Machine Learning

Topics:

1. Introduction
2. Linear Representation with One Variable
4. Linear Regression with Multiple Variables
5. Octave Tutorial
6. Logistic Regression
7. Regularization
8. Neural Network Representation
9. Neural Networks: Learning
10. Advice for Applying Machine Learning
11. Machine Learning System Design
12. Support Vector Machines
13. Clustering
14. Dimensionality Reduction
15. Anomaly Detection
16. Recommender Systems
17. Large Scale Machine Learning
18. Application Example: Photo OCR

Machine Learning

1. Introduction

Machine Learning:
- → grew out of AI
- → new capability for computers

Examples:
- → data mining
  - web click - better serve users
  - medical records - understand disease
  - biology - understanding DNA
  - engineering -
- → applications which can't be programmed by hand
  - autonomous helicopter
  - handwriting recognition
  - computer vision
- → self-customizing programs
  - recommendations
- → understand human learning (brain, real AI)

1.2 What is Machine Learning?

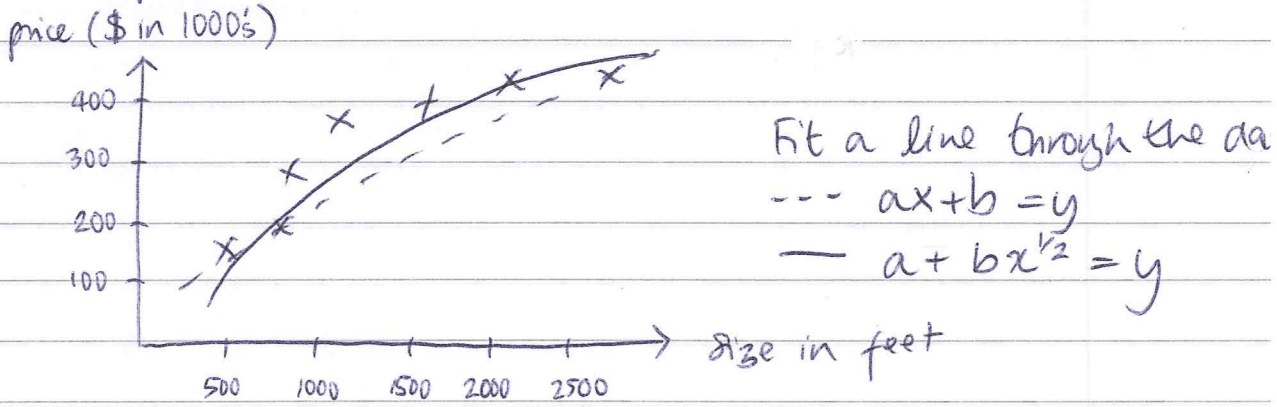
Definition: Arthur Samuel (1959) Checkers playing alg
Field of study that gives computers the ability to learn without being explicitly programmed

Definition: Tom Mitchell (1998) Rhyming definition
Well posed learning problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with E.

Machine learning algorithms
- → supervised learning: we teach the computer
- → unsupervised learning: computer learns by itself
- → Reinforcement learning
- → Recommender systems

1.3 Supervised Learning

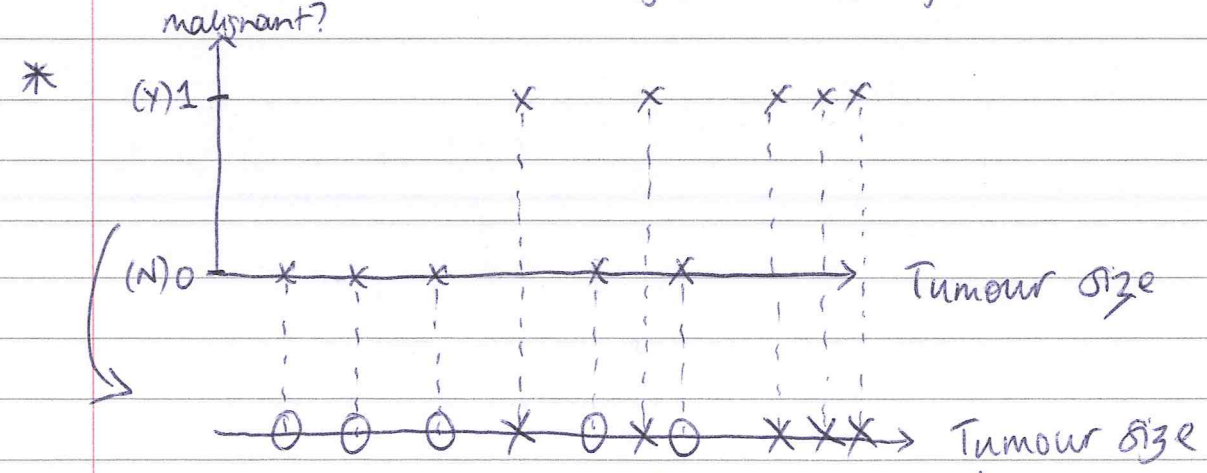
1.3.1 House price prediction



Supervised learning:
Right answers given

[Regression:
Predict continuous valued output (price)]

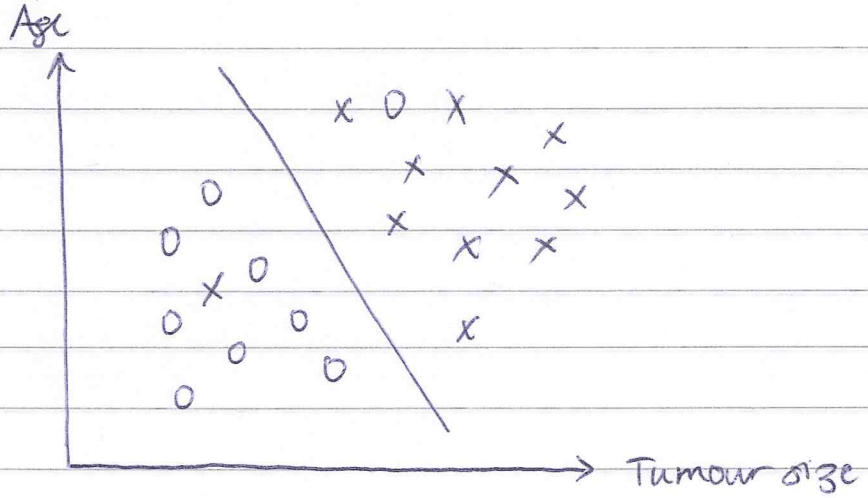
1.3.2 Breast cancer (malignant, benign)



- here we have one feature as input...

[Classification:
Discrete valued output (0 = N or 1 = Y)]

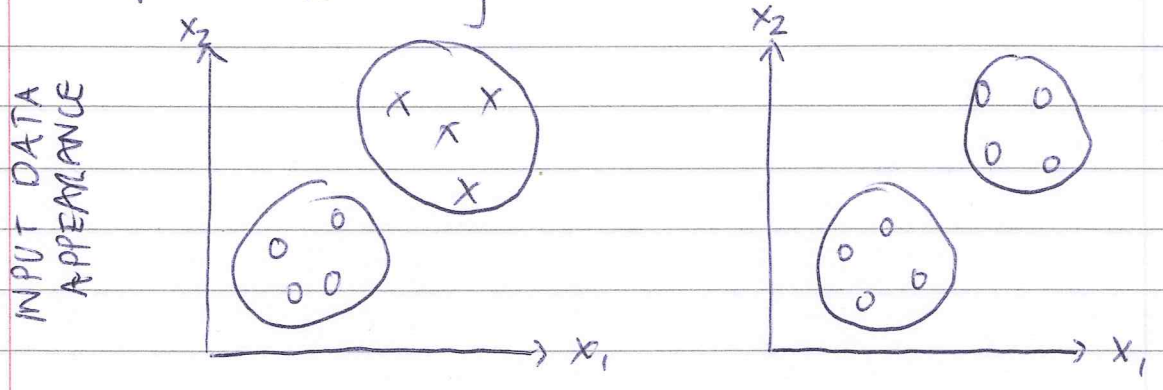
*



- here two features as input

other possible features
→ clump thickness
→ uniformity of cell size
→ uniformity of cell shape

1.4 Unsupervised Learning



supervised learning unsupervised learning

input data is provided and the algorithm looks for structure in it

Clustering algorithms used in
- → Google news for grouping stories on the web
- → Gene data -
- → Organise computer clusters
- → Social network analysis
- → Market segmentation
- → Astronomical data analysis

Cocktail party problem
- two speakers, two microphones - the problem is to separate the sounds from the two people out

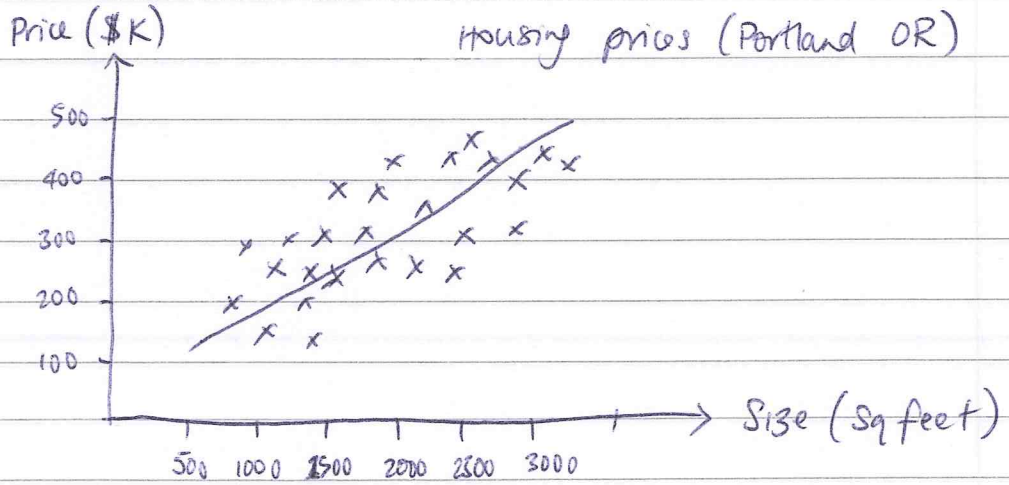
Solution:

```
[W,S,V] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x
```

octave open source free language - fast good for prototy
svd = singular value decomposition

2. Linear Regression with one variable

2.1 Model Representation



Supervised learning:
Given the "right answer" for each example in the data

Regression problem:
Predict real valued output ass oppose to a classification problem with discrete valued out

Training set of housing prices (Portland, OR)

Size in feet (x)	Price ($K) (y)
2104	460
1416	232
1534	315
852	178
...	...

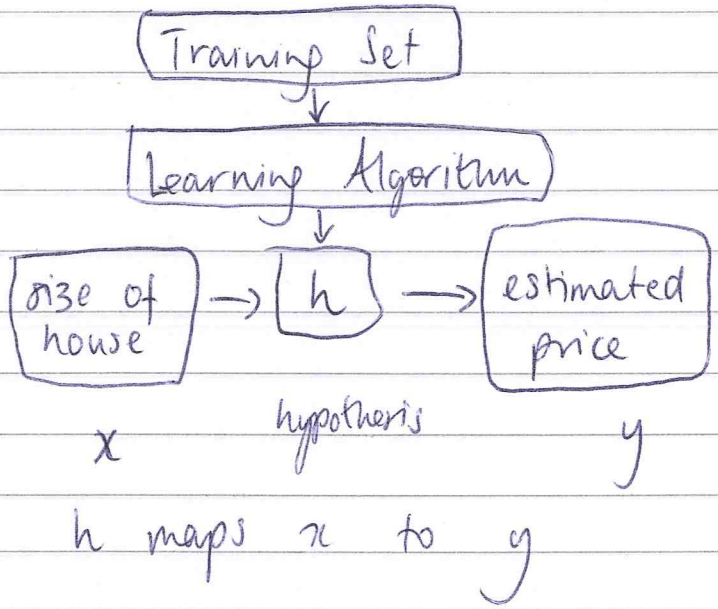
Notation:

m = # training examples

x's = input variable / features

y's = output variable / target variable

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

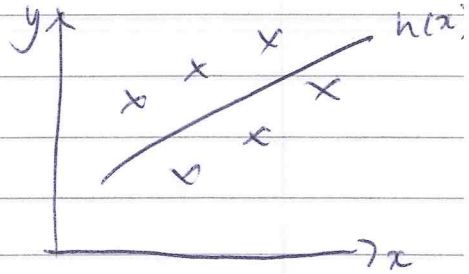


h maps x to y

how do we represent

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

shorthand $h(x)$



This is called

- Linear regression with one variable or
- Univariate linear regression

2.2 Cost function (Intuition I and II)

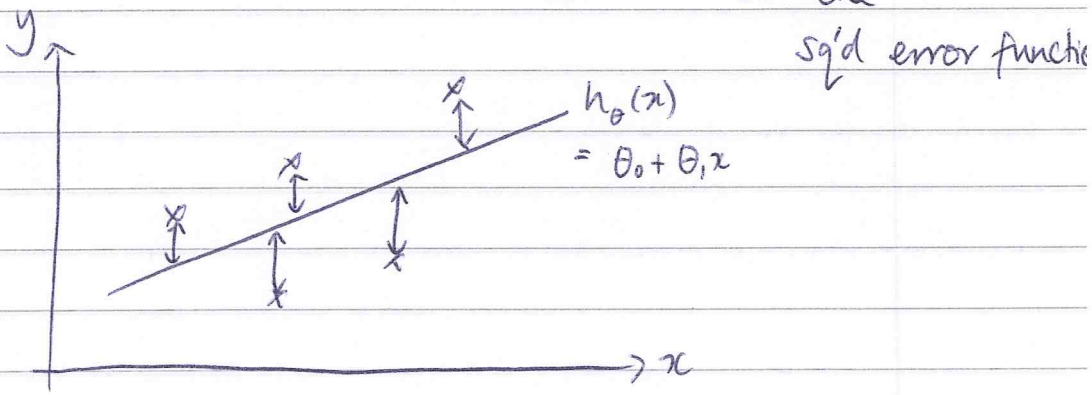
Different choices of θ_0 and θ_1 parameters give different hypothesis functions

We choose θ_0 and θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y). To do

this we solve a minimisation problem i.e. minimise

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

cost function $J(\theta_0, \theta_1)$ or the sq'd error functio



2.3 Gradient descent

The idea is that if you plot your cost functions in terms of its parameters you work your way (downhill) to a minimum as quickly as possible

Gradient descent algorithm:

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

α = learning rate = step size downhill

for $j = 0, 1$ simultaneously update θ_0 and θ_1

Correct:

$$\text{temp}0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{temp}1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp}0$$

$$\theta_1 := \text{temp}1$$

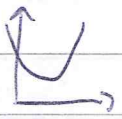
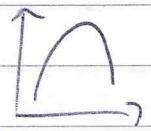
Incorrect: will probably work but is not gradient decent

$$\text{temp}0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp}0$$

$$\text{temp}1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_1 := \text{temp}1$$

Recall: convex  concave 

4 Linear Regression with Multiple Variables

4.1 Multiple features

Denote $x_j^{(i)}$ value of feature j in the ith training example

So now we have

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

for convenience, define $x_0 = 1$

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

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

This is called Multivariate Linear Regression

4.2 Gradient decent for multiple variables

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 $\{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \}$ simultaneously update for every $j = 0, \ldots, n$

Recall that for one variable we had

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

simultaneously update θ_0 and θ_1

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

Now we have

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

simultaneously update θ_j for $j = 0, 1, \ldots, n$

4.3 Gradient descent in practice I : feature scaling

If the various features are orders of magnitude different, it's worth scaling them to similar sizes. This leads to faster convergence.

