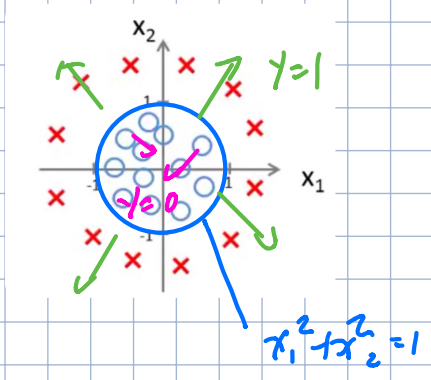
here $x_1+x_2=3 \Rightarrow h_\theta(x)=0.5$

* Decision boundary is a property of hypothesis $h_\theta(x)$, including $\theta_0, \theta_1, \theta_2$.

i.e. not a property of the data set.

↳ once trained, $\theta_0, \theta_1, \theta_2$ will completely define the decision boundary.

## Non-linear boundaries




$$h_\theta(x)=g(\theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_1^2+\theta_4x_2^2)$$

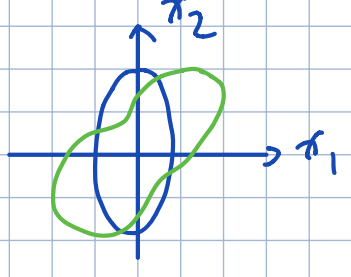
add higher order polynomial terms

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

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

↳ $x_1^2+x_2^2\geq 1$

even higher polynomial terms:




$$h_\theta(x)=g(\theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_1^2+\theta_4x_1^2x_2+\theta_5x_1^2x_2^2+\theta_6x_1^3x_2+\dots)$$

## ④ Cost Function

Recall the logistic regression problem:

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

$m$ examples

$n$ features

$$x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad x_0 = 1$$

$$y \in \{0, 1\}$$

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

how to chose parameters $\theta$

Recall for linear regression:

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

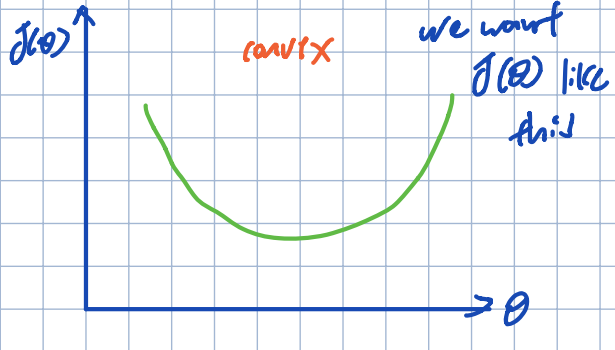
$$= \frac{1}{m} \sum_{i=1}^{m} \text{cost}(h_\theta(x^{(i)}), y^{(i)})$$

where $\text{cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$

↑ a cost for learning algorithm have to pay, if output $h_\theta(x^{(i)})$ where actual label is $y^{(i)}$.

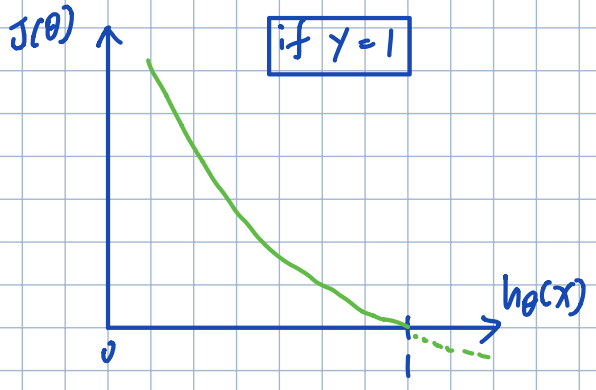
we cannot use $\frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$ as cost function in logistic regression.

If we plug in $h_\theta(x) = \frac{1}{1 + e^{-\theta x}}$ into $J(\theta)$, we get a "non-convex" function, due to non-linearity of function.

For logistic regression

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

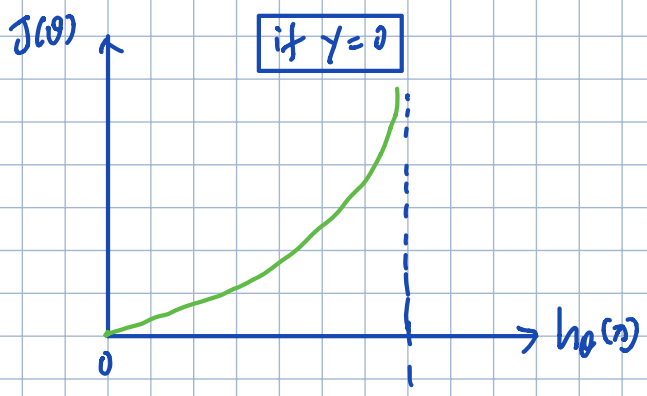



* Cost = 0 if $y=1$, $h_\theta(x)=1$
but as $h_\theta(x) \to 0$, Cost $\to \infty$
capture intuition that if $h_\theta(x)=0$
(i.e. predict $P(y=1|x;\theta)=0$), but $y=1$,
we penalise learning algorithm by a large cost.

Vectorised implementation:

$h = g(X\theta)$

$J(\theta) = \frac{1}{m} \cdot (-y^T \log(h) - (1-y)^T \log(1-h))$




Because $y=0$ or $1$ always, we can simplify the cost function

one-line logistic regression cost function:

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

Logistic regression cost function

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

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

can be derived from principle of maximum likelihood estimation

To fit parameters $\theta$:

$\min_\theta J(\theta) \Rightarrow$ get $\theta$

## ⑤ Gradient Descent to find $\theta$

To make a prediction given new $x$:

$$\text{output } h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \iff p(y=1|x;\theta)$$

Using gradient descent to minimise cost function:

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

Repeat {

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

(simultaneously update all $\theta_j$)

}

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

vectorised implementation:

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$

* looks exactly same to linear regression!

But $h_\theta(x)$ is different { linear $h_\theta(x) = \theta^T x$; logistic $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

* feature scaling also applied to gradient descent for logistic regression.

## ⑥ Advanced Optimisation

Goal: allows logistic regression run more quickly with gradient descent, also allows very large ML problems, with large no. of features to be run.

Optimisation algorithm so far:

cost function $J(\theta)$, want $\min_\theta J(\theta)$

Given $\theta$, we have code that can compute:

$J(\theta)$ ← to monitor convergence

$\frac{\partial}{\partial \theta_j} J(\theta)$ ← needed for gradient descent (for $j = 0, 1, \dots, n$)

what gradient descent does is:

Repeat {

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

}

* Gradient descent is not the only algorithm for optimisation $J(\theta)$
  - ↳ Conjugate gradient
  - ↳ BFGS
  - ↳ L-BFGS

Advantages:

* no need to manually pick $\alpha$
* often faster than gradient descent

Disadvantages:

* more complex

e.g. $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

If we want to minimise $\min_\theta J(\theta)$

then $\theta_1 = 5$, $\theta_2 = 5$

returns

```
function [jVal, gradient] = costFunction(theta)
```

$J(\theta)$ →
```
jVal = (theta(1)-5)^2 + (theta(2)-5)^2;
```

```
gradient = zeros(2,1);
```
$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$\frac{\partial}{\partial \theta_1} J(\theta)$ →
```
gradient(1) = 2*(theta(1)-5);
```

$\frac{\partial}{\partial \theta_2} J(\theta)$ →
```
gradient(2) = 2*(theta(2)-5);
```
} element of gradient

```
% call advanced optimisation
options = optimset('GradObj', 'on', 'MaxIter', '100');
```
↑ data structure that stores the options we want

```
initialTheta = zeros(2,1);
```

$\theta \in \mathbb{R}^d$, $d \geq 2$

```
[optTheta, functionVal, exitFlag]
    = fminunc(@costFunction, initialTheta, options);
```
↑ function optimisation unconstrained

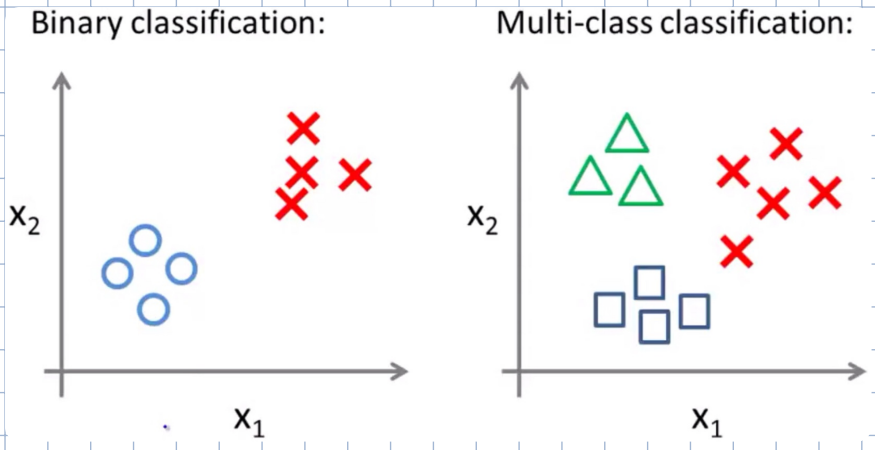
↖ pointer to cost function

## Logistic Regression

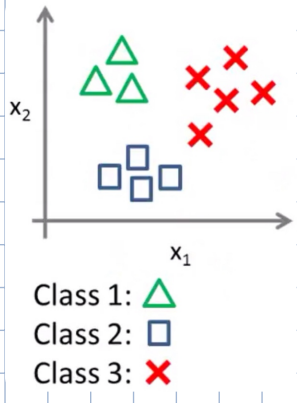
$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ → theta(1) in octave

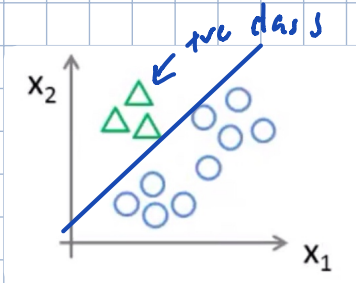
```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(θ)];
    gradient(1) = [code to compute ∂/∂θ0 J(θ)];
    gradient(2) = [code to compute ∂/∂θ1 J(θ)];
    ⋮
    gradient(n+1) = [code to compute ∂/∂θn J(θ)];
```