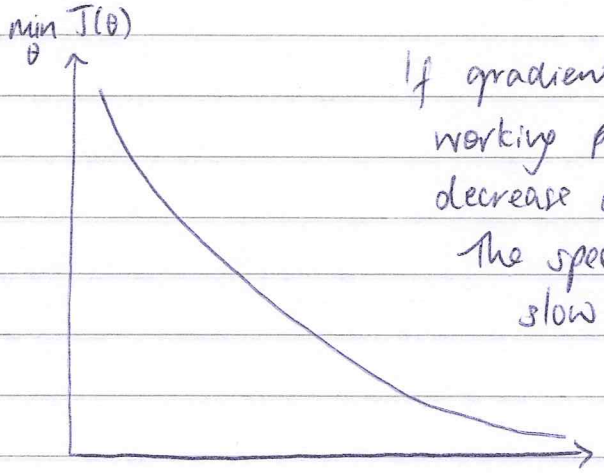
Mean normalisation:

Replace x_i with $(x_i - \mu_i)/\sigma_i$ to make features have approximately zero mean and a reasonable range.

μ_i = mean
σ_i = std dev / range } ith feature

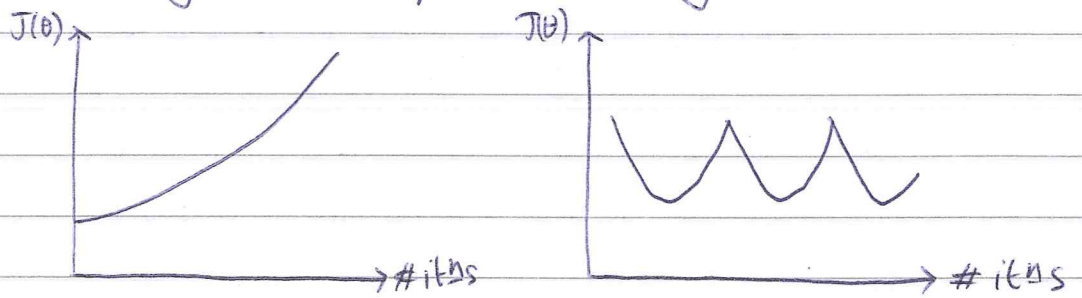
4.4 Gradient descent in practice II : Learning Rate

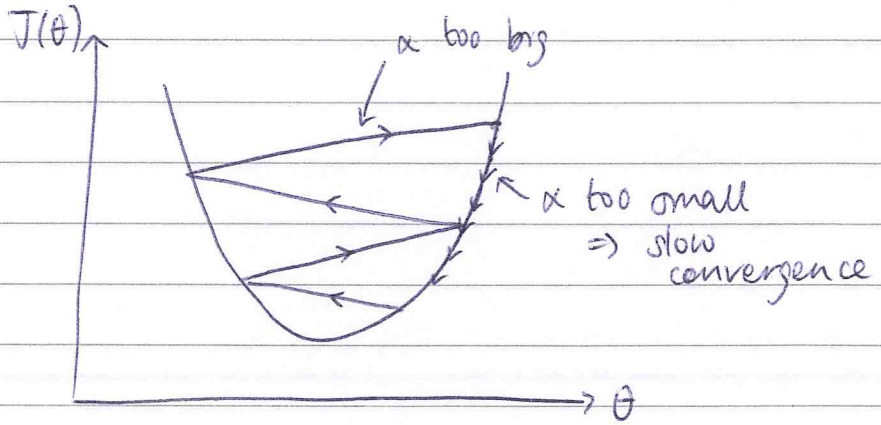
Making sure gradient descent is working properly.



If gradient descent is working properly $J(\theta)$ will decrease after every iteration. The speed of decay also slow as you converge

If $J(\theta)$ increases then it is likely that you are using too large a learning rate.





4.5 Features & Polynomial Regression

The linear dependence need not be assumed on the features you could for example optimise over

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

or

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

you can fit any polynomial functions

for example. When using these types of functions feature scaling becomes even more important.

4.6 Normal Equation

This gives us an analytic formula for the optimal θ

Reall: In 1D $J(\theta)$ is just a quadratic in θ. We can find the minimum analytically. Clearly this will also work for n features, we just have to solve n+1 quadratic equations simultaneously i.e.

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

set $\frac{\partial}{\partial \theta_j} J(\theta) = 0 \quad \forall j$ (i.e. $j = 0, 1, \ldots, n$)

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

x_0	size (ft^2)	# bedrooms	# floors	age	Price
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

X $\qquad$ y

$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ 1 & \vdots & \vdots & & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{pmatrix} \qquad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} \qquad y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

$X \in \mathbb{R}^{m, n+1}$ $\quad x_0^{(j)} = 1 \ \forall j$ $\qquad \theta \in \mathbb{R}^{n+1}$ $\qquad y \in \mathbb{R}^m$

Want θ such that $\min_\theta \left[(X\theta - y)^T (X\theta - y) \right]$

Normal equation:

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

In Octave this can be written as

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

m training examples and n features

Gradient Descent	Normal Equation
• Need to choose α	• No need to choose α
• Needs many iterations	• No need to iterate
• Works well even when n is large	• Need to compute $(X^T X)^{-1}$ $O(n^3)$
	• Slow if n is very large

→ If $n > 10{,}000$ consider using gradient descent rather than

4.7 Normal Equation Non-invertability

Normal equation: $(X^T X)^{-1} X^T y$

- what if $X^T X$ is non-invertible? (singular / degenerate)
- Octave: $pinv(X' * X) * X' * y$
  ↑ pseudo inverse as oppose to the actual inverse function

Two common causes of $X^T X$ being singular
→ Redundant features (linearly dependent features)
e.g. x_1 = size in $feet^2$
x_2 = size in m^2
→ Too many features and not enough samples
e.g. $m \leq n$
Either delete some features or use regularisation
we'll come back to this

5 Octave Tutorial

5.1 Basic Operations

- Logical operations:
  equal → A == B
  not equal → A ~= B
  and → A && B
  or → A || B , xor(A,B)
  false = 0
  true = 1

- Command line: PS1('>> ');

- Semicolon supresses output

- apostrophes for strings
- % for comments
- disp(sprintf('2 decimals: %0.6f', a))

  format long 14 dp
  short 4dp
- A = [1 2; 3 4; 5 6]
- v = [1 2 3] v = [1; 2; 3]
- v = 1:0.1:2

  ⇓

  v = (1, 1.1, 1.2, ..., 2)
- ones(2,3) ⇒ $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
- 2 * ones(2,3) ⇒ $\begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix}$
- zeros(1,3) ⇒ (0 0 0)
- w = rand(1,3)
- w = randn(1,3)
- w = -6 + sqrt(10) * (randn(1,10000))
- hist(w) print the histogram
  hist(w, 50) print with 50 bins

- eye(4) ⇒ $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

- help eye
- help rand
- help help

5.2 Moving data around

- size(A) ⇒ [r, c] 1×2 matrix
- size(A,1) ⇒ number rows
  size(A,2) number columns
- length(V)
  length(A) gives the longest dimension
  i.e. max(r, c)
- pwd gives current directory
- cd 'C:\...' to change directory
- ls to list files
- load featuresX.dat or load('featuresX.dat')
  load priceY.dat
- who tells you which variables you have in memory
  whos gives name, size, bytes and class (double, ...)
  " double precision

- clear A
- v = priceY(1:10) gives v = first 10 elements of priceY
- save hello.mat v;
- clear just clears everything
- save hello.txt v -ascii % save as text (ASCII)
- A(3,2) => $A_{3,2}$
  A(2,:) => 2nd row of A
  A(:,2) => 2nd column of A

  A([1,3],:) 1st & 3rd row of A
  A(:,2) = [10; 11; 12] => sets 2nd column of A as specified
  A = [A, [100; 101; 102]] => appends the matrix with the column vector specified
  A(:) => puts all elements of A in a column vector
  C = [A B] } concatinates
  C = [A; B] } matricies

  n.b. [A B] = [A, B]

5.3 Computing on Data

- $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$ $B = \begin{pmatrix} 11 & 12 \\ 13 & 14 \\ 15 & 16 \end{pmatrix}$ $A.*B = \begin{pmatrix} 11 & 24 \\ 39 & 56 \\ 75 & 96 \end{pmatrix}$
- element wise operations e.g .*, ./, .+, etc..

- log(A), exp(A), abs(A)
  all element wise operations
- A' transpose A
- A+1 adds 1 to all elements
- max(A) column wise max default
  max(max(A)) for max of all elements
  max(v)
- v < 3 gives true false element wise

  find(a<3) returns the elements of a which are <3

sum(a) prod(a) sum and products
floor(a) ceil(a) rounding down and up

max(A, [], 1) max of each column
max(A, [], 2) max of each row

A=magic(9)
sum(A,1) sums each column
sum(A,2) " " row
A.*eye(9)
sum(sum(A.*eye(9)))
flipud(eye(9)) gives 1 on off diag

5.4 Plotting data

t=[0, 0.01, 1]; y1 = sin(2*pi*4*t)
plot(t, y1)

```
hold on;  % adds plots to the same graph
y2 = cos(2*pi*4*t)
plot(t,y2,'r')
title('my plot')
xlabel('time')
ylabel('value')
legend('sin','cos')
print -dpng 'myplot.png'
close  % closes the window with the plot in

figure(1); plot(t,y1)
figure(2); plot(t,y2)

subplot(1,2,1); plot(t,y1)
subplot(1,2,2); plot(t,y2)

axis([0.5 1 -1 1])  sets axis max and min

imagesc(A), colourbar, colourmap gray;
            % comma chaining function calls
            % semi colons do the same but with
              ans surpressed
```

5.5 Control statements : for, while & if statements and functions

```
v = zeros(10,1)
for i=1:10, v(i) = 2^i; end;

indices = 1:10
for i= indices, disp(i); end;

break, continue
```

```
i=1
while i<=5, v(i)=100; i=i+1; end;
```

```
i=1;
while true,
   v(i)=999;
   i=i+1;
   if i==6,
      break;
   end;
end;
```

```
v(1)=2
if v(1)==1
   disp('one');
elseif v(1)==2
   disp('two');
else
   disp('not one/two');
end;
```

- Save functions in .m file
  open these in WordPad not notepad

```
function y = squareThisNumber(x)
y=x^2;
```

of course the file has to be in your working directory or add the place to your search paths

```
addpath('C:\Users\...')
```

```
function [y1,y2] = squareAndCubeThisNumber(x)
y1=x^2;
y2=x^3;
```

```
function J = costFunctionJ(X, y, theta)

% X = "design matrix" containing our training examples
% y is the class labels

m = size(X,1)
predictions = X*theta';

sqrErrors = (predictions - y).^2;

J = 1/(2*m)*sum(sqrErrors);
```

5.6 Vectorization

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j = \theta^T x \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \quad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

- Unvectorized implementation:

```
prediction = 0.0;
for j = 1:n+1,
    prediction = prediction + theta(j) * x(j)
end;
```

- Vectorized implementation

```
prediction = theta' * x;
```

6. Logistic Regression

6.1 Classification

→ Email : spam / not spam
→ Online transactions : fraudulent / not fraudulent
→ Tumour : Malignant / Benign

$y \in \{0, 1\}$ 0 : +ve class
1 : -ve class

Linear regression not suitable for classification problems

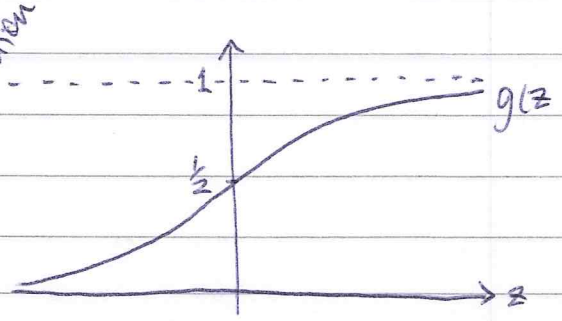
Logistic regression : $0 \leq h_\theta(x) \leq 1$

6.2 Hypothesis representation

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

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

Logistic function or Sigmoid function

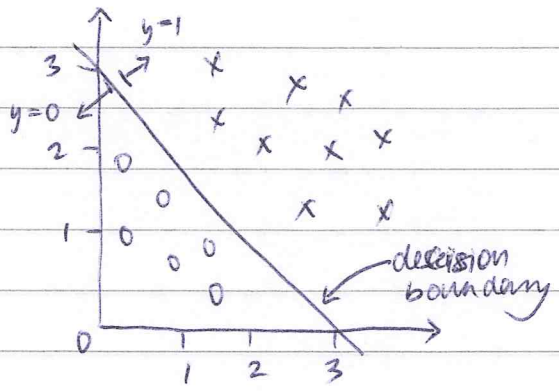


Interpretation:
$h_\theta(x)$ = estimated probability that y
$= P(y = 1 \mid x; \theta)$

6.3 Decision Boundary

Suppose we predict "$y = 1$" if $h_\theta(x) \geq 0.5$
i.e. $z = \theta^T x \geq 0$

⇒ "$y = 0$ if $h_\theta(x) < 0.5$
i.e. $z = \theta^T x < 0$



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

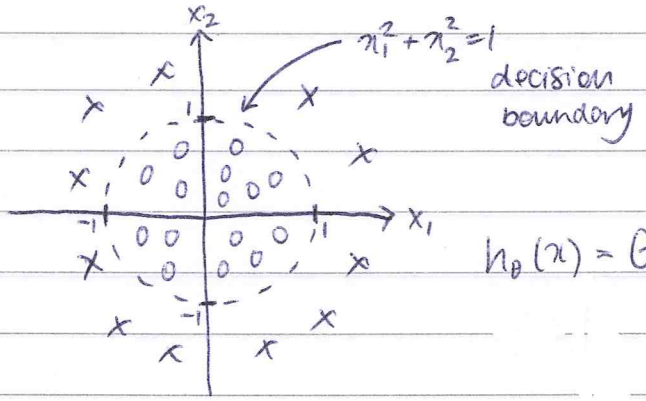
$\theta = \begin{pmatrix} -3 \\ 1 \\ 1 \end{pmatrix}$

predict 'y=1' if $-3 + x_1 + x_2 \geq 0$

i.e. $x_1 + x_2 \geq 3$

i.e.

Non-linear decision boundaries



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

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

6.4 Cost function

Recall:

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

$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^{n+1}$ $x_0 = 1$ $y \in \{0, 1\}$

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

How do we choose our parameters θ?

In linear regression ...

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

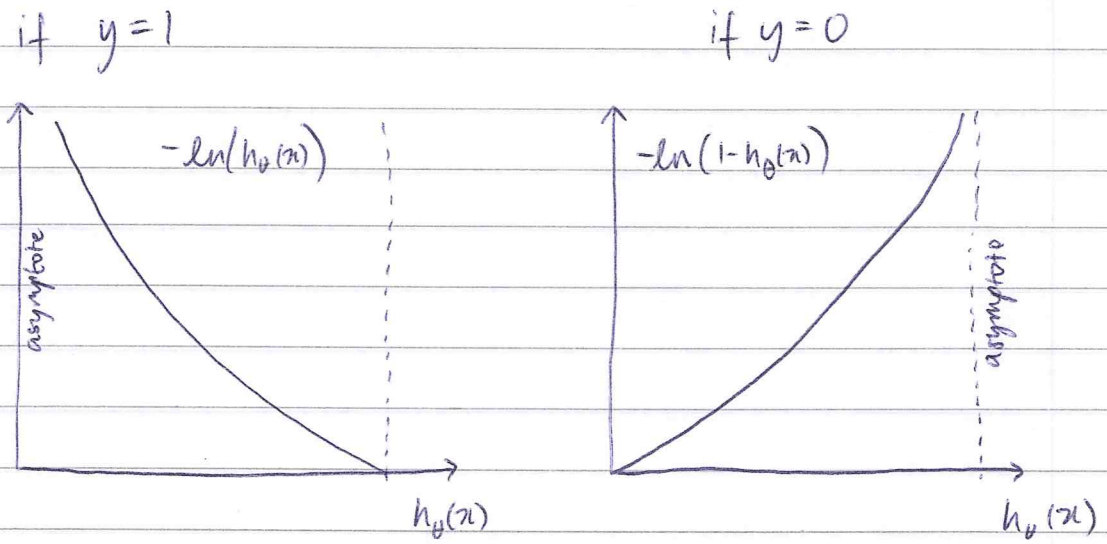
This cost function is non-convex for

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

so this is not suitable.

Logistic regression cost function

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



6.5 Simplified cost function & gradient descent

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

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

rewrite:

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

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

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

To fit parameters θ: $\min_\theta J(\theta)$

Use gradient descent i.e.

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

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

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

simultaneously update all θ_j

6.6 Advanced Optimisation

Optimisation algorithm

→ cost function $J(\theta)$. Want $\min_\theta J(\theta)$.

→ given θ, we have code that can compute

- $J(\theta)$ ← to monitor convergence
- $\frac{\partial J(\theta)}{\partial \theta_j}$ for $j = 0, 1, \ldots, n$

Optimization algorithms:

→ Gradient descent

→ Conjugate gradient

→ BFGS

→ L-BFGS

Advantages

+ no need to choose α

→ faster than gradient descent

Disadvantages

→ more complex

Example:

$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ $\min J(\theta)$

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

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

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

```
function [jVal, gradient] = costFunction(theta)
  jVal = (theta(1)-5)^2 + (theta(2)-5)^2;
  gradient = zeros(2,1);
  gradient(1) = 2*(theta(1)-5);
  gradient(2) = 2*(theta(2)-5);
```

```
options = optimset('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag] ...
    = fminunc(@costFunction, initialTheta, options);
```

fminunc ← unconstrained minimisation; @ ← pointer

"Gradient descent on steroids"!

$\theta \in \mathbb{R}^d$ $d \geq 2$ for this to work

$theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}$ remember octave uses 1 indexing

```
function [jVal, gradient] = costFunction(theta)
  [code to compute...
  jVal = 
  gradient(1) = d/dθ0 (J(θ)) ]
  gradient(2) = d/dθ1 (J(θ)) ]
  ⋮                ⋮
  gradient(n+1) = d/dθn (J(θ)) ]
```

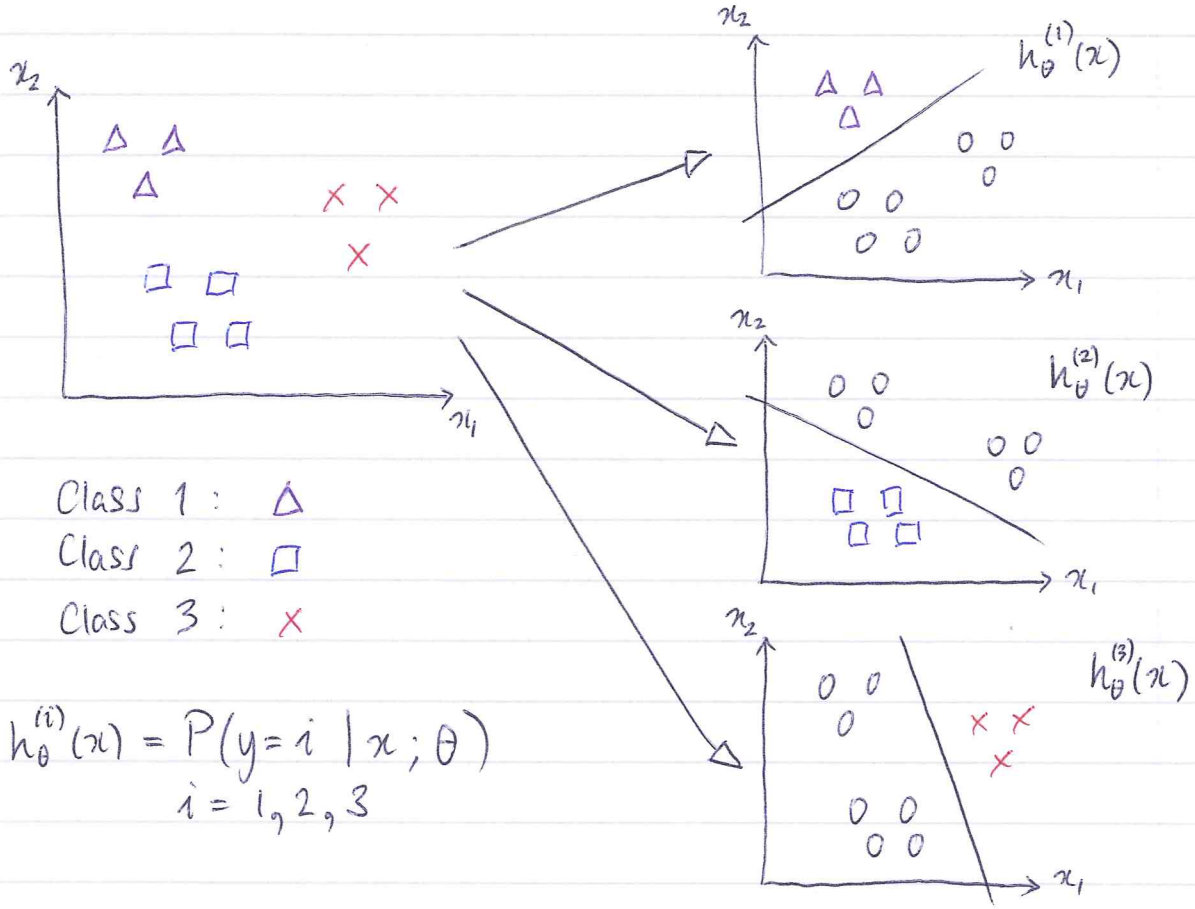
6.7 Multiclass Classification: One-vs-all

Multiclass classification

- → Email folder / tagging: Work / friends / family / hobby
- → Medical diagrams: Not ill / cold / flu
- → Weather: Sunny / cloudy / rain / snow

here $y = 0, 1, 2, \cdots, n$ ← n = no. of classes

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



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

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

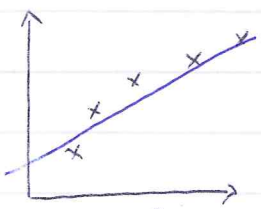
On a new input x, to make a prediction, pick the class i that maximises

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

7 Regularization

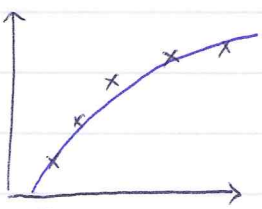
7.1 The problem of over-fitting

Example: Linear Regression



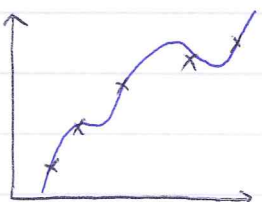
$\theta_0 + \theta_1 x$

"underfit" / "high bias"



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

"just right"



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

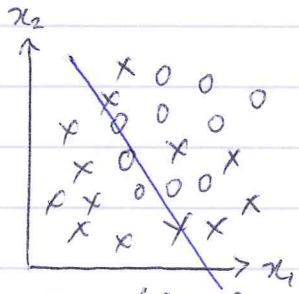
"overfit" / "high variance"

Example: Logistic Regression

Overfitting: If we have too many features the learned hypothesis may fit the training set very well i.e.

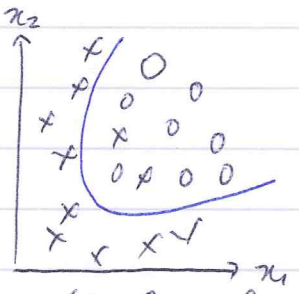
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0 ,$$

but fail to generalise to new examples (predict prices on new examples).



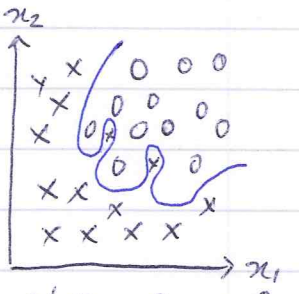
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
g = sigmoid function

"underfit"



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

"just right"



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

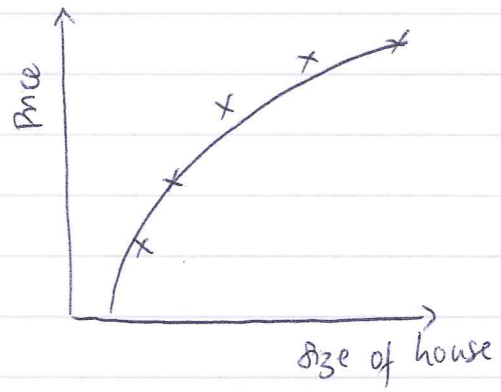
"overfit"

Addressing overfitting

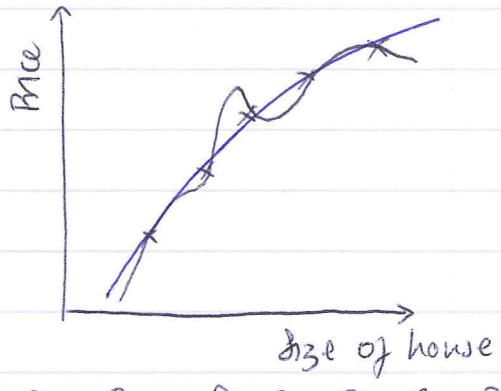
- Common when you have many features and few training examples. Here you need to lose some features
  - → manually select which features to keep
  - → model selection algorithm to choose
- Regularisation
  - → keep all the features, but reduce magnitude / values of parameters θ_j
  - → works well when we have lots of features, each of which contributes a bit to predicting y.

7.2 Cost function

Intuition



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



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

Suppose we penalize and force θ_3 and θ_4 to be small

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

→ the minimisation process will force $\theta_3 \approx 0$ and $\theta_4 \approx 0$

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

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

regularization parameter

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

we don't usually regularise θ_0 although in practice it makes little difference if we do.