## ③ Multi-class classification: one-vs-all




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



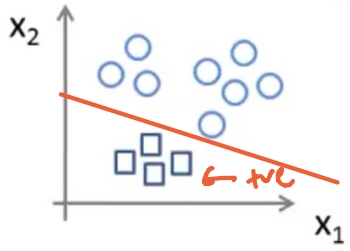




$h_\theta^{(1)}(x)$ ← for 1st class

△ = 1

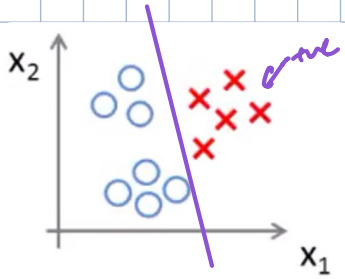
○ = 0

$P(y=1|x;\theta)$




$h_\theta^{(2)}(x)$

$P(y=2|x;\theta)$




$h_\theta^{(3)}(x)$

$P(y=3|x;\theta)$

In summary:

$h_\theta^{(i)}(x) = P(y=i|x;\theta)$

$(i = 1, 2, 3)$

⇒ trained 3 classifiers

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

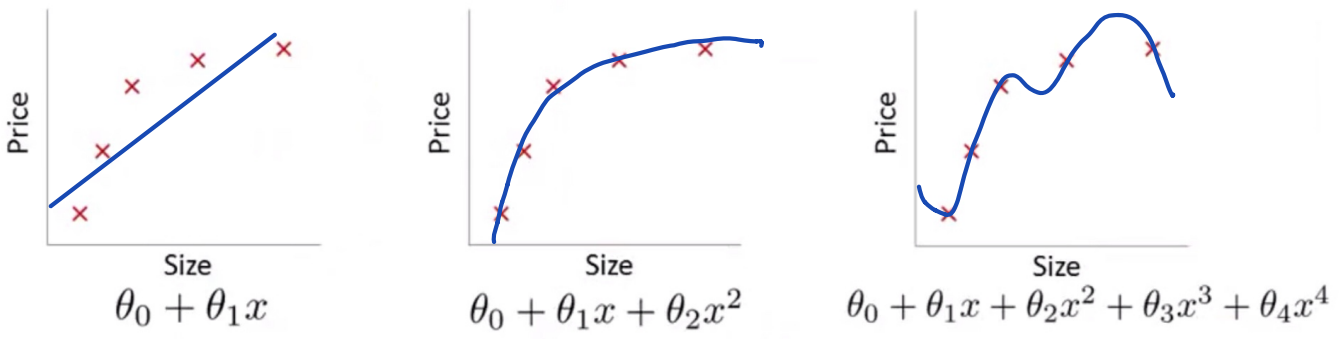
* On a new input $x$, to make a prediction, pick the class $i$ that maximises

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

i.e. run all 3 classifier, pick the class $i$ that maximises the 3. (most confident)

## Overfitting

e.g. linear regression



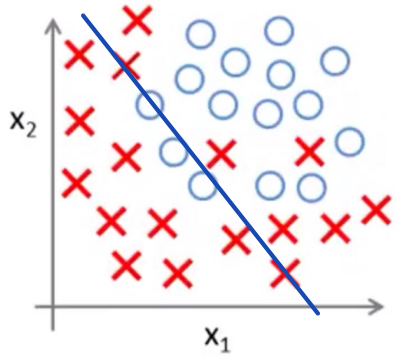

$\theta_0 + \theta_1 x$ $\quad$ $\theta_0 + \theta_1 x + \theta_2 x^2$ $\quad$ $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

under fit
high bias
↙
pre-conceptions

overfit
high variance

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

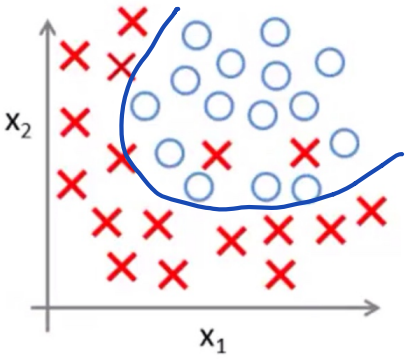
e.g. logistic regression



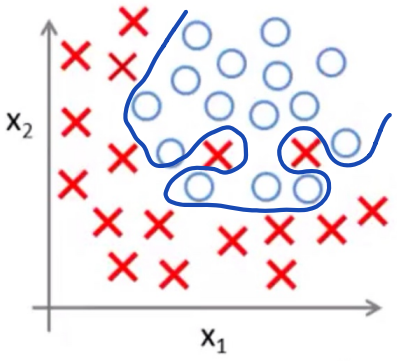

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

($g$ = sigmoid function)

under fit




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




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

overfit

## Addressing overfitting

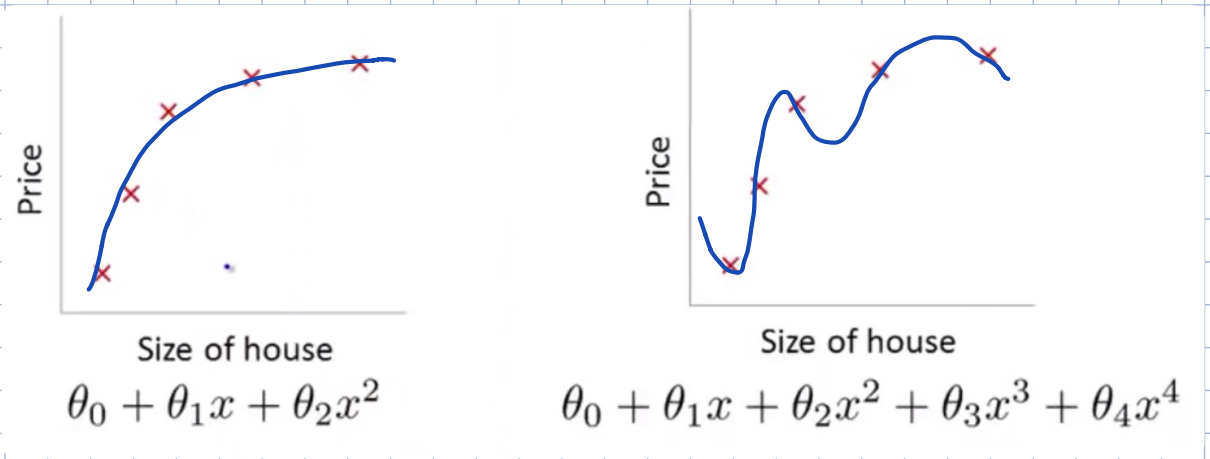
2 main options:

1. Reduce number of features
   - ↳ manually select which features to keep
   - ↳ model selection algorithm

2. Regularisation
   - ↳ keep all features, but reduce magnitude/value of $\theta_j$
   - ↳ works well when we have a lot of features, each of which contributes a bit to predicting $y$.

# Regularisation: cost function

## Intuition




suppose we penalise and make $\theta_3, \theta_4$ really small

the usual squared error cost function + some extra terms on $\theta_3, \theta_4$

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

If we were to minimise this function, the only way is for $\theta_3$ and $\theta_4$ to be small. $\Rightarrow \theta_3 \approx \theta_4 \approx 0$ likely.

↳ similar to get rid of the $\theta_3, \theta_4$ term

## Regularisation:

small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$

↳ simpler hypothesis

↳ less prone to overfitting

# Regularised Linear Regression

e.g. housing: → features: $x_1, x_2, \dots x_{100}$ ← hard to pick in advance the relevant features

→ parameters: $\theta_0, \theta_1, \dots, \theta_{100}$

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

← shrink all parameters

$\lambda$ → regularisation parameter

$\theta_1, \dots, \theta_{100}$ are all shrunk as a result

↳ by convention, $\theta_0$ (i.e. the intercept) is NOT penalised.

↳ in practice, include $\theta_0$ or not makes little difference

$\lambda$ (regularisation parameter) controls a trade off between 2 different goals:

① $\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ : fit training data well

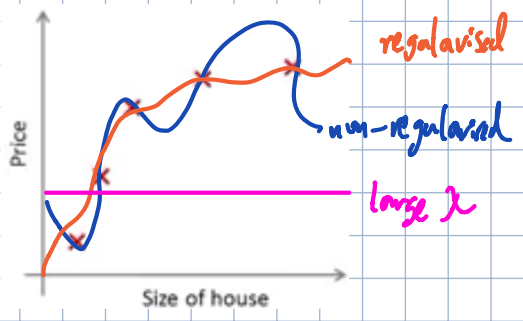
② $\sum_{j=1}^{n}\theta_j^2$ : keeps the parameter ($\theta$) small

If $\lambda$ is set to an extremely large value:

then all $\theta \approx 0$

↳ $h_\theta(x) = \theta_0 + \cancel{\theta_1 x} + \cancel{\theta_2 x^2} + \cancel{\theta_3 x^3} + \dots$

$h_\theta(x) = \theta_0$ ⇒ flat horizontal line ⇒ underfit ⇒ high bias



# Regularised Linear Regression

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

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

## Gradient descent

Repeat {

treat $\theta_0$ separately ∴ no regularisation on $\theta_0$

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

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

$\rightarrow \frac{\partial}{\partial\theta_j}J(\theta)$ + regularised

} equivalently, take $\theta_j$ out

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

$1 - \alpha\frac{\lambda}{m}$ usually $< 1$, e.g. 0.99

$\Rightarrow \theta_j \rightarrow \theta_j \times 0.99 \Rightarrow \theta_j^2$ smaller

* This is just a mathematical representation, in computation, we just perform gradient descent on the regularised $J(\theta)$.