- too large a λ choice can result in underfitting
- ∃ algorithms which choose λ well for you

7.3 Regularized Linear Regression

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

Gradient descent

Repeat $\{$

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

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

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

$\}$

rewrite,

$$\theta_j := \theta_j\underbrace{\left(1 - \frac{\alpha\lambda}{m}\right)}_{<1 \text{ usually only just e.g. } 1-\frac{\alpha\lambda}{m} \approx 0.99} - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

Normal Equation

$$X = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & & \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{pmatrix} \qquad y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

$\min_\theta J(\theta)$ $\qquad \frac{\partial J(\theta)}{\partial \theta_j} = 0$ gives min

$$\theta = \underbrace{\left(X^TX + \lambda\begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}\right)^{-1}}_{\text{Non-invertability (this happens when } m \le n)} X^Ty$$

Non-invertability (this happens when $m \le n$) - it turns out that as long a $\lambda > 0$ we no longer have this problem i.e. you can prove $X^TX + \lambda\begin{bmatrix} 0 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$ is invertible

7.4 Regularized Logistic Regression

Here we have

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

Repeat {

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

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

}

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

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

function [jVal, gradient] = costFunction(theta)

[code to compute ...

jVal = ... $J(\theta)$ = as above

gradient (1) = ... $\frac{\partial}{\partial\theta_0}(J(\theta)) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_0^{(i)}$

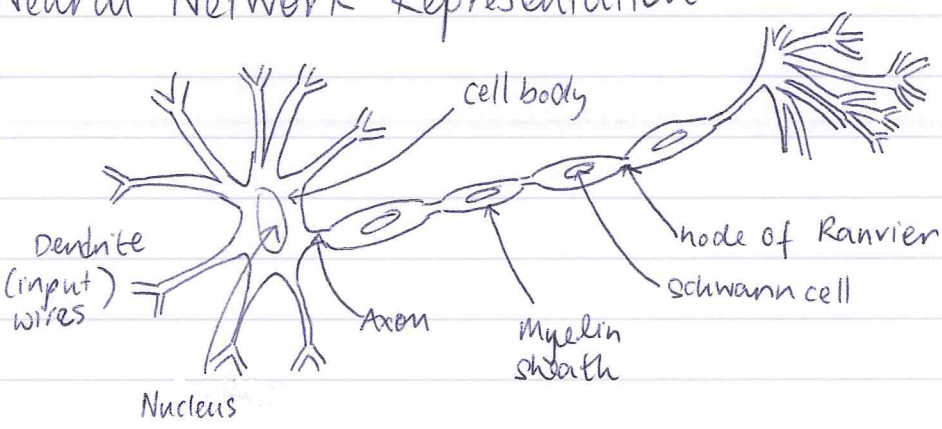
gradient (2) = ... $\frac{\partial}{\partial\theta_1}(J(\theta))$

⋮ ⋮

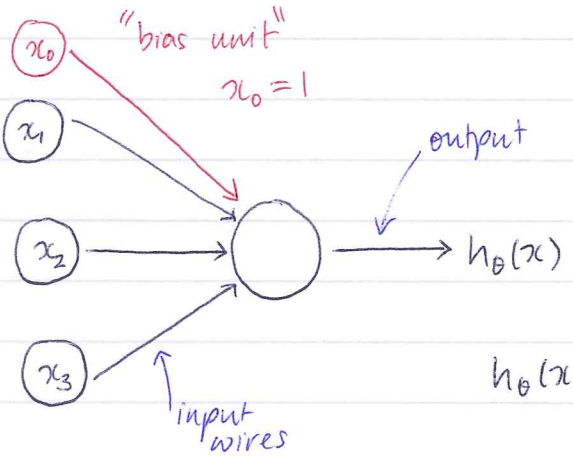
gradient (n+1) = ... $\frac{\partial}{\partial\theta_n}(J(\theta))$

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

8 Neural Network Representation



Neuron model : Logistic unit



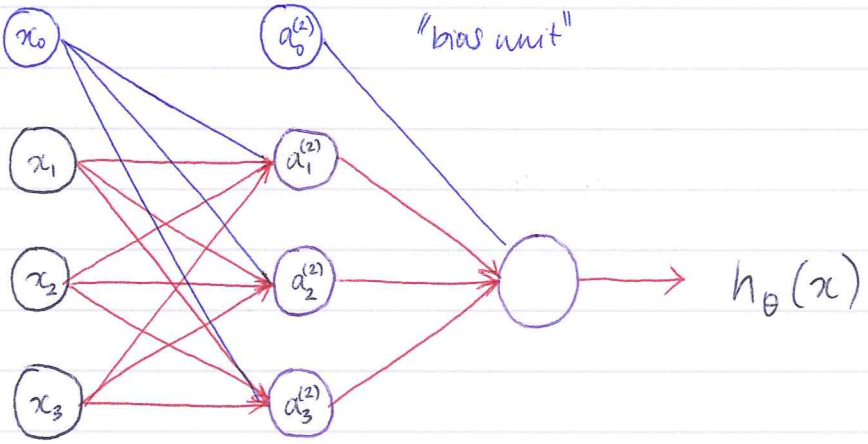
$$x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}$$

weights or parameters

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

This is a neuron with a Sigmoid / Logistic activation function

Neutral Network



Layer 1 (input layer) X

Layer 2 (hidden layer)

Layer 3 (output layer) y

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

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

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) = g(z_1^{(2)})$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) = g(z_2^{(2)})$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) = g(z_3^{(3)})$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}$$

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

Forward propagation: Vectorised implementation

$$\begin{pmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{pmatrix}$$

$$\Rightarrow \Theta^{(1)} x = z^{(2)} \quad \text{and}$$

$$a^{(2)} = \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{pmatrix} = g(z^{(2)})$$

Add $a_0^{(2)}$ corresponding to our bias unit

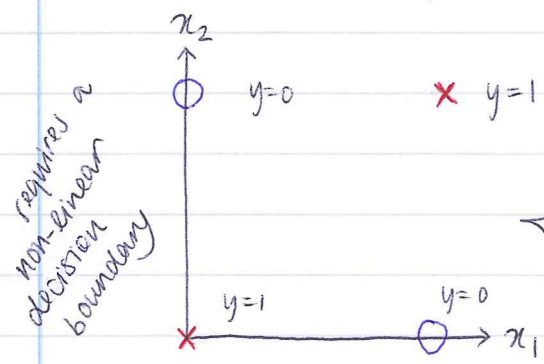
$$a^{(2)} = \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{pmatrix}$$

x can be thought of a $a^{(1)}$

$$\Theta^{(2)} a^{(2)} = z^{(3)} \quad \text{and} \quad h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

In neural networks we use logistic regression recursively to learn its own features to feed in ($a^{(2)}$) rather than ($x = a^{(1)}$). The network can have many layers. There is only one input layer and one output layer, all the others are hidden layers.

Non-linear classification example : XOR / XNOR



x_1, x_2 are binary

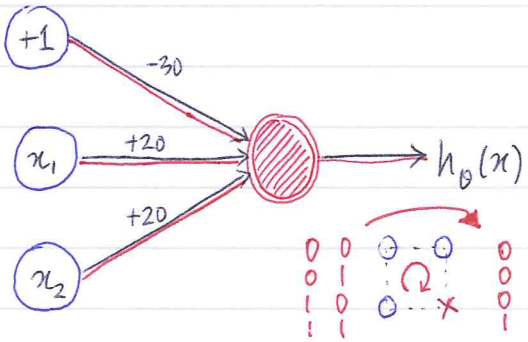
$y = x_1$ XOR x_2 } true if exactly one of x_1, x_2 are true

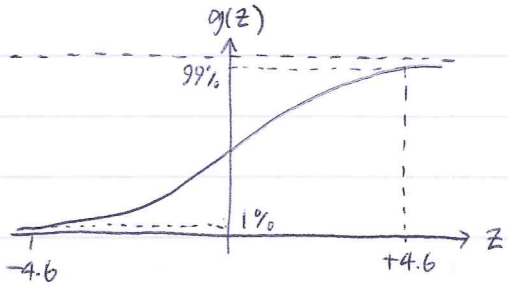
x_1 XNOR x_2
$= \text{NOT}(x_1 \text{ XOR } x_2)$ } true if both or neither x_1, x_2 are true

Get a neural network to fit this training set

Simple example : AND

$\{x_1, x_2\} \in \{0, 1\}$ $\quad y = x_1$ AND x_2



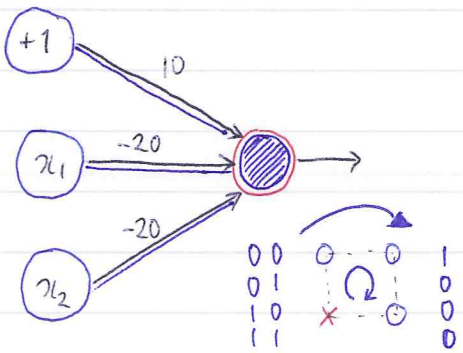


$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$

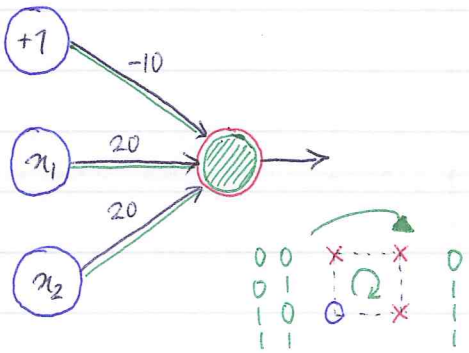
$\theta_{10}^{(1)}$ $\theta_{11}^{(1)}$ $\theta_{12}^{(1)}$

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	$g(10) \approx 1$

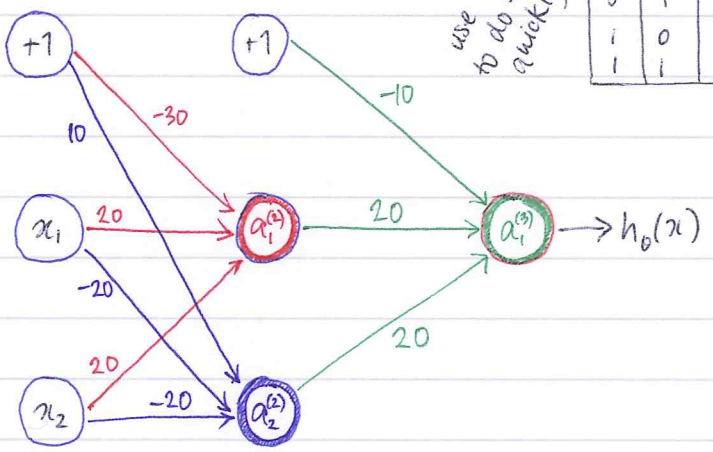
$h_\theta(x) \approx x_1$ AND x_2



(NOT x_1) AND (NOT x_2)



x_1 OR x_2



use this to do these quickly

x_1	x_2	$\theta^T x$
0	0	a
0	1	a+c
1	0	a+b
1	1	a+b+c

$\theta = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$

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

so we see how we can use neural networks as simple decision boundaries to build more complex non-linear ones!

Multiclass Classification

Multiple output unit: One vs All

If we want to classify data as one of k classes we need to have k elements in our output layer - one for each class. In this case we should predict

$h_\theta(x) = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ when the example is in our first class

$h_\theta(x) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ when the example is in our second class

and so on, i.e. $h_\theta(x) \in \mathbb{R}^k$

As before we have our training set

$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

but now $y^{(i)} \in \left\{ \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \right\}$

9 Neural Networks : Learning

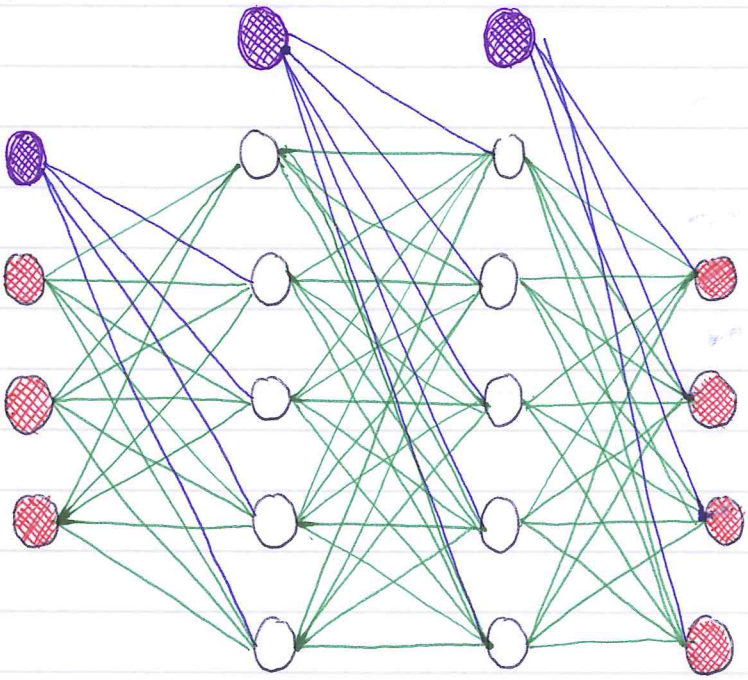
Neural Network (Classification)

$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

L = total number of layers in network (inc. input and output layers).

s_l = number of units (not including the bias unit) in Layer L

Example:



$L = 4$

$s_1 = 3$

$s_2 = s_3 = 5$

$s_4 = 4$

Layer 1 Layer 2 Layer 3 Layer 4

Binary classification	Multi-class classification (K classes)
$s_L = 1$ $y = 0$ or 1	$y \in \mathbb{R}^K$ e.g. $\begin{pmatrix}1\\0\\0\\\vdots\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\\vdots\\0\end{pmatrix}, \dots, \begin{pmatrix}0\\0\\\vdots\\0\\1\end{pmatrix}$
$h_\theta(x) \in \mathbb{R}$ $K = 1$	$h_\theta(x) \in \mathbb{R}^K$ $K \geq 3$
	$s_L = K$

9.1 Cost function

Logistic regression:

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

Neural Network:

$$h_\theta(x) \in \mathbb{R}^K \qquad (h_\theta(x))_i = i^{th} \text{ element}$$

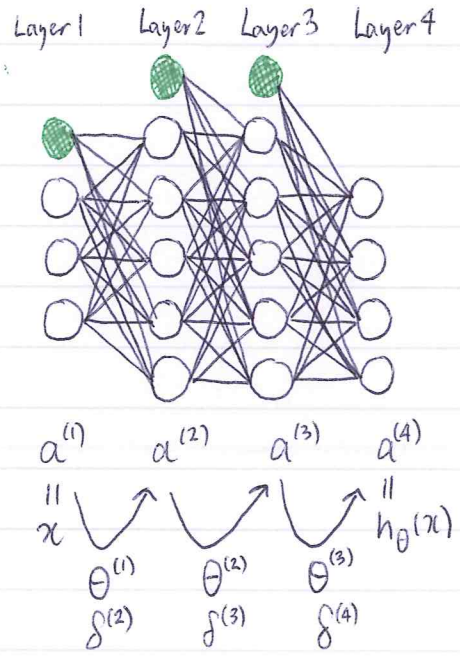
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \ln\left(h_\theta(x^{(i)})\right)_k + (1-y^{(i)}) \ln\left(1-(h_\theta(x^{(i)}))_k\right)\right]$$
$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}} \left(\Theta_{ji}^{(l)}\right)^2$$

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

9.2 Backpropogation Algorithm

Gradient computation
Given one training example (x, y): Forward propagation:

$a^{(1)} = x$
$z^{(2)} = \Theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)
$z^{(3)} = \Theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)
$z^{(4)} = \Theta^{(3)} a^{(3)}$
$a^{(4)} = h_\theta(x) = g(z^{(4)})$



Gradient computation: Backpropagation algorithm

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

For each output unit in layer $L=4$

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

i.e. $\delta^{(4)} = a^{(4)} - y$

$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$ $\qquad g'(z^{(3)}) = a^{(3)} .* (1 - a^{(3)})$

$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$ $\qquad g'(z^{(2)}) = a^{(2)} .* (1 - a^{(2)})$

(derivatives)

no $\delta^{(1)}$

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

(λ: regularization term)

Now assume we have a large training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0 \quad (\forall\, l, i, j)$

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform fwd prop to compute $a^{(l)}$ for $l = 2, 3, \dots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

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

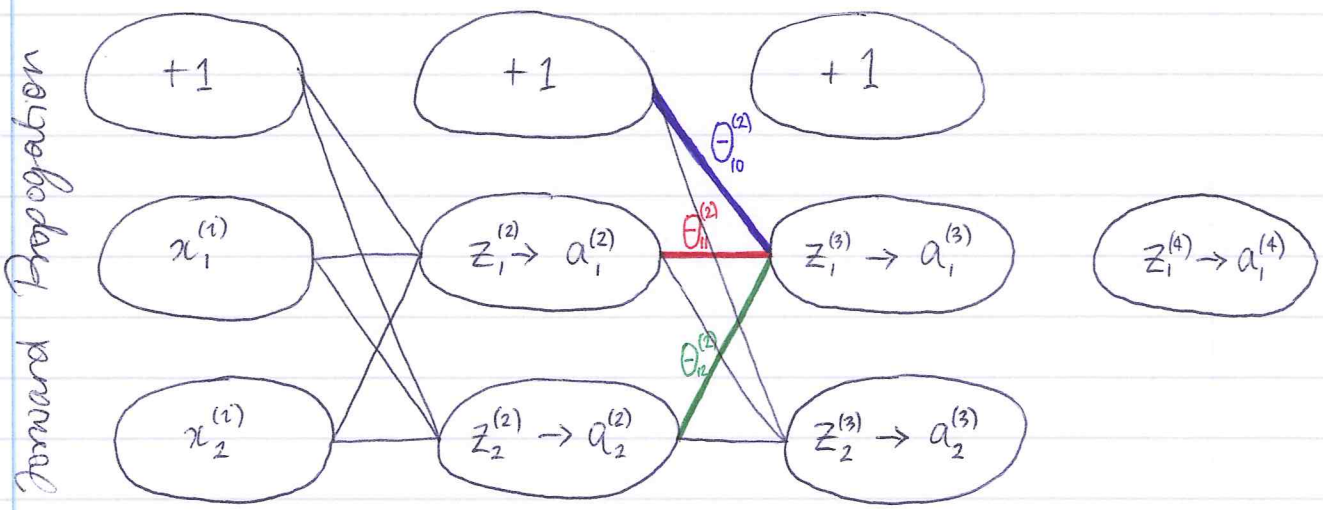
End For

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

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

Propogation Intuition

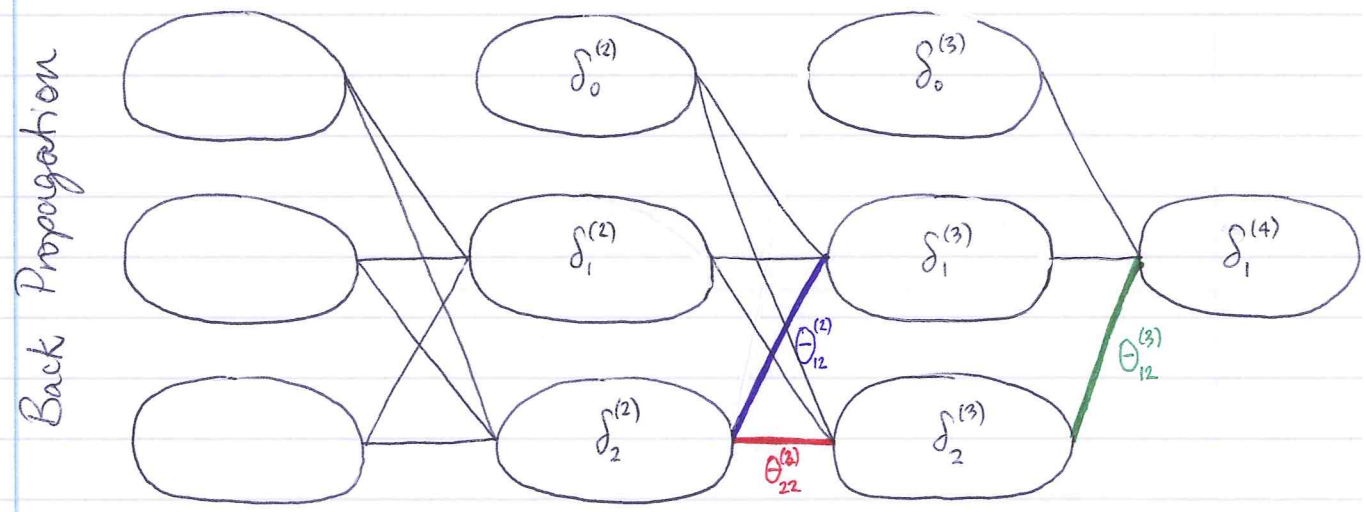
Forward Propogation



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

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

Back Propogation



$\delta_j^{(l)}$ = "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

$$\text{cost}(i) = y^{(i)} \ln\left(h_\theta(x^{(i)})\right) + (1 - y^{(i)}) \ln\left(1 - h_\theta(x^{(i)})\right)$$

$$\delta_1^{(4)} = y^{(i)} - a_1^{(4)} \qquad \delta_2^{(3)} = \Theta_{12}^{(3)} \delta_1^{(4)}$$

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

9.4 Implementation note: unrolling parameters

To we the advanced optimisation functions in Octave we must have our parameters $\Theta_{ij}^{(l)}$ and gradients $D_{ij}^{(l)}$ in vector (rather than matrix) form.

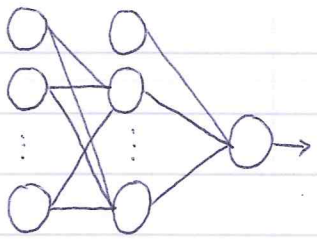
```
function [jVal, gradient] = costFunction (theta)
...
optTheta = fminunc (@costFunction, initialTheta, options)
```

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

Neural network $L=4$
- $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)
- $D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (D1, D2, D3)

"Unroll" into vectors

Example: $S_1 = 10 \quad S_2 = 10 \quad S_3 = 1$



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

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

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

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

Learning Algorithm
- Unroll initial param's $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ to get initialTheta to pass into fminunc (@costFunction, initialTheta, options)
- function [jVal, gradient] = costFunction (thetaVec)
  From thetaVec get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ and then use fwd/back to compute $D^{(1)}, D^{(2)}, D^{(3)}, J(\Theta)$ then unroll to get gradientVec.

9.5 Gradient checking

Numerical estimation of gradients

To check our code is working we can calculate the gradient by central differencing and then check that our approximation is close to DVec.

i.e.

$\theta \in \mathbb{R}^n$ (our unrolled version of $\theta^{(1)}, \theta^{(2)}, \ldots$

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

$$\frac{\partial}{\partial \theta_i} J(\theta) = \frac{J(\theta_1, \ldots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \ldots, \theta_n) - J(\theta_1, \ldots, \theta_{i-1}, \theta_i - \varepsilon, \theta_{i+1}, \ldots, \theta_n)}{2\varepsilon}$$

```
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;
```

EPSILON = 10^{-4} is a good number

Check that gradApprox ≈ DVec

Be sure to disable the gradient checking code before training the classifier as it will slow the code down alot.

For gradient descent and advanced optimisation we need an initial value for theta, θ

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

Set initialTheta = zeros(n, 1) ???

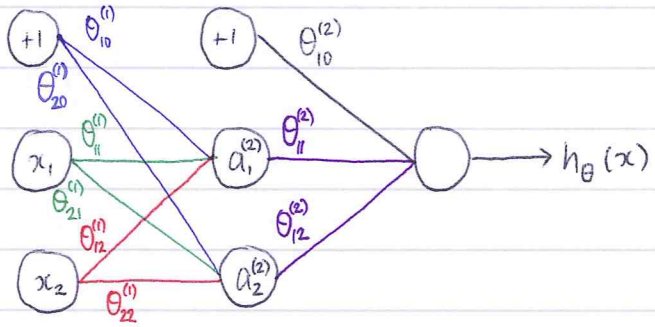
9.6 Random Initialisation

How do we choose our initial Theta? It turns out that in a neural network we can't choose Θ

$$\Theta_{ij}^{(l)} = x \quad \forall\, i,j,l.$$

If we do, the parameters corresponding to inputs going into each hidden unit are identical

e.g.



The problem of symmetric weights

$$\underbrace{\Theta_{10}^{(1)} = \Theta_{20}^{(1)} \qquad \Theta_{11}^{(1)} = \Theta_{21}^{(1)} \qquad \Theta_{12}^{(1)} = \Theta_{22}^{(1)}}$$

$$\Rightarrow a_1^{(2)} = a_2^{(2)}$$

$$\Theta_{11}^{(2)} = \Theta_{12}^{(2)} \Rightarrow \delta_1^{(2)} = \delta_2^{(2)}$$

also

$$\Rightarrow \frac{\partial J(\Theta)}{\partial \Theta_{10}^{(1)}} = \frac{\partial J(\Theta)}{\partial \Theta_{20}^{(1)}} \Rightarrow \Theta_{10}^{(1)} = \Theta_{20}^{(1)},$$

$$\frac{\partial J(\Theta)}{\partial \Theta_{11}^{(1)}} = \frac{\partial J(\Theta)}{\partial \Theta_{21}^{(1)}} \Rightarrow \Theta_{11}^{(1)} = \Theta_{21}^{(1)},$$

and

$$\frac{\partial J(\Theta)}{\partial \Theta_{12}^{(1)}} = \frac{\partial J(\Theta)}{\partial \Theta_{22}^{(1)}} \Rightarrow \Theta_{12}^{(1)} = \Theta_{22}^{(1)}$$

on the next iteration

$\Rightarrow a_1^{(2)} = a_2^{(2)}$ and round and round we go!

This means our neural network becomes degenerate and unable to calculate interesting functions

Random Initialisation: Symmetry Breaking

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

e.g.

```
Theta(1) = rand(10,11) * (2*INIT_EPSILON) - INIT_EPSILON;
Theta(2) = rand(1,11) * (2*INIT_EPSILON) - INIT_EPSILON;
```

N.B. rand(m,n) gives a m×n matrix whose elements are randomly drawn from the distribution U(0,1)

9.7 Putting it together

Training a neural network:

1. Pick a network architecture (connectivity pattern between neurons
   → No. input units: Dimension of features x
   → No. output units: Number of classes
   → Reasonable default: 1 hidden layer, or if >1, have the same number of hidden units in every layer (usually the more the better)
   → More hidden units than input features is a good idea

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

→ for $i = 1 : m$ {

Perform fwd prop and back prop on example $(x^{(i)}, y^{(i)})$
(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, 3, \ldots, L$.)
$a^{(l)} := \text{sigmoid}(\theta^{(l-1)} * a^{(l)})$;
$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} * (a^{(l)})^T$; }

Compute $\frac{\partial J(\theta)}{\partial \theta_{jk}^{(l)}}$

5. Use gradient checking to compare $\frac{\partial J(\theta)}{\partial \theta_{jk}^{(l)}}$ computed back prop vs numerical estimate of gradient of $J(\theta)$
   → Then disable gradient checking code
6. Use gradient descent or advanced optimisation method with back-propagation to try to minimize $J(\theta)$ as a function of parameters θ

N.B. $J(\theta)$, for neural networks, in general is non-convex so subject to local minima but in practice this is not in general a massive problem.

10 Advice for applying machine learning

10.1 Deciding what to try next

Debugging a learning algorith:
Suppose you have implemented regularized linear regression to predict housing prices,

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

However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try

next?

- Get more training examples
- Try a smaller set of features
- Try getting additional features
- Try adding polynomial features
- Try decreasing λ
- Try increasing λ

Machine Learning diagnostic:
This is a test you can run to gain insight into what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance. Diagnostics can take time to implement but can be a good use of time.

10.2 Evaluating a hypothesis

We have seen how if we overfit our training data our hypothesis fails to generalise to new examples not in our training set. For a small number of feature overfitting can be picked up by simply plotting our hypothesis against our training data but for a large number of features it becomes harder to detect.

Training/testing procedure for linear regression

- Take 70% of your data for training and use the remaining 30% for testing
- Learn the parameter θ from your training set (by minimising $J(\theta)$)
- Compute your test error:

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

For logistic regression the proceedure is the same but the test error is given by

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

or we can use the "misclassification error" or "0/1 misclassification error":

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

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

10.3 Model Selection and Train Validation Test Sets

In general, once parameters $\theta_0, \theta_1, \ldots, \theta_n$ have been fit to some set of data (the 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 generalisation error.

Model selection:
Suppose you want to decide what degree polynomial to use to fit your data. Let d be the degree

$d=1$ $\quad h_\theta(x) = \theta_0 + \theta_1 x \quad \rightarrow \theta^{(1)} \rightarrow J_{test}(\theta^{(1)})$

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

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

$\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots$

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

↑ by fitting to training data ↑ error on test data

We could pick d with the lowest test set error $J_{test}(\theta^{(d)})$ but now we have no data left to do a generalisation error on since clearly the test data is now biased.

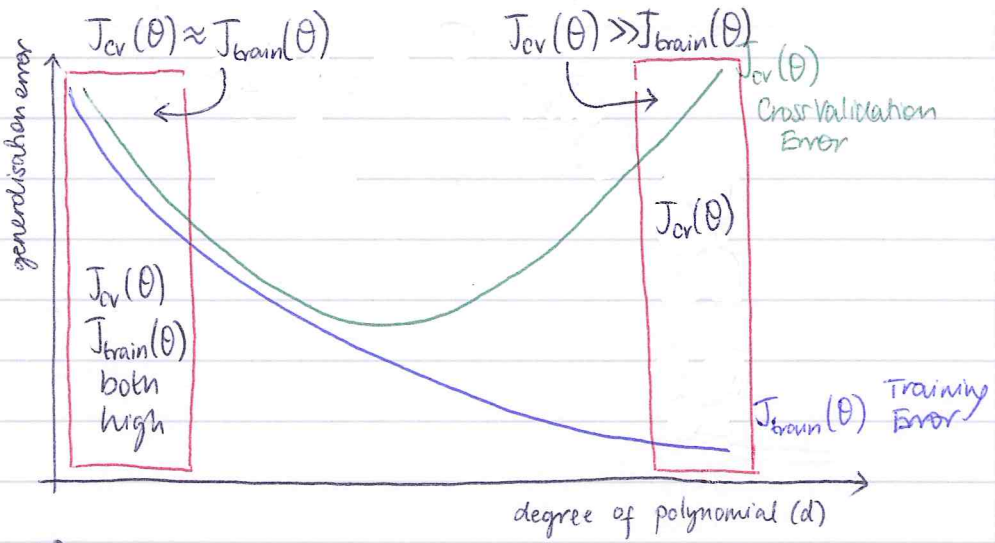
Cross-Validation:

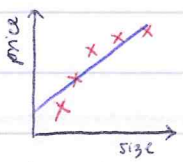
We split our data into 3 sets...