## Normal equation

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}_{m \times (n+1)} \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

$\frac{\partial}{\partial \theta_j} J(\theta) = 0$ → get $\theta$

$$\min_\theta J(\theta) \Rightarrow \theta = (X^T X + \lambda \underbrace{\begin{bmatrix} 0 & & \\ & 1 & \\ & & \ddots \end{bmatrix}}_{(n+1)\times(n+1)})^{-1} X^T y$$

Suppose $m \leq n$ (example, feature)

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

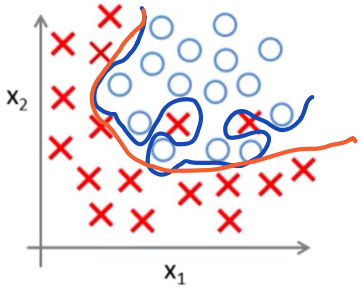
$X^T X$ non-invertable

| |
|---|
| $m \leq n$ non-invertable |
| $m \geq n$ may be non-invertable |

If $\lambda > 0$

$$\theta = (\underbrace{X^T X + \lambda \begin{bmatrix} 0 & & \\ & 1 & \\ & & \ddots \end{bmatrix}}_{\text{invertable}})^{-1} X^T y$$

## Regularised Logistic Regression




$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

easy to overfit

Cost function:

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

## Gradient descent

Repeat {

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

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

$(j = 1, 2, \ldots, n)$

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

## Advanced Optimisation

**Advanced optimization**

fminunc(@costFunction, ..., $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ theta(1), theta(2), theta(n+1))

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(θ)];
        → J(θ) = [ −(1/m) Σ_{i=1}^{m} y^(i) log(h_θ(x^(i))) + (1 − y^(i)) log 1 − h_θ(x^(i)) ] + (λ/2m) Σ_{j=1}^{n} θ_j^2
    → gradient(1) = [code to compute ∂/∂θ_0 J(θ)];
            (1/m) Σ_{i=1}^{m} (h_θ(x^(i)) − y^(i)) x_0^(i)
    → gradient(2) = [code to compute ∂/∂θ_1 J(θ)];
            (1/m) Σ_{i=1}^{m} (h_θ(x^(i)) − y^(i)) x_1^(i) + (λ/m) θ_1
    → gradient(3) = [code to compute ∂/∂θ_2 J(θ)];
            (1/m) Σ_{i=1}^{m} (h_θ(x^(i)) − y^(i)) x_2^(i) + (λ/m) θ_2
    ⋮
    gradient(n+1) = [code to compute ∂/∂θ_n J(θ)];
```

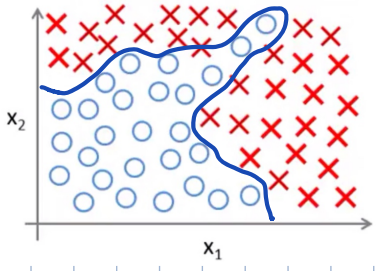
Andrew Ng

# week 4

## Neural Networks

### Representation: Non-linear hypotheses

motivation:




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

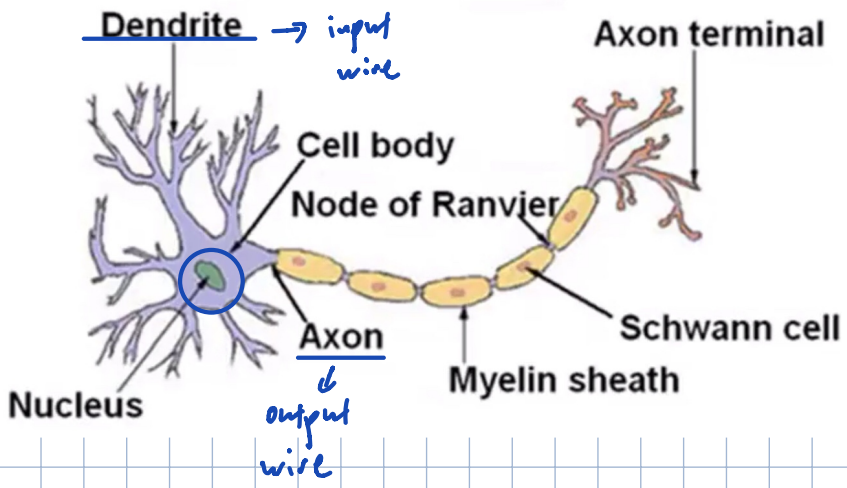
If we want to apply logistic regression for the above classification, we can include lots of polynomial terms

↳ works well for 2 features

↳ will overfit for a lot of features

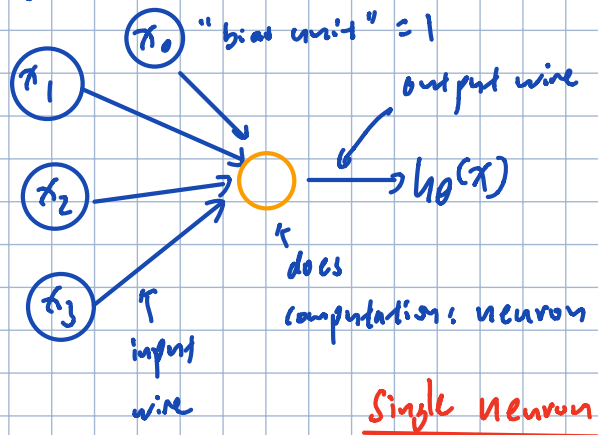
↳ can do feature selection, but will not get nice decision boundary

### Model Representation




fun fact:
Neuron in the brain

## Logistic unit : a simple model




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

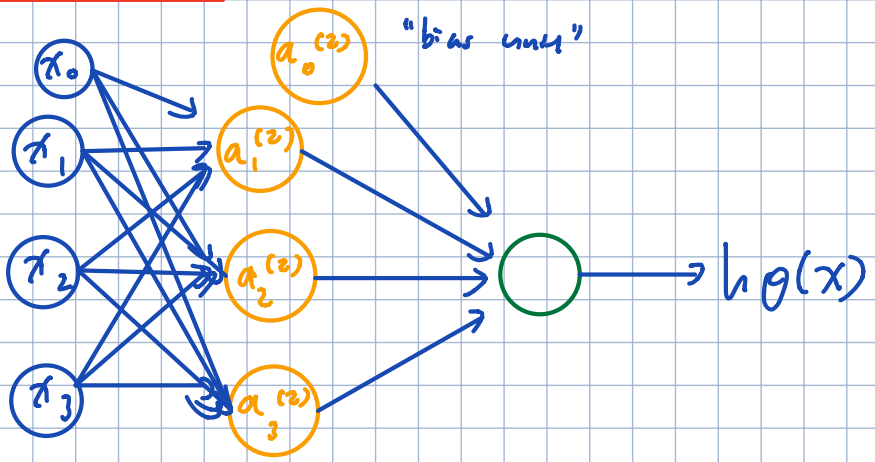
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

↑ weight

**Single neuron**

↳ an artificial neuron with a sigmoid / logistic activation function: $g(z) = \frac{1}{1+e^{-z}}$

## Neural Network




layer 1 — input layer — $x$

layer 2 — hidden layer

layer 3 — output layer — $y$

i.e. $a_1^{(2)}$ : activation of the first unit in layer 2

"neuron"

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

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

The computation that are represented by the diagrams

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

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

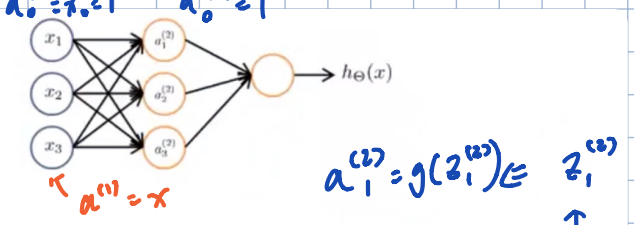
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

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

Here, 3 input, 3 hidden layer $\Rightarrow \Theta^{(1)} \in \mathbb{R}^{3\times 4}$

$\Rightarrow$ 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




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

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \quad \uparrow z_2^{(2)}$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \quad \nearrow z_3^{(2)}$$

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

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

$\Theta^{(1)} x$

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

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

$a^{(2)} = g(z^{(2)})$ $\quad$ element-wise application

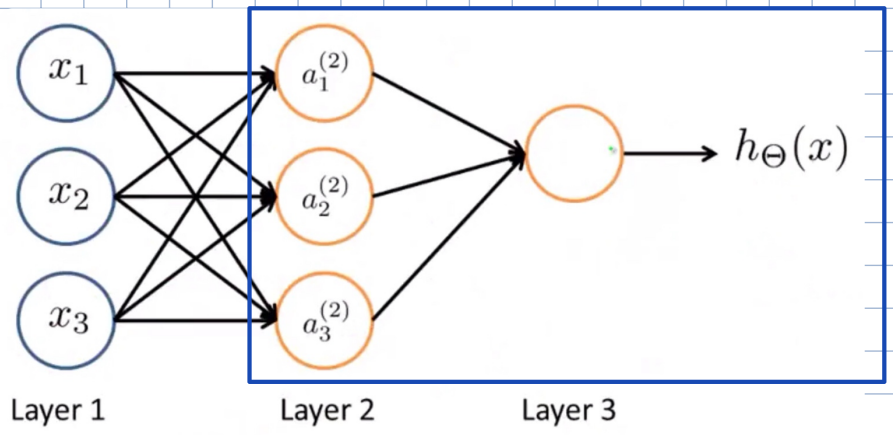
$a^{(2)} \in \mathbb{R}^3$ $\quad z^{(2)} \in \mathbb{R}^3$

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

↳ $z^{(3)} = \Theta^{(2)} a^{(2)}$

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

# Neural Network learning its own features:



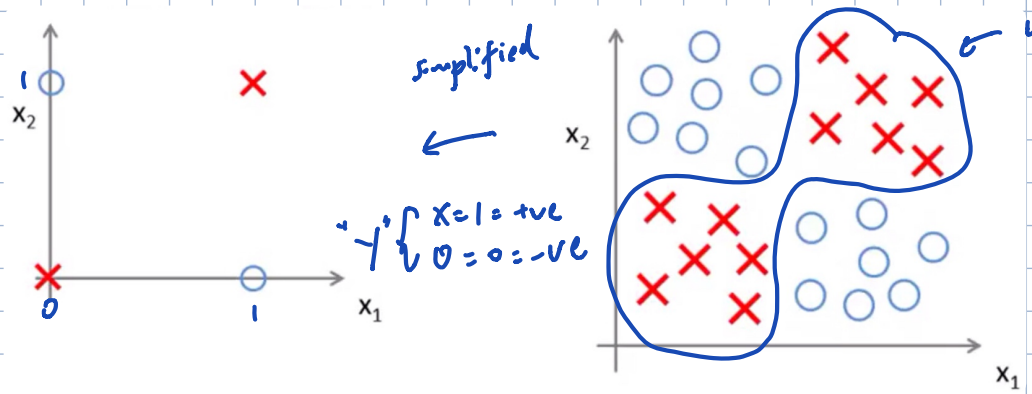

$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

↑ logistic regression with 3 "features": $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$

but these are learned from actual feature $x_1, x_2, x_3$ through $\Theta^{(1)}$ ←

## Examples and Intuitions

Consider we have features $x_1$ and $x_2$ that are binary values. (0 or 1)




$Y = x_1$ XOR $x_2$

$x_1$ XNOR $x_2$ ↑

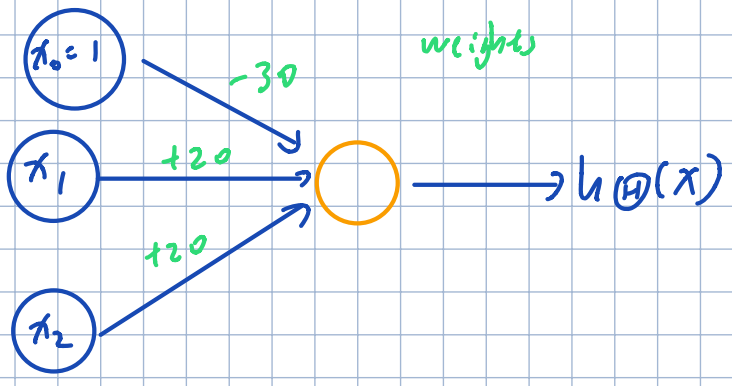
| A | B | A XOR B | A XNOR B |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

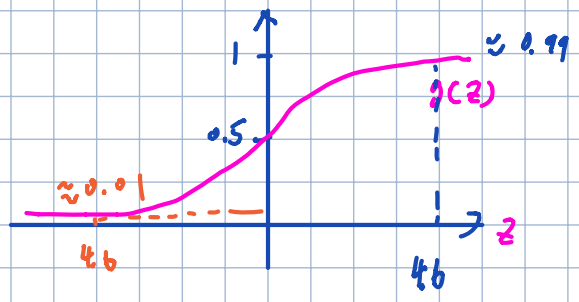
In order to build up to a network that fits the XNOR example, we start with a slightly simpler one and show a network that fits the AND function:

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

$y = x_1 \text{ AND } x_2$

| A | B | A AND B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




mathematically, the hypothesis:

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

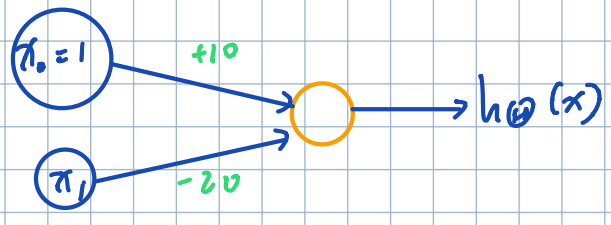
$\Theta_{10}^{(1)}$ $\Theta_{11}^{(1)}$ $\Theta_{12}^{(1)}$

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

AND

$\hookrightarrow h_\Theta(x) \approx x_1 \text{ AND } x_2$

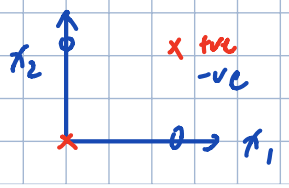
Negation: NOT $x_1$



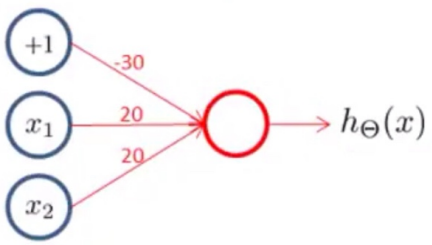

| $x_1$ | $h_\Theta(x)$ |
|---|---|
| 0 | $g(10) \approx 1$ |
| 1 | $g(-10) \approx 0$ |

$$h_\Theta(x) = g(10 - 20x_1)$$

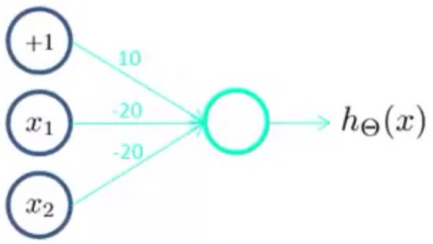
## $x_1$ XNOR $x_2$



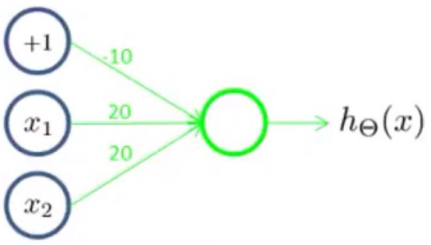

**Putting it together:** $x_1$ XNOR $x_2$



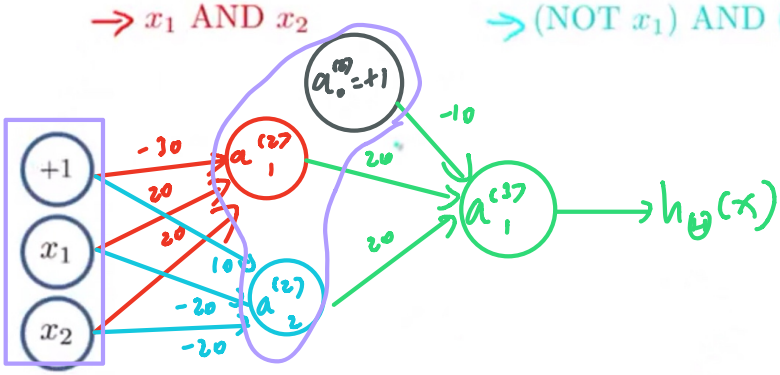

$x_1$ AND $x_2$




(NOT $x_1$) AND (NOT $x_2$)




$x_1$ OR $x_2$




| $x_1$ | $x_2$ | $a_1^{(2)}$ | $a_2^{(2)}$ | $h_\Theta(x)$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

input hidden output

## Neural Network: multi-class classification

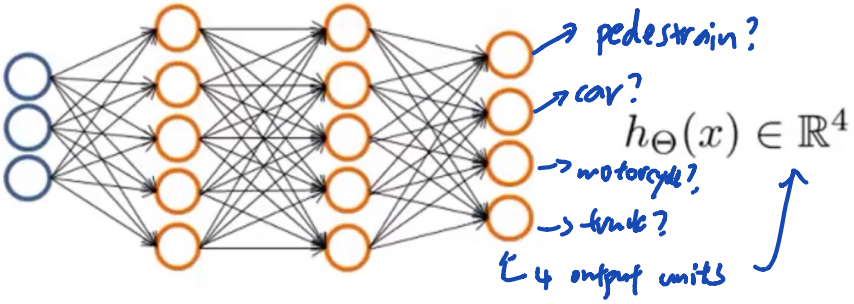
**Multiple output units: One-vs-all.**

Pedestrian

Car

Motorcycle

Truck

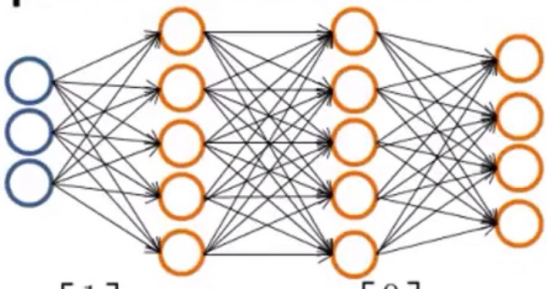



Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

an extension of 1-vs-all method

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

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

$x^{(i)}$: image

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

$\mathbb{R}^4$

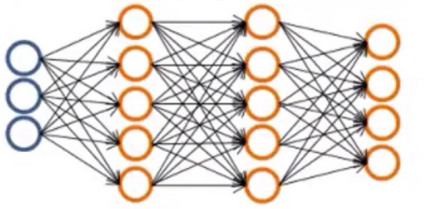
# Week 5

## Neural Networks: Cost function

Cost function for fitting the parameters of the network.

### Classification problems

**Neural Network (Classification)**




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

$L =$ total no. of layers in network e.g. $L=4$

$s_l =$ no. of units (not counting bias unit) in layer $l$ e.g. $S_1 = 3$, $S_2 = 5$, $S_4 = S_L = 4$

2 types of classification problems:

① Binary classification

$y = 0$ or $1$


1 output unit

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

$S_L = 1$ or $K=1$

$S_L$ = final layer

② Multi-class classification (K classes)

$y \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian, car, bike, truck

K output units

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

$S_L = K$ $(K \geq 3)$

The cost function we use for the neural network is going to be a generalisation of the one we use for logistic regression.

## Logistic Regression cost function

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

Instead of having just 1 compression output unit, we have K of them

## Neural Network cost function

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta(x^{(i)})\right)_k + (1-y_k^{(i)}) \log\left(1-(h_\Theta(x^{(i)}))_k\right)\right]$$

↳ sum over K outputs for each training example

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} \left(\Theta_{ji}^{(l)}\right)^2$$

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

↙ $K=1$ for binary classification

K-dimensional vector

$\Theta$: $s_{l+1} \times s_l$ ($j$ : $i$)

↳ don't sum over bias values:

i.e. don't sum over the terms where $i = 0$

↳ $\Theta_{j0}^{(2)} x_0 + \Theta_{j1}^{(2)} x_1 + \dots$ 1st hidden layer

$x_0$ not included when $i=0$

* double sum simply adds up the logistic regression costs calculated for each cell in the output layer.

* triple sum simply adds up the square of all the individual $\Theta$s in the entire network

# Back propagation algorithm

An algorithm that is trying to minimise the cost function,

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

$$\min_{\Theta} J(\Theta)$$

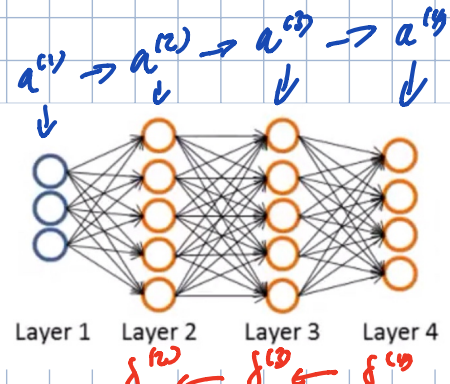
Need code to compute: either gradient descent or other advanced algorithm

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

$\Theta_{ij}^{(l)} \in \mathbb{R}$

## Gradient computation

consider the case with one training example: $(x, y)$




① Apply forward propagation:

↳ compute what a hypothesis actually outputs given this input $x$.

$a^{(1)} = x$

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

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

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

$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)

$z^{(4)} = \Theta^{(3)} a^{(3)}$

$a^{(4)} = h_\Theta(x) = g(z^{(4)})$

② Back propagation: → compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

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

↳ for each node

recall $a_j^{(l)}$: activation of the $j^{th}$ unit in layer $l$.

Here, for each output unit (layer $L=4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j \quad \rightarrow (h_\Theta(x))_j$$

earlier layers:

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

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

vector element wise multiplication vector

derivatives of activation function $g$ evaluated on input value given by $z^{(3)}$

$$= a^{(3)} .* (1 - a^{(3)})$$

vectors vector of 1's vectors

$$= a^{(2)} .* (1 - a^{(2)})$$

no $\delta^{(1)}$; just features

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

(ignoring $\lambda$; regularisation $\lambda = 0$)

Putting together: to get $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Back propagation algorithm (for the entire training set)

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

→ set $\Delta_{ij}^{(l)} = 0$ (for all $l, i, j$), eventually used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$, in the process, they are used as accumulators that will slowly add things in order to get $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$.

→ loop through training set

For $i = 1$ to $m$ — for $i^{th}$ iteration, works with $(x^{(i)}, y^{(i)})$

↳ set $a^{(1)} = x^{(i)}$ — set input $x$ as $a^{(1)}$, first layer

↳ perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

↳ using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$ — hypo − actual

↳ compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots \delta^{(2)}$ — back propagation

↳ $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ — vectorise: $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

* not run from $1 \to m$. should be no. of nodes in $l+1$ layer

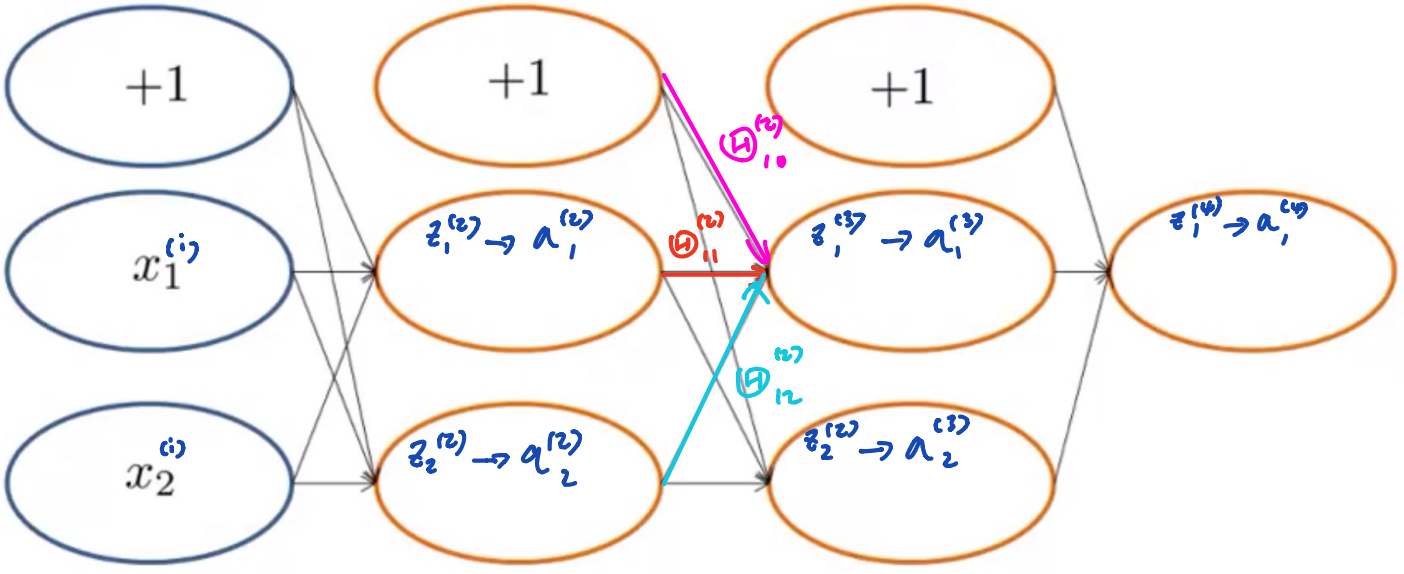
→ $D_{ij}^{(l)} = \frac{1}{m}\left(\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}\right)$ if $j \neq 0$

$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$ (bias terms)

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

↳ this gradient then can be used for gradient descent etc.

# Back propagation : intuition




↑ feed $(x^{(i)}, y^{(i)})$

$z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)}$

## What is backpropagation doing?

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

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

$K=1$

Focusing on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

$$\text{cost(i)} = y^{(i)} \log(h_\Theta(x^{(i)})) + (1 - y^{(i)}) \log \overline{(1 - h_\Theta(x^{(i)}))}$$

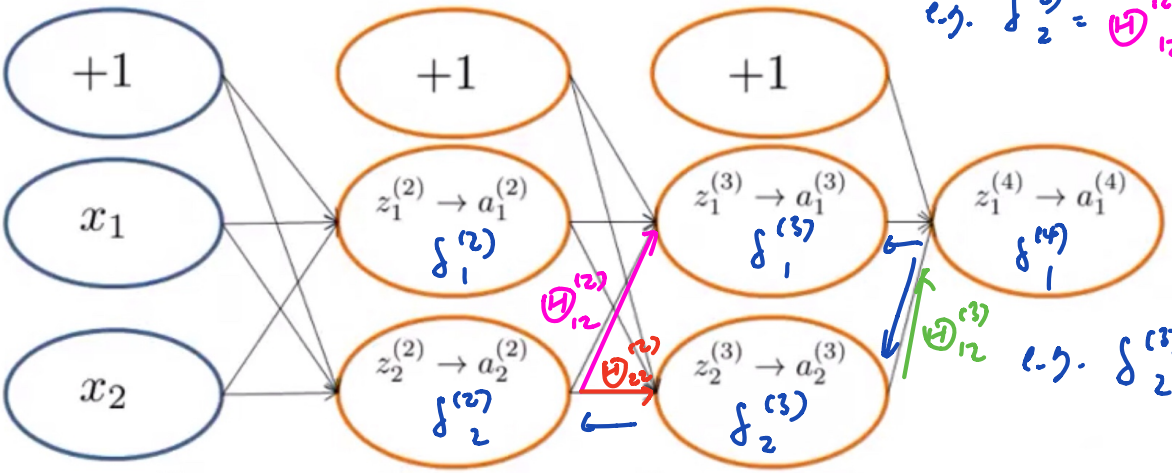
← for $(x^{(i)}, y^{(i)})$

(Think of $\text{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$) ↙ intuitively

I.e. how well is the network doing on example i?

$\delta_1^{(4)} = a_1^{(4)} - y^{(i)}$

e.g. $\delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \delta_2^{(3)}$




e.g. $\delta_2^{(3)} = \Theta_{12}^{(3)} \delta_1^{(4)}$

$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit $j$ in layer $l$).

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost(i)}$ (for $j \geq 0$), where

$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$

$(1 - h_\Theta(x^{(i)}))$

varying $z_j^{(l)}$ to change the value of network's output → change cost function

↳ so $\delta$ are measure of how much would we like to change the neural network's weights, in order to affect these intermediate values ($z$s). ⇒ affect final output $h_\Theta(x)$ ⇒ affect overall cost.

## Implementation note: unrolling parameters

unrolling parameters from matrices into vectors, which we need in order to use the advanced optimisation routines.

Advanced optimisation