60%	20%	20%
Training Set	Cross Validation set	Test set
$(x^{(i)}, y^{(i)})$	$(x_{cv}^{(i)}, y_{cv}^{(i)})$	$(x_{test}^{(i)}, y_{test}^{(i)})$
m	m_{cv}	m_{test}

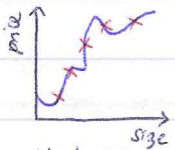
→ we minimise $J(\theta)$ on the training set for a range of d
→ choose d that gives the minimum $J_{cv}(\theta^{(d)})$
→ estimate generalisation error using your test set i.e. $J_{test}(\theta)$

10.4 Diagnosing bias vs variance





High Bias
Underfit



High Variance
Overfit

10.5 Choosing the regularisation parameter λ

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

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

→ Try a range of λ and for each minimise $J(\theta)$ on the training set to find $\theta^{(i)}$

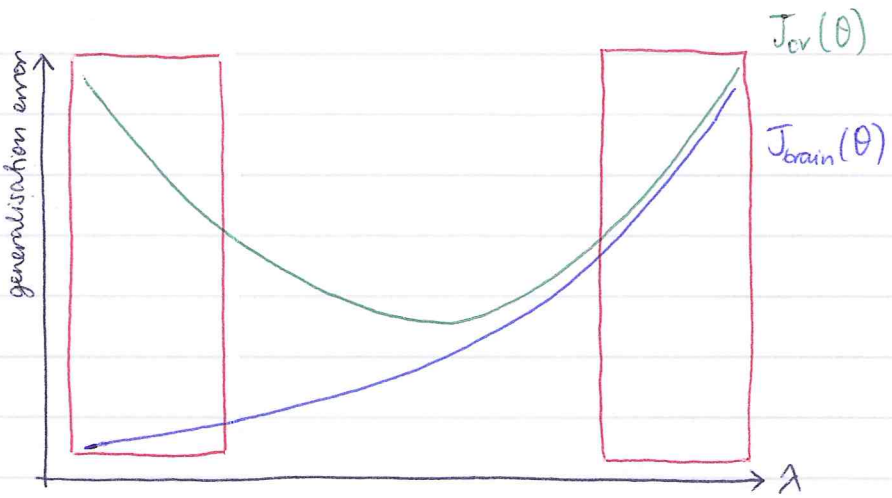
→ Pick λ such that $J_{cv}(\theta^{(i)})$ is minimised

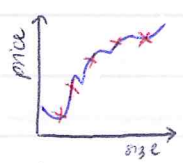
→ Use the test set to estimate the generalisation error

Define $J(\theta)$ as above and...

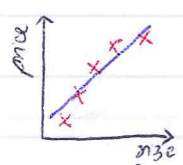
$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

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



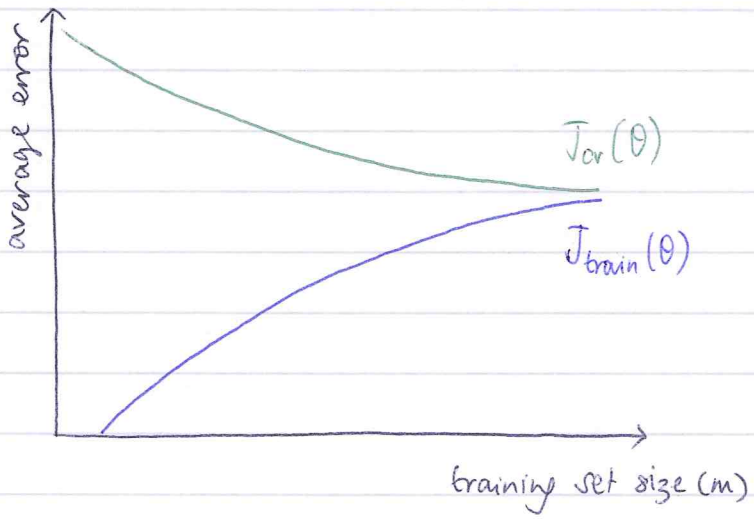


High Variance
Overfit

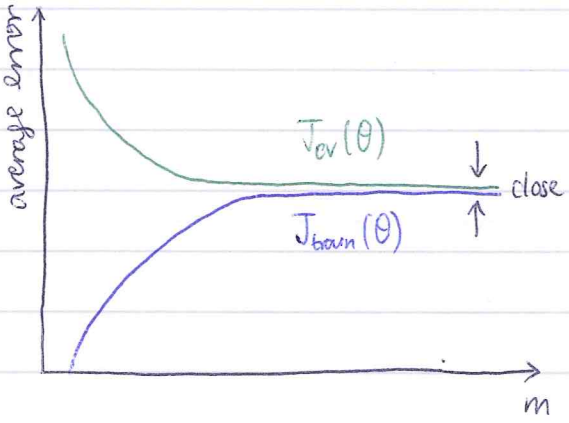


High Bias
Underfit

10.6 Learning Curves

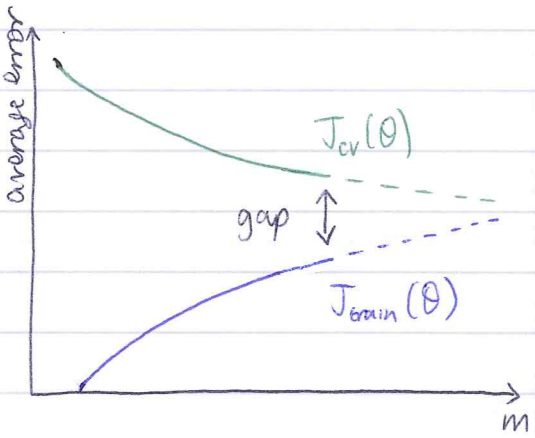


High Bias ↓



Here getting more data will not help

High Variance ↓



Here getting more data will help

10.7 Deciding what to do next revisited...

- → Get more training examples
- → Try a smaller set of features
- → Try increasing λ

} fixes high variance

- → Try decreasing λ
- → Try getting additional features
- → Try adding polynomial features

} fixes high bias

Neural Networks and Overfitting:

* "Small" neural networks - fewer parameters
  - prone to underfitting
  - computationally cheaper

* "Large" neural networks - more parameters
  - prone to overfitting
  - computationally expensive

  Regularisation can be used to address overfitting here

* We can use our cross validation techniques to decide how many units to put in the hidden layer of our neural network (if we have just one hidden layer)

11 Machine Learning System Design

11.1 Prioritising what to work on

Example: Building a spam classifier.

Supervised learning. x = features of email, y = spam (1) / not spam (0

Features x: choose 100 words indicative of spam / not spam

e.g.

$$x = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ \vdots \end{pmatrix} \begin{matrix} \text{Leena} \\ \text{buy} \\ \text{deal} \\ \text{discount} \\ \\ \end{matrix} \qquad x \in \mathbb{R}^{100}$$

$$x_j = \begin{cases} 1 & \text{if word } j \text{ appears} \\ 0 & \text{otherwise} \end{cases}$$

N.B. In practice, take most frequently occurring n words (10K-50K) in training set, rather than manually picking 100

How does one best spend their time to make their classifier have a low error?

→ Collect lots of data?
  → e.g. "honeypot" project.

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

→ Develop sophisticated features for message body, e.g. should singular and plural versions be treated as the same word? Punctuation features?

→ Develop sophisticated algorithm to detect misspellings (e.g. mOrtgages, med1cine, w4tches.)

11.2 Error analysis

Recommended approach:

→ Start with a simple algorithm that you can implement quickly. Test it on your cross-validation data

→ Plot learning curves to decide if more features, more data ect. are likely to help

→ Error analysis: Manually examine the examples (in cross-validation set) that your algorithm made errors on. See if you can spot any systematic trend in what type of examples it is making errors on.

Example: Spam classifier

$m_{cv} = 500$ examples in the cross-validation set

Suppose the algorithm misclassifies 100 emails.

Manually examine and categorise the errors based on:

(i) what type of email it is

(ii) what features do you think would have helped classify them correctly

e.g. Email Type:		Email Features	
Pharma	12	Deliberate Misspelling	5
Replica / fake	14	Unusual routing	16
Phishing	53	Unusual punctuation	32
Other	31		

→ gives an indication of fruitful avenues

The importance of numerical evaluation:

- Should discount / discounts / discounted / discounting be treated as the same word?
- Could use "stemming" software (e.g. "Porter stemmer")
- May have a negative impact by removing differentiation between some words
- Error analysis may not be helpful in this situation.
- Only solution is to try it and see if it works in a quick and dirty implementation
- The cross-validation error of the algorithms performance with and without stemming will allow us to judge its effectiveness
- Single real number evaluation metric

11.3 Error metrics for skewed classes

Cancer classification example

→ train logistic regression model $h_\theta(x) \rightarrow y = 1$ if cancer, $y = 0$ otherwise

→ suppose you find 1% error on test set (99% correct diagnoses)

→ actually, only 0.5% of patients have cancer

→ this is an example of a SKEWED CLASS i.e. the class occurs on an extreme percentage of the population in this case close to zero

→ in this case its difficult to judge if our algorithm is performing well by simply looking at our usual error meth

→ afterall even if we always have $y = 0$ our error is just 0.5%!

Precision / Recall Error metrics:

Predicted Class \ Actual Class	1	0
1	True +ve	False +ve
0	False -ve	True -ve

case: $y=1$ in presence of rare class we wish to detect

Precision: Of all the patients where we predicted $y=1$, what fraction actually has cancer?

$$= \frac{\text{True +ve}}{\#\text{ predicted +ve}} = \frac{\text{True +ve}}{\text{True +ve} + \text{False +ve}}$$

Recall: Of all the patients which actually have cancer, what fraction did we correctly detect as having it?

$$= \frac{\text{True +ve}}{\#\text{ actul +ve}} = \frac{\text{True +ve}}{\text{True +ve} + \text{False -ve}}$$

N.B.
- High precision and recall values signify good performance
- Note if we choose $y=0$ always here

11.4 Trading off precision and recall

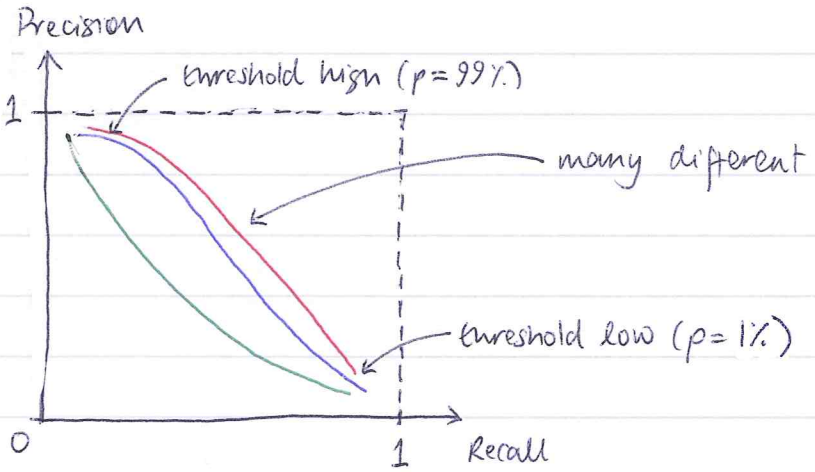
Logistic regression: $0 \leq h_\theta(x) \leq 1$

→ predict 1 if $h_\theta(x) \geq p$
0 if $h_\theta(x) < p$ } so far we've looked at $p = 0.5$

→ suppose we want to predict $y=1$ (cancer) only if we are very confident.
→ we can take a larger p
→ results in higher precision and lower recall

→ suppose we want to avoid missing too many cases

of cancer (avoid false -ves)
→ we can take a lower p
→ results in higher recall, lower precision



F_1 Score (F Score)
→ How do we compare precision/recall values for different algorithms
→ Ideally we want a single real valued metric

Example

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

→ Average is a bad idea as it's likely to have a reasonably consistent value (since the two have a trade off for different values of the threshold)
→ F_1 score or F score $= \frac{2PR}{P+R}$

11.5 Data for machine learning

Designing a high accuracy learning system

Studies have shown that inferior classification algorithms will often outperform better algorithms given more data

"It's not who has the best algorithm that wins.
It's who has the most data."

When is this true?