```
function [jVal, gradient] = costFunction(theta)
```

→ input $\theta$
→ output $J(\theta)$, $\frac{\partial}{\partial \theta} J(\theta)$

...

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

Both routines assume that theta and initialTheta are parameter vectors (e.g. $\in \mathbb{R}^{n+1}$). Also assumes that the gradient (2nd return value) is also a vector $\in \mathbb{R}^{n+1}$.

This works fine when we were using logistic regression, but now for neural network, our parameters are no longer vectors.

e.g. for a full neural network (L=4), we have

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ → matrices (Theta1, Theta2, Theta3)

$\Theta^{(l)}$: no. of nodes in $l$ layer × (no. of feature + 1)

$D^{(1)}, D^{(2)}, D^{(3)}$ → matrices (D1, D2, D3)

gradient that are returned

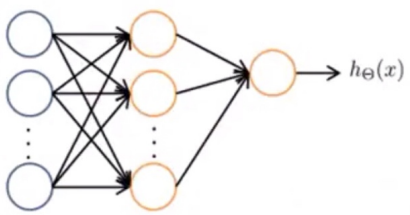
↳ "unroll" these matrices into vectors, so they end up being suitable to be used in the octave code above (for fminunc)

**Example**

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

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

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




In Octave, convert these matrices → vectors

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

Vector ⟶ matrix

pull the 1st 110 terms

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

e.g. Learning Algorithm

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

→ unroll to get initialTheta to pass to

fminunc(@costFunction, initialTheta, options)

implementation of the cost function:

rolled into 1 vector

```
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

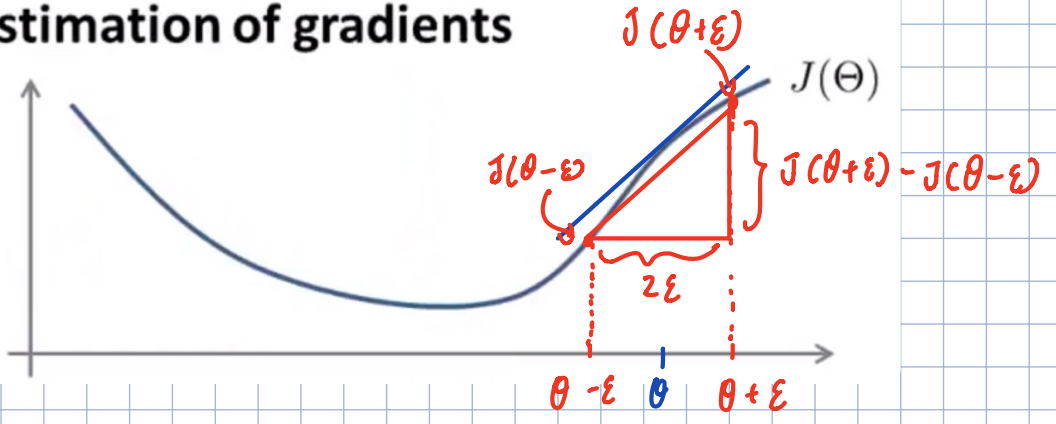
With gradient descent implementation, sometimes the cost function $J(\Theta)$ of the neural network decreases with each iteration, the implementation can be buggy.

⇒ gradient checking can be used to eliminate these problems.