Large data set rationale:

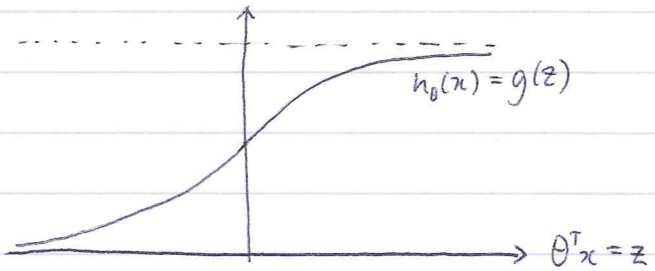
→ Assume features $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately
→ A useful test is to see if you could classify it with the given data (or a human expert if it's more complicated than classifying spam)
→ Use a learning algorithm with many parameters (e.g. logistic regression / linear regression with many features; neural network with many hidden units). → LOW BIAS ALGORITHM $J_{train}(\theta)$ small
→ Use a very large training set (unlikely to overfit)
→ LOW VARIANCE ALGORITM $J_{train}(\theta) \approx J_{test}(\theta)$
→ LOW BIAS and LOW VARIANCE
⇒ $J_{test}(\theta)$ small

12 Support Vector Machines

12.1 Optimisation objective

Alternative view of logistic regression

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



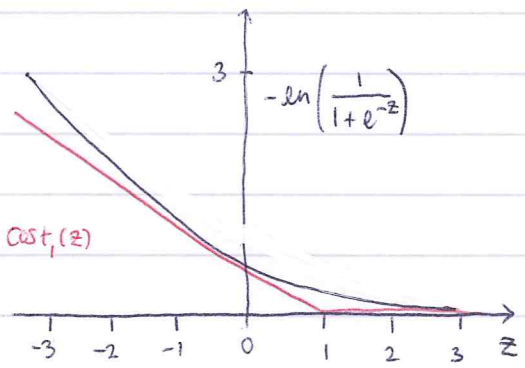
if $y=1$ need $h_\theta(x) \approx 1$
and $\theta^T x \gg 0$

if $y=0$ need $h_\theta(x) \approx 0$
and $\theta^T x \ll 0$

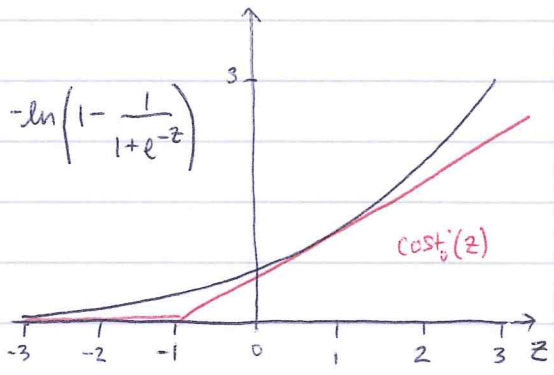
Cost of example: $-(y \ln(h_\theta(x)) + (1-y)\ln(1-h_\theta(x)))$

$$= -y \ln\left(\frac{1}{1+e^{-\theta^T x}}\right) - (1-y)\ln\left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$$

if $y=1$ want $\theta^T x \gg 0$



If $y=0$ want $\theta^T x \ll 0$



Logistic Regression:

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

Support Vector Machines

$$\min_\theta \left\{ C \sum_{i=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 \right\}$$

C ↑ regularisation parameter

SVM hypothesis:

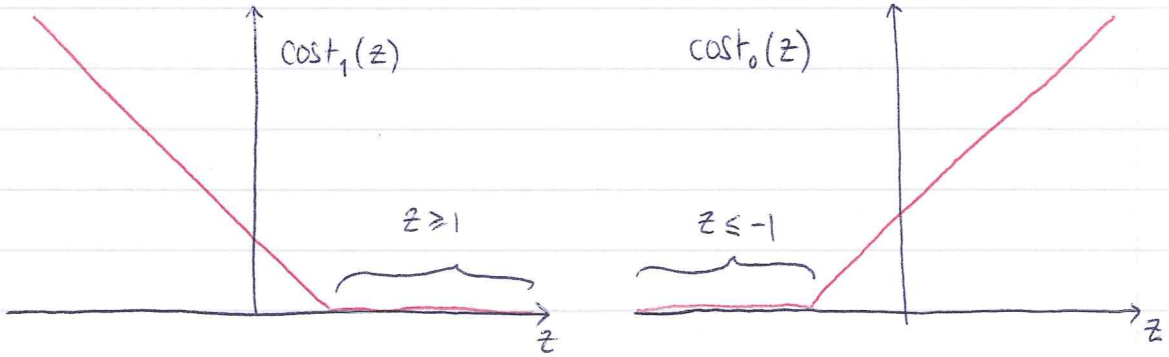
We don't get a probability like with logistic regression:

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

12.2 Large Margin Intuition

Support Vector Machine

$$\min_\theta \left\{ C \sum_{i=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_i^2 \right.$$

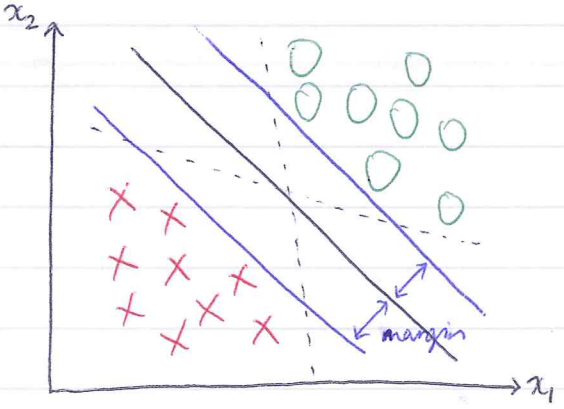


if $y=1$ need $\theta^T x \geq 1$ (not just $\theta^T x \geq 0$)

if $y=0$ need $\theta^T x \leq -1$ (not just $\theta^T x < 0$)

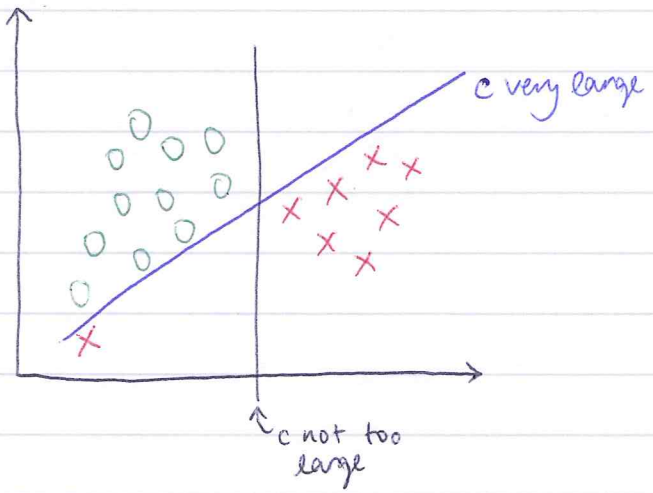
Large margin classifier: take C large

Linearly separable case:



Here there are many linear separators - SVM chooses the one with a big margin

Large margin classifier in the presence of outliers



12.3 The mathematics behind large margin classification

Recall: $\theta^T x = \theta \cdot x$ (φ = angle between vectors θ and x)

$= |\theta||x| \cos\varphi$

$= (\text{length of projection of } x \text{ onto } \theta)|\theta| = p|\theta|$ (with direction)

SVM decision boundary:

$$\min_\theta \left\{ \underbrace{\frac{1}{2}\sum_{j=1}^{n=1} \theta_j^2}_{=|\theta|^2/2} \right\} \quad \text{s.t.} \quad \begin{cases} p^{(i)}|\theta| \geq 1 & \text{if } y^{(i)} = 1 \\ p^{(i)}|\theta| \leq 1 & \text{if } y^{(i)} = 0 \end{cases}$$

(with direction)

where $p^{(i)}$ is the length of the projection of $x^{(i)}$ onto θ

simplification: $\theta_0 = 0$

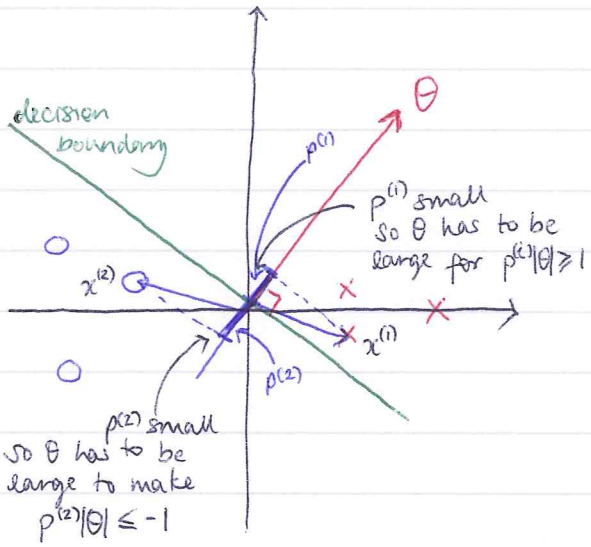
→ in this case the decision boundary looks like

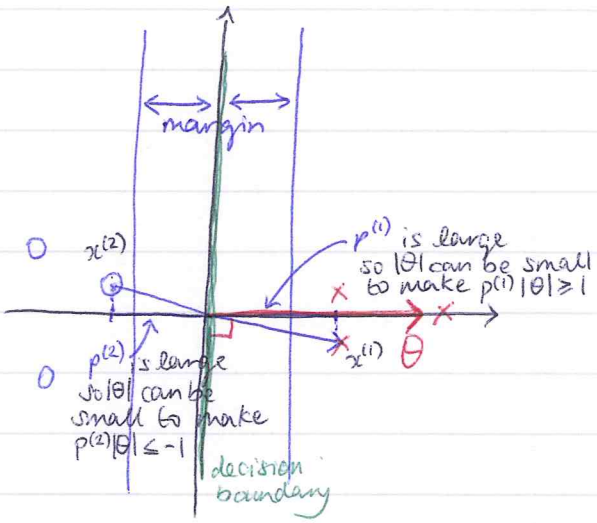
$\theta_1 x_1 + \theta_2 x_2 = 0$

→ note that θ is $\perp$ lar to the decision boundary

The algorithm tries to enforce
$p^{(i)}|\theta| \geq 1$ if $y^{(i)} = 1$
$p^{(i)}|\theta| \leq -1$ if $y^{(i)} = 0$
while minimising $\frac{1}{2}|\theta|^2$

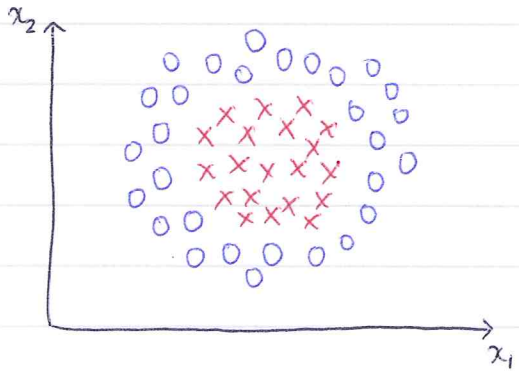
best way to do this is to maximise the $p^{(i)}$ so that θ can be small





12.4 Kernels I

Non-linear decision boundries



in logistic regression we would add features

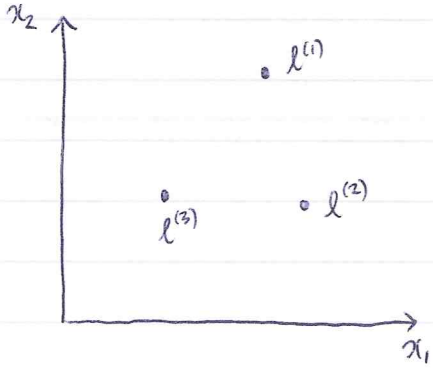
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2$$

let us write this as

$$h_\theta(x) = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \theta_4 f_4 + \theta_5 f_5$$

$$f_1 = x_1,\ f_2 = x_2,\ f_3 = x_1 x_2,\ f_4 = x_1^2 \text{ and } f_5 = x_2^2$$

Is there a different, better choice of features $f_1, f_2, \ldots$



Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

Kernels & similarity

Given x:

$$f_1 = \text{similarity}(x, \ell^{(1)}) = \exp\left(-|x-\ell^{(1)}|^2/2\sigma^2\right)$$
$$f_2 = \text{similarity}(x, \ell^{(2)}) = \exp\left(-|x-\ell^{(2)}|/2\sigma^2\right)$$
$$f_3 = \text{similarity}(x, \ell^{(3)}) = \exp\left(-|x-\ell^{(3)}|/2\sigma^2\right)$$

kernel — Gaussian kernel

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

If x_1 is close to $\ell^{(1)}$ then $f_1 \approx 1$
If x_1 is far from $\ell^{(1)}$ then $f_1 \approx 0$

Predict $y=1$ when $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

12.5 Kernels II

Choosing the landmarks:

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

choose
$\ell^{(1)} = x^{(1)}, \ell^{(2)} = x^{(2)}, \ldots, \ell^{(m)} = x^{(m)}$

then

$$\left.\begin{aligned} f_1 &= \text{similarity}(x^{(i)}, \ell^{(1)}) \\ f_2 &= \text{similarity}(x^{(i)}, \ell^{(2)}) \\ &\vdots \\ f_m &= \text{similarity}(x^{(i)}, \ell^{(m)}) \end{aligned}\right\} \quad f = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_m \end{pmatrix} \qquad f_0 = 1$$

given x, compute $f \in \mathbb{R}^{m+1}$
predict $y=1$ if $\theta^T f \geq 0$

Training:

$$\min_{\theta} \left\{ C \sum_{i=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2 \right\}$$

$$\theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{pmatrix}$$ ignoring θ_0

implementation

$$\sum_{j=1}^{m} \theta_j^2 = \theta^T \theta = |\theta|^2$$

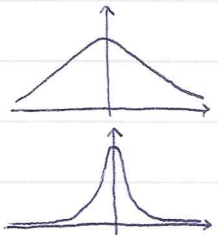
$\theta^T M \theta$

↑ depends on the kernel

SVM parameters

$C \left(= \frac{1}{\lambda}\right)$ Large C: Lower Bias, High Variance
Small C: High Bias, Low Variance

σ^2 Large σ^2: Features f_i vary smoothly
High Bias, Low Variance
Small σ^2: Features f_i vary abruptly
Low Bias, High Variance



12.6 Using an SVM

Use SVM software package (e.g. liblinear, libsvm, ...) to solve for parameters θ.

Need to specify
→ Choice of parameter C
→ Choice of kernel (similarity function)

Convention: No kernel ("linear kernel") } suitable when n large, m small
i.e. predict "y=1" if $\theta^T x \geq 0$ }

Gaussian & linear kernels are the most common

Gaussian kernel:

- $f_i = \exp\left(-\frac{|x - l^{(i)}|^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$ } suitable when n small and m large
- need to choose σ^2
- can use this to compute complex non-linear decision boundary

Some packages require the kernel (similarity) function to be implemented by the user. E.g.