⇒ make sure that forward prop and backward prop is correct.

e.g. assume $\theta \in \mathbb{R}$ (i.e. $\theta$ is just a real number)

## Numerical estimation of gradients




procedure for numerically approximating the derivative

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

($\varepsilon = 10^{-4}$, small value)

$\frac{J(\theta+\varepsilon) - J(\theta)}{\varepsilon}$ ⇒ 1-sided differences

2-sided differences

octave:

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

we considered here $\theta$ is a rolled number,

more generally, $\theta$ can be a vector parameter

$\theta \in \mathbb{R}^n$ (e.g. $\theta$ is "unrolled" version of $(W^{[1]}, W^{[2]}, W^{[3]})$

↳ $\theta = [\theta_1, \theta_2, \dots, \theta_n]$

similar idea

↳ $\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\varepsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1-\varepsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\varepsilon}$

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

$\vdots$

$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \dots, \theta_n+\varepsilon) - J(\theta_1, \theta_2, \dots \theta_n-\varepsilon)}{2\varepsilon}$

Octave implementation:

```
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;
```

unrolled version: a long list of all $\theta$ in the neural network

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i + \varepsilon \\ \vdots \\ \theta_n \end{bmatrix}$$

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

- check that gradApprox $\approx$ DVec
  - [regularized gradients averaged over the training set]
  - from back prop, all the $D_1, D_2 \dots$
  - if true, much more confident that the implementation of back prop is correct,
  - then plug these DVec vectors into gradient descent or other advanced algorithm.

## Numerical Gradient Checking

- implement backprop to compute DVec (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$)
- implement numerical gradient check to compute gradApprox.
- make sure they give similar values.
- turn off gradient checking. Using back prop code for learning.

## Random Initialisation

When running gradient descent and advanced optimisation method, we need initial value for $\theta$.

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

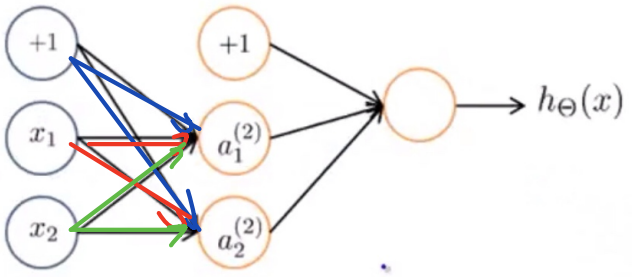
consider gradient descent

↳ do we set initialTheta = zeros(n, 1) ?

↳ works OK for logistic regression

↳ DOES NOT work for neural network

why? Consider training the following neural network




$$\Theta_{ij}^{(l)} = 0 \text{ for all } i, j, l.$$

this gives $a_1^{(2)} = a_2^{(2)} \Rightarrow \delta_1^{(1)} = \delta_2^{(2)}$

$$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)$$

after 1 gradient descent iteration/update : $\Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

all will end up some non-zero value

⇒ after each update, parameters corresponding to inputs going into each of 2 hidden units are identical.

↳ the neural network really can't compute interesting/complex functions. All nodes in hidden layer will compute the same thing.

* Need random initialisation : symmetry breaking

↳ Initialise each $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$

i.e. $-\varepsilon \leq \Theta_{ij}^{(l)} \leq \varepsilon$

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

`theta2 = rand(1,11)*(2*INIT_EPSILON) - INIT_EPSILON;`

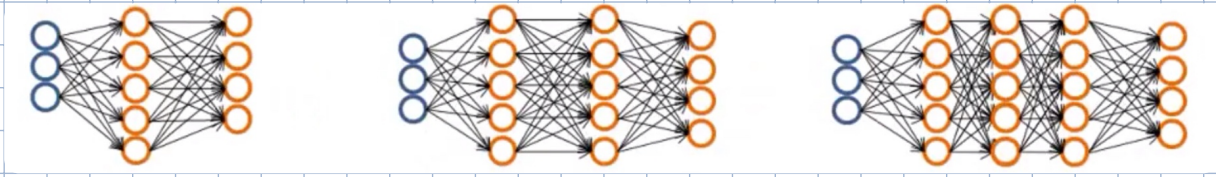
random $10 \times 11$ matrix (between 0 and 1)

$[-\varepsilon, \varepsilon]$ this $\varepsilon$ is <u>not the same</u> as the one used in gradient checking.

## Putting together: Neural Network Training

### Choosing architecture

when training a neural network, first we need to pick a network architecture (i.e. connectivity pattern between neurons)



No. of input units: Dimension of feature $x^{(i)}$

No. of output units: No. of classes (for classification problems)

$y \in \{1, 2, \ldots, 10\} \Rightarrow y = \begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\\vdots\\0\end{bmatrix}, \ldots, \begin{bmatrix}0\\0\\\vdots\\1\end{bmatrix}$

if $y=5$, then at output nodes, the vector $= \begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix}$

For the hidden layers:

reasonable default: 1 hidden layer, or >1 hidden layer, have same no. of hidden units in every layer (usually the more the better).

$\Rightarrow$ usually the no. of hidden units in each layer will be comparable to the dimension of $x$ (e.g. 3x, 5x)

# Training neural network

① Randomly initialise weights
usually randomise the weights to small values near 0.

② Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$ (training example $i$)

③ Implement code to compute cost function $J(\Theta)$

④ Implement back prop to compute $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$

usually with a for loop over the training examples

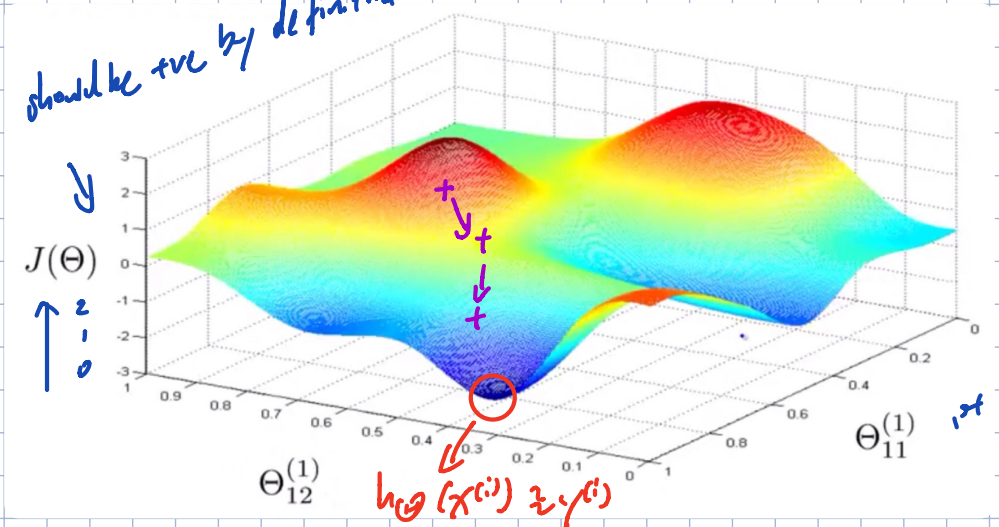
for $i = 1 : m$ {
perform forward prop and back prop using example $(x^{(i)}, y^{(i)})$
(get activations $a^{(l)}$ and $\delta^{(l)}$ for $l = 2, \dots, L$)
$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$
...
}
...
$D_{ij}$
compute $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$; (including regularisation terms)

⑤ Use gradient checking to compare $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$ computed using back prop vs using numerical estimate of gradient of $J(\Theta)$
↳ then disable gradient checking code

⑥ Use gradient descent or advanced optimisation method with back prop to try minimise $J(\Theta)$ as a function of parameters $\Theta$.

$\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$ ; $J(\Theta)$ is non-convex for neural network
↳ might have local minima

should be +ve by definition




1st layer

$J(\Theta)$ measures how well the neural network fits the training data.

back prop computes the direction of the gradient

↳ gradient descent takes little steps downhill.

# Week 6

## Debugging a learning algorithm

Suppose we have implemented regularised linear regression to predict (e.g. house price)

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

However, if we test the hypothesis on a new set of houses, we find that it makes very large errors in its predictions, what should we do next?

- → Get more training examples
- → try smaller sets of features
- → try getting additional features
- → try adding polynomial features ($x_1^2$, $x_2^2$, $x_1x_2$, etc)
- → try decrease / increase $\lambda$

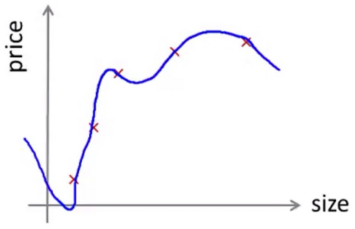
⇒ Machine Learning diagnostic:
A test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

Takes time to implement, but a good use of time.

# Evaluating a hypothesis

Evaluating a hypothesis that has been learned by our algorithm.

**Evaluating your hypothesis**




Fails to generalize to new examples not in training set.

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
$x_{100}$

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

Hypotheses with low training error not necessary a good thing → overfit

The standard way to evaluate a learned hypothesis is as follows:

Dataset:

| | Size | Price | |
|---|---|---|---|
| 70% | 2104 | 400 | training set |
| | 1600 | 330 | |
| | 2400 | 369 | |
| | 1416 | 232 | |
| | 3000 | 540 | |
| | 1985 | 300 | |
| | 1534 | 315 | |
| 30% | 1427 | 199 | test set |
| | 1380 | 212 | |
| | 1494 | 243 | |

training set → $(x^{(1)}, y^{(1)}) \ \vdots \ (x^{(m)}, y^{(m)})$

test set → $(x^{(1)}_{test}, y^{(1)}_{test}) \ \vdots \ (x^{(m_{test})}_{test}, y^{(m_{test})}_{test})$

$m_{test}$: no. of test examples

→ better to randomly split train/test set

## Training/testing procedure for linear regression

→ learn parameter $\theta$ from training data (minimising training error $J(\theta)$)

of 70% data

→ Compute test set error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)}\right)^2 \quad \text{(linear regression)}$$

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log h_\theta(x_{test}^{(i)})$$

(logistic regression)

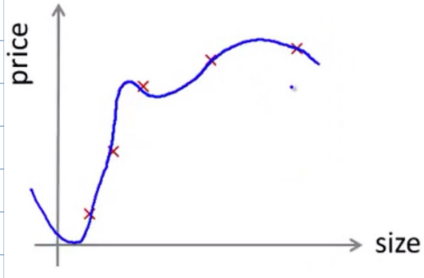
↳ Sometimes there is an alternative test set that might be easier to interpret: Misclassification error (0/1 misclassification error):

$$err(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5,\ y=0 \text{ or if } h_\theta(x) < 0.5,\ y=1 \\ 0 & \text{otherwise} \end{cases} \Big\} \text{error}$$

$$\text{test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x_{test}^{(i)}), y^{(i)})$$

## Model selection

**Overfitting example**




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

Once parameters $\theta_0, \theta_1, \ldots, \theta_4$ were fit to some set of data (training set), the error of the parameters as measured on that data (the training error $J(\theta)$) is likely to be lower than the actual generalization error.