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

N.B. Do perform feature scaling before using the Gaussian kernel

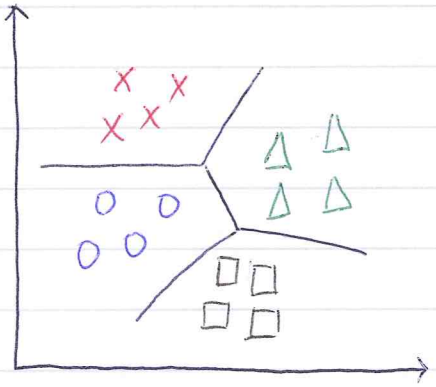
Other choices of kernel

Note: Not all similarity functions make valid kernels. For a function to be a valid kernel it must satisfy "Mercer's Theorem". This ensures an SVM package's optimisers run correctly - doesn't diverge

Many off the shelf kernels are available

- Polynomial kernel: $k(x, l) = (x^T l + c)^d$ for some constants c and d; d is called the degree
- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

Multiclass classification



$y \in \{1, 2, \ldots, K\}$

Many SVM packages already have built-in multi-class classification functionality

Otherwise one-vs-all method works
- → train K SVMs, one to distinguish $y=i$ from the rest for $i = 1, 2, \ldots, K$
- → get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$
- → for new example predict $y=i$ such that $\max_i \{(\theta^{(i)})^T x\}$

Logistic Regression vs SVM

n = number of features ($x \in \mathbb{R}^{n+1}$)
m = number of training examples

if n is large relative to m: $m \leq n$
→ use logistic regression or SVM with linear kernel
If n is small, m is intermediate: $m \geq 10n$
→ use Gaussian kernel
If n is small, m is large: $m \geq 50n$
→ create/add more features, then use logistic regression or SVM without a kernel
* Neural network is likely to work well for all of the above but may be slower to train

13 Clustering

Unsupervised learning - given data is unlabelled and the algorithm looks for structure in the data

13.2 K-means Algorithm

Input: → K (number of clusters)
→ Training $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $x^{(i)} \in \mathbb{R}^n$

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$
Repeat {

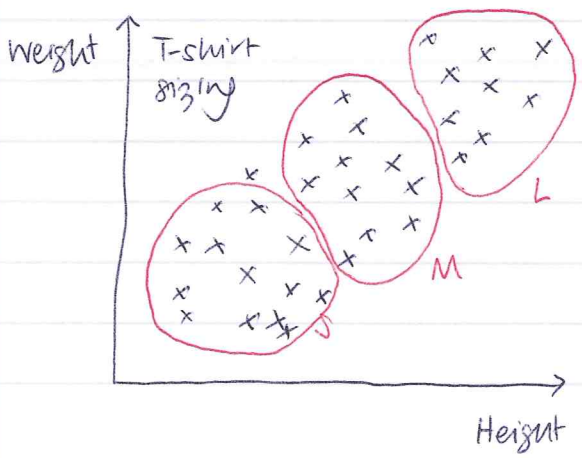
cluster assignment step:
for $i = 1 : m$
$c^{(i)} :=$ index (from $1 \to K$) of cluster centroid closest to $x^{(i)}$

move centroid step:
for $k = 1 : K$
$\mu_k :=$ average (mean) of points assigned to cluster k

}

If you end up with a cluster with no $x^{(i)}$ assigned to it, you can
(i) eliminate the cluster
or (ii) randomly reassign the cluster centres.

K-means for non-separated clusters



Market segmentation

13.3 Optimisation Objective

Notation:

$c^{(i)}$ = index of cluster $(1,2,\dots,K)$ to which $x^{(i)}$ is assigned

μ_k = cluster centroid k $(\mu_k \in \mathbb{R}^n)$

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which $x^{(i)}$ is assigned

Optimization objective

$$J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K) = \frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2$$

$$\min_{\substack{c^{(i)} \\ \mu_k}} \left\{ J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K) \right\} \qquad \text{AKA Distortion}$$

$1 \le i \le m$, $1 \le k \le K$

N.B. CLUSTER ASSIGNMENT STEP

→ minimises J over $c^{(i)}$ $i \in [1,m]$

MOVE CENTROID STEP

→ minimises J over μ_k $k \in [1,K]$

13.4 Random Initialisation

Should have $K < m$

Randomly pick K training examples

Set $\mu_1, \mu_2, \dots, \mu_K$ equal to these K examples

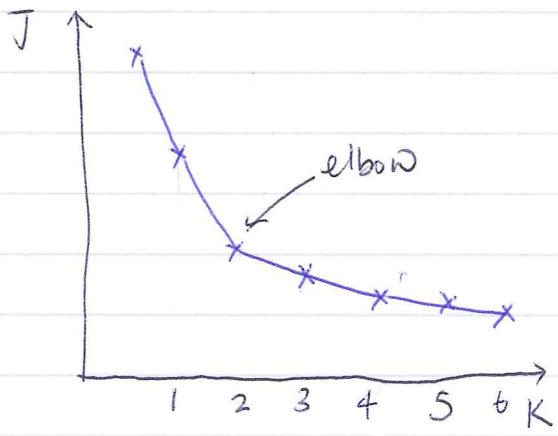
⇒ K-means is vulnerable to LOCAL OPTIMA

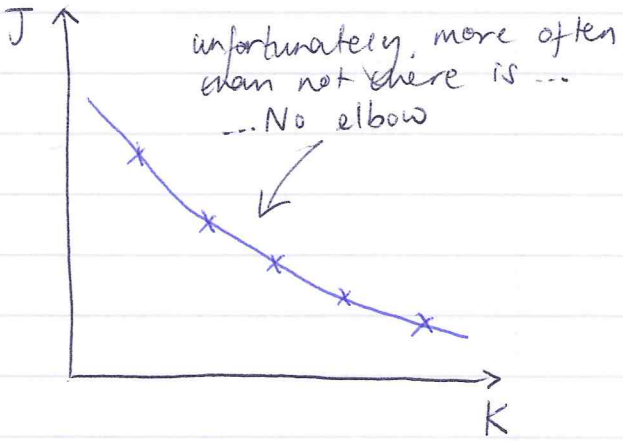
```
for i = 1:100 {
    Randomly initialize K-means
    Run K-means. Get c(1), c(2), ..., c(m), μ1, μ2, ..., μK
    Compute J(c(1), c(2), ..., c(m), μ1, μ2, ..., μK)
}
```

Pick clustering that gave the lowest J

13.5 Choosing the number of clusters

Elbow method:





N.B. J should always decrease with K

14 Dimensionality Reduction

14.1 Data compression: The data is projected onto the reduced dimension space, i.e. we represent

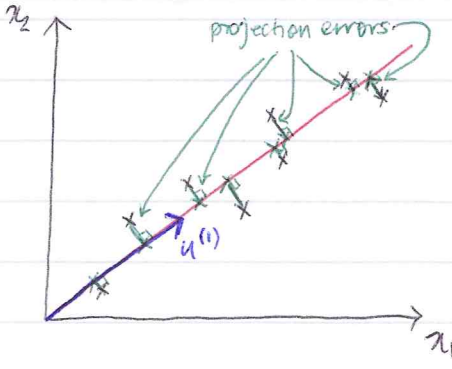
$$\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\} \rightarrow \{z^{(1)}, z^{(2)}, \dots, z^{(m)}\}$$

$$x^{(i)} \in \mathbb{R}^n \rightarrow z^{(i)} \in \mathbb{R}^k \quad k < n$$

14.2 Data visualization: Dimension reduction to size two/three can allow us to better visualize the data

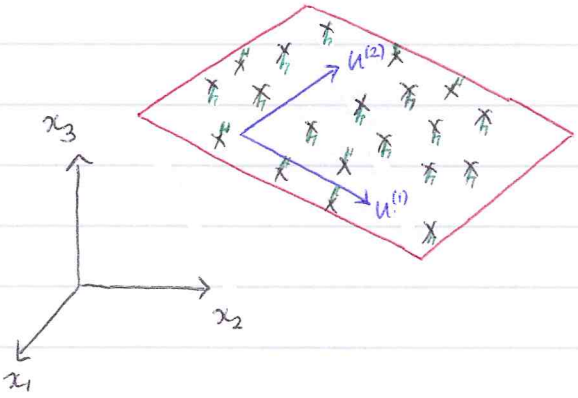
14.3 Principal Component Analysis (PCA)

Problem Formulation: 2D → 1D example ($n = 2$)



Reduce from 2D to 1D: find a direction vector $u^{(1)} \in \mathbb{R}^n$ onto which to project the data so as to minimise the projection error

3D → 2D example $(n=3)$

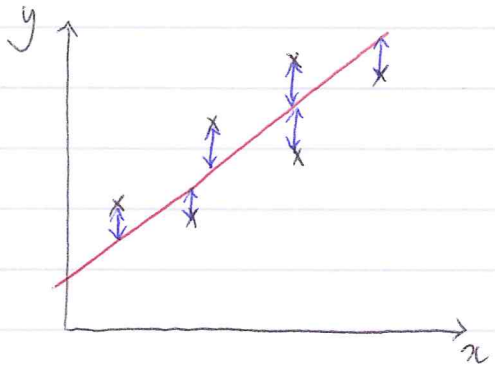


Here we find two direction vectors $u^{(1)}$ and $u^{(2)}$ onto which to project the data so as to minimise the projection error

In general: Reduce from n dimension to k dimension
Find a set of k vectors $u^{(1)}, u^{(2)}, \dots, u^{(k)}$ onto which to project the data so as to minimise the projection error

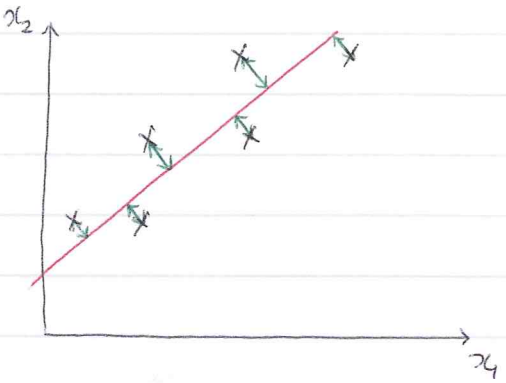
PCA is not linear regression

Linear Regression



y is viewed to be dependent on x i.e. $x^{(i)} \to y^{(i)}$
$x^{(i)}, y^{(i)} \in \mathbb{R}^n$

Principal Component Analysis



$x^{(i)} = (x_1, x_2, x_3, \dots, x_n) \in \mathbb{R}^n$
All features are treated equally

14.4 Principal Component Analysis Algorithm

Data preprocessing

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

Preprocessing (feature scaling / mean normalisation):

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

replace each $x_j^{(i)}$ with $x_j^{(i)} - \mu_j$

If features are on different scales they should be rescaled to have comparable ranges.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

$\sigma_j \leftarrow$ range / standard deviation

Algorithm.

We need to find two things...

① $z^{(i)} \in \mathbb{R}^k \leftarrow x^{(i)} \in \mathbb{R}^n$

② $u^{(1)}, u^{(2)}, \ldots, u^{(k)} \in \mathbb{R}^n$ onto which we project our $x^{(i)}$ to find our $z^{(i)}$

To reduce data from n-dimensions to k-dimensions,
Compute "covariance matrix":

$$\underbrace{\Sigma}_{n \times n} = \frac{1}{m} \sum_{i=1}^{n} \underbrace{(x^{(i)})}_{n \times 1} \underbrace{(x^{(i)})^T}_{1 \times n} \qquad = \text{Sigma}$$

Compute "eigenvectors" of the covariance matrix Σ:

```
[U, S, V] = svd(Sigma);
```

$$U = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{n \times n}$$

take the first k columns

i.e.

$$U_{reduce} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{n \times k}$$

$$z = \underbrace{U_{reduce}^T}_{k \times n} \underbrace{x}_{n \times 1}, \quad z: k \times 1$$

Summary

After mean normalisation (ensure every feature has zero mean) and optionally feature scaling:

$$Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})(x^{(i)})^T \qquad = X' * X / m;$$

N.B. as usual

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

```
[U,S,V] = svd(Sigma);
Ureduce = U(:,1:k);
z = Ureduce' * x;
```

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

14.5 Choosing the number of principal components

Average squared projection error $= \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2$

Total Variance in the data $= \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \|^2$

Typically choose k to be the smallest value such that

$$\frac{\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01$$

i.e. such that "99% of the variance is retained".

Naively we would do the following...

for $k = 1 : n$

Compute $U_{reduce}, z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x^{(1)}_{approx}, x^{(2)}_{approx}, \ldots, x^{(m)}_{approx}$

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x^{(i)}_{approx} \|^2}{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01$$

If not we choose $k-1$

end;

But actually our method uses

$$[U, S, V] = svd(Sigma);$$

S is a diagonal matrix

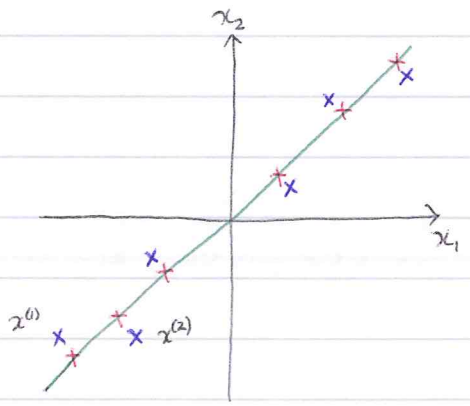
$$S = \begin{bmatrix} S_{11} & & & \\ & S_{22} & & \\ & & S_{33} & \\ & & & \ddots \\ & & & & S_{nn} \end{bmatrix}$$

For given k

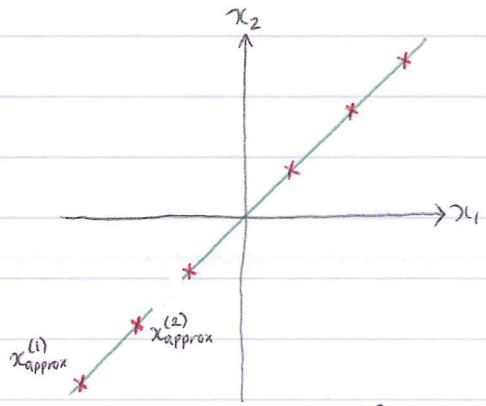
$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} = \rightarrow \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

i.e. pick the smallest value of k such that

14.6 Reconstruction from Compressed Representation



$z = U_{reduce}^{T} x$



$z \in \mathbb{R} \rightarrow x \in \mathbb{R}^2$

$x^{(i)}_{approx} = U_{reduce} \cdot z^{(i)}$

$\in \mathbb{R}^n$ $\quad n \times k \quad k \times 1$

$n \times 1$

14.7 Advice for applying PCA

Supervised learning speedup

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

Extract Inputs:

Unlabelled data set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^n$ n large

$\downarrow$ PCA

$z^{(1)}, z^{(2)}, \ldots, z^{(m)} \in \mathbb{R}^k$ $k < n$

New training set:

$(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \ldots, (z^{(m)}, y^{(m)})$

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

New example x

$x \rightarrow z \rightarrow h_\theta(z)$

for prediction

Note: Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on the training set. This is like a parameter.

$n = 10k$ is not unrealistic

Application:

→ Compression
- Reduce memory needed to store data
- Speed up learning algorithm
  - choose k by % of variance retained

→ Visualization
- $k = 2$ or 3

Bad use of PCA: To prevent overfitting (might work but bad!)

→ PCA reduces the number of feature from n to k

→ This may work but is not a good way to address overfitting as it does not take into account y values

→ Better to just use regularisation

→ PCA should really only be used to speed up your learning algorithm
→ Try doing ML without PCA first and only if the performance is slow implement PCA

15 Anomaly Detection

15.1 Problem Motivation

Example: Fraud