# Model selection

after fitting training data

d=1 1. $h_\theta(x) = \theta_0 + \theta_1 x \longrightarrow \theta^{(1)} \Rightarrow \begin{pmatrix}\theta_0\\ \theta_1\end{pmatrix}$ for $d=1 \rightarrow J_{test}(\theta^{(1)})$

d=2 2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \longrightarrow \theta^{(2)} \rightarrow J_{test}(\theta^{(2)})$

d=3 3. $h_\theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_3 x^3 \rightarrow \theta^{(3)} \rightarrow J_{test}(\theta^{(3)})$

⋮

d=10 10. $h_\theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_{10} x^{10} \longrightarrow \theta^{(10)} \rightarrow J_{test}(\theta^{(10)})$

as if a new parameter in this algorithm

d = degree of polynomial we pick

based on lowest lowest test set error

say choose $\theta_0 + \ldots + \theta_5 x^5$

How well does this model generalise? Report test set error $J_{test}(\theta^{(5)})$

It is likely not fair:

Problem: $J_{test}(\theta^{(5)})$ is likely to be an optimistic estimate of generalisation error. i.e. our extra parameter (d = degree of polynomial) is fit to test set.

To address this problem

Dataset:

| | Size | Price | |
|---|---|---|---|
| 60% | 2104 | 400 | training set |
| | 1600 | 330 | |
| | 2400 | 369 | |
| | 1416 | 232 | |
| | 3000 | 540 | |
| | 1985 | 300 | |
| 20% | 1534 | 315 | cross validation (CV) set |
| | 1427 | 199 | |
| 20% | 1380 | 212 | test set |
| | 1494 | 243 | |

$(x^{(1)}, y^{(1)})$

⋮

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

$(x^{(1)}_{cv}, y^{(1)}_{cv})$

⋮

$(x^{(m_{cv})}_{cv}, y^{(m_{cv})}_{cv})$

$m_{cv}$: no. of cv examples

$(x^{(1)}_{test}, y^{(1)}_{test})$

⋮

$(x^{(m_{test})}_{test}, y^{(m_{test})}_{test})$

**Train/validation/test error**

Training error:

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

Cross Validation error:

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

Test error:

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

Instead of using the test set to select the model, we can use validation (or cv) set to select the model.

**Model selection**

1. $h_\theta(x) = \theta_0 + \theta_1 x$ → $\min_\theta J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ → $\theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$ → $\theta^{(3)}$

⋮

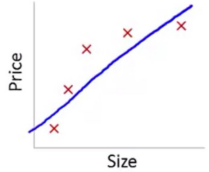
10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$ → $\theta^{(10)} \rightarrow J_{cv}(\theta^{(10)})$

say $J_{cv}(\theta^{(5)})$ is the lowest

↳ estimate generalisation error for test set $J_{test}(\theta^{(5)})$

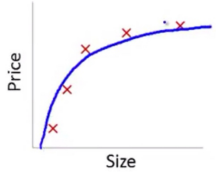
Diagnose: Bias vs Variance

**Bias/variance**



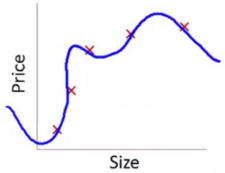

$\theta_0 + \theta_1 x$

High bias (underfit) $d=1$




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

"Just right" $d=2$



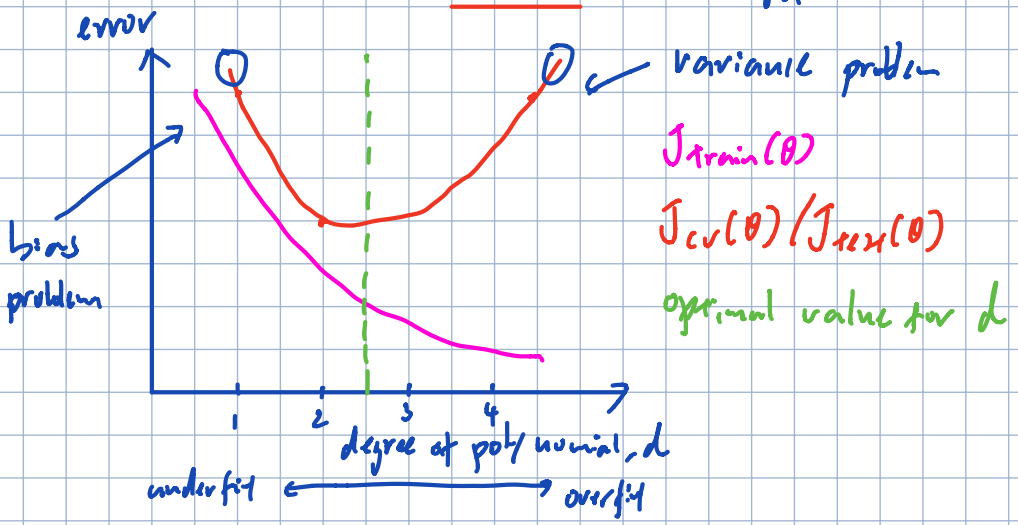

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance (overfit) $d=4$

# Bias/variance

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

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




variance problem

$J_{train}(\theta)$

$J_{cv}(\theta)$ / $J_{test}(\theta)$

optimal value for d

Suppose our learning algorithm is performing less well than we were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high). Is it bias or 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

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

## Regularisation and bias/variance

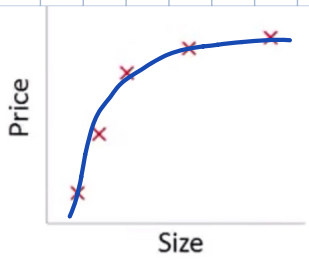
Suppose we're fitting a high order polynomial, for linear regression with regularisation:

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

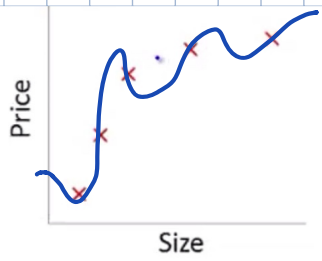
$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

Let's consider 3 cases:


Large $\lambda$
High bias (underfit)
$\lambda = 10000.\ \theta_1 \approx 0, \theta_2 \approx 0, \ldots$
$h_\theta(x) \approx \theta_0$




Intermediate $\lambda$
"Just right"




Small $\lambda \approx 0$
High variance (overfit)

If we define:

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

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

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

(w/o regularisation)

1. try $\lambda = 0$ → $\min_\theta J(\theta)$ → $\theta^{(1)}$ → $J_{cv}(\theta^{(1)})$
2. try $\lambda = 0.01$
3. try $\lambda = 0.02$
4. try $\lambda = 0.04$
5. try $\lambda = 0.08$
⋮
12. try $\lambda = 10$

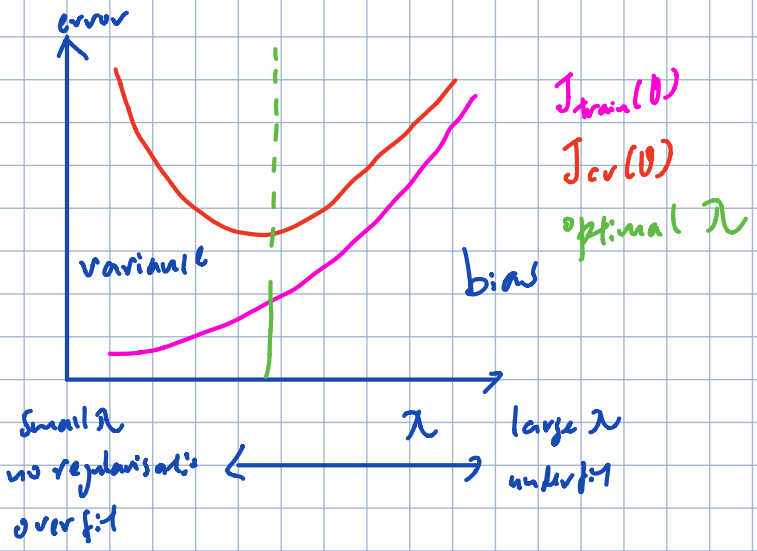
$\theta^{(12)}$ → $J_{cv}(\theta^{(12)})$

Say $\theta^{(5)}$ has the lowest $J_{cv}(\theta^{(5)})$, we can pick that

↳ compute $J_{test}(\theta^{(5)})$ to check generalisation error.

Bias / variance as a function of the regularisation parameter $\lambda$

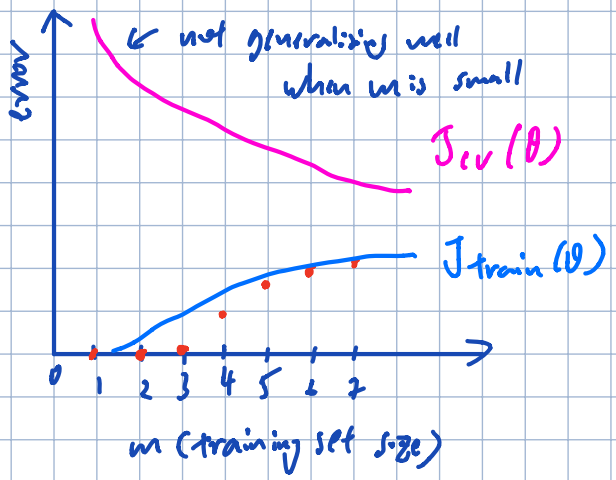
$J_{train}(\theta)$ / $J_{cv}(\theta)$ has no regularisation term for this section

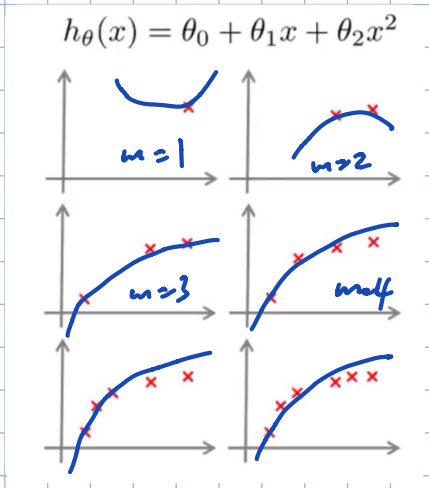



## Learning Curves

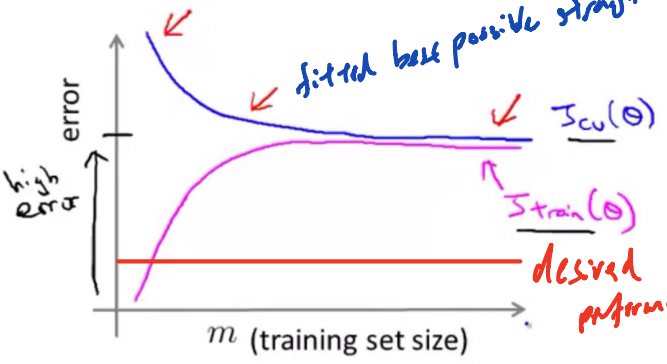
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

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

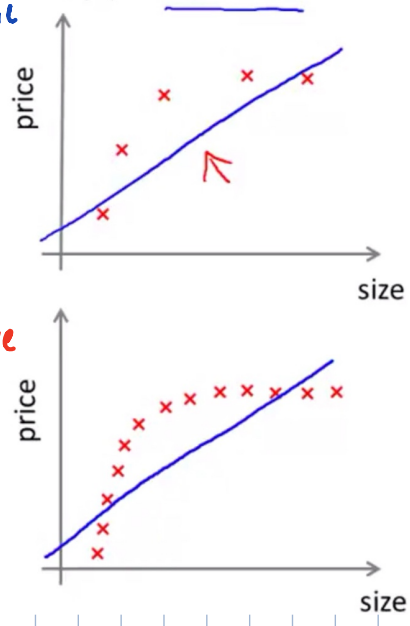


## High bias



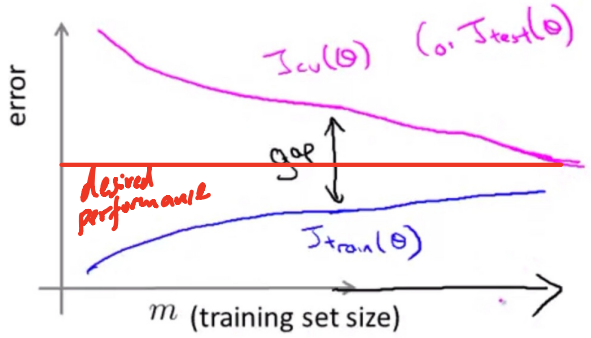

$h_\theta(x) = \theta_0 + \theta_1 x$ ← high bias model

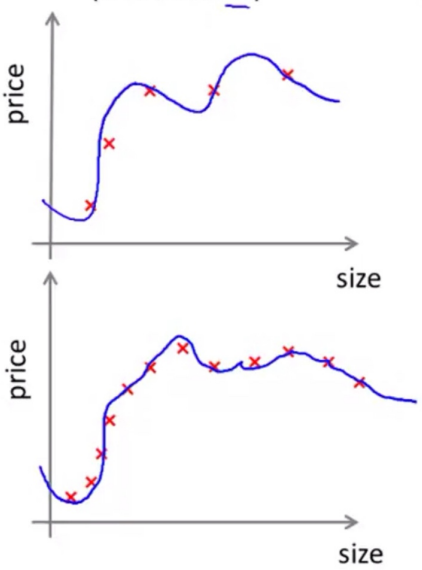



If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

## High variance

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




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

**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?

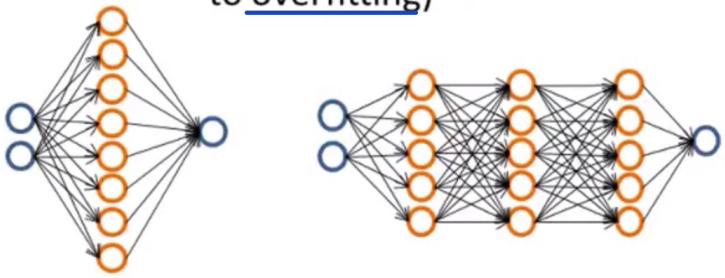
- Get more training examples → fix high variance
- Try smaller sets of features → fix high variance
- Try getting additional features → fix high bias, current model too simple
- Try adding polynomial features $(x_1^2, x_2^2, x_1 x_2, \text{etc})$ → fix high bias
- Try decreasing $\lambda$ → fix high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

## Neural networks and overfitting

"Small" neural network (fewer parameters; more prone to underfitting)

Computationally cheaper

"Large" neural network (more parameters; more prone to overfitting)



Computationally more expensive.

Use regularization ($\lambda$) to address overfitting.

## Building a spam classifier

Supervised learning.

$x$ = features of email  $y$ = Spam (1) or ham (0)

Feature $x$: Choose 100 words indicative of spam/ham.

e.g. deal, buy, discount...

Given an email, we can construct a feature vector

$$x = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \text{buy} \\ \text{deal} \\ \vdots \\ \text{now} \\ \vdots \end{matrix} \qquad x_j = \begin{cases} 1 & \text{if word } j \text{ appears in email} \\ 0 & \text{otherwise} \end{cases}$$

$x \in \mathbb{R}^{100}$

In practice, take most frequently occuring n words (10k - 50k) in training set, rather than picking 100 words manually.

## Building a spam classifier

How to spend our time to make it have low error

→ collect lots of data

→ develop sophisticated features based on email routing information (from email header)

→ develop sophisticated feature for message body

→ detect misspelling

## Recommended approach

→ Start with a simple algorithm that we can implement quickly
Implement it and test it on our cross-validation data

→ Plot learning curves to decide if more data, more features, etc are likly to help

→ error analysis: Manually examine the examples (in cross validation set) that our algorithm made errors on. See if we spot any systematic trend in what type of examples it is making errors on.

e.g. $m_{cv} = 500$ examples in cv set

Algorithm misclassifies 100 emails

↳ manually examine 100 errors, and categorise them based on:

(i) what type of email it is

(ii) what cues (features) we think would have helped the algorithm classify them correctly.

# Error metrics for skewed classes

Cancer classification example:

train logistic regression model $h_\theta(x)$

$y = \begin{cases} 1 & \text{cancer} \\ 0 & \text{otherwise} \end{cases}$

find that you got 1% error on test set, 99% correct diagnoses

⇒ only 0.5% of patients have cancer.

⇒ then error percentage is big.

Precision / Recall

✡ $y=1$ in presence of rare class that we want to detect

| predicted class | actual class: 1 | actual class: 0 |
|---|---|---|
| 1 | TP | FP |
| 0 | FN | TN |

**Precision**

of all patients where we predict $y=1$, what fraction actually has cancer?

$$\frac{TP}{\#\text{predicted positive}} = \frac{TP}{TP + FP}$$

**Recall**

of all patients that actually have cancer, what fraction did we correctly detect as having cancer?

$$\frac{TP}{\#\text{actual positive}} = \frac{TP}{TP + FN}$$

want high precision
high recall

# Trading off precision and recall

logistic regression: $0 \le h_\theta(x) \le 1$

predict 1 if $h_\theta(x) \ge 0.5$ → 0.7

predict 0 if $h_\theta(x) < 0.5$ → 0.7

more confident → 0.9 → 0.7

less confident → 0.3

⇒ Suppose we want to predict $y=1$ (cancer) only if very confident

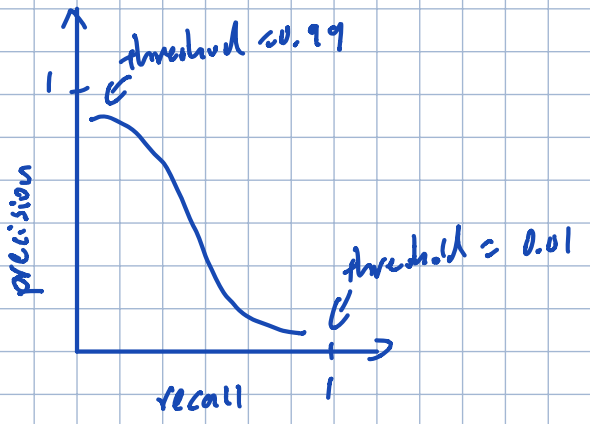
→ higher precision

→ lower recall (∵ predict $y=1$ less often)

⇒ Suppose we want to avoid missing too many cases of cancer (i.e. avoid false negative)

→ low precision

→ higher recall




# $F_1$ score ($F$ score)

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 |

A single metric to decide (instead of P and R)

average: $\frac{P+R}{2}$ ✗

Not very good, classifier which predicts $y=1$ all the time, will get very high recall. (algorithm 3)

classifier that predicts $y=0$ all the time, (by setting threshold very high), ⇒ very high precision and very low recall

$F_1$ Score: $2\frac{PR}{P+R}$ $P=0, R=0,$ F score $=0$

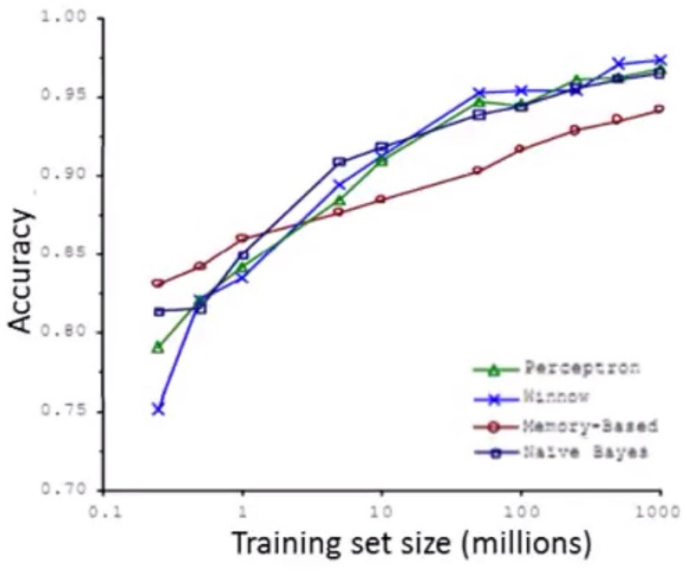
$P=1, R=1,$ F score $=1$

(good if both P and R are high, punishes if one of them are very low)

* try a range of threshold, compute $F_1$ score on the CV set, pick the threshold that gives the maximum $F_1$ score

✗ Assumes feature $x \in \mathbb{R}^{n+1}$ has enough info to predict y accurately

## Data for ML




## Large data rationale

use a learning algorithm with many parameters (e.g. logistic regression with many features; neural network with hidden units)

low bias algorithm — low bias

→ $J_{train}(\theta)$ will be small

use a very large training set (unlikely to overfit) — low variance

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

hopefully

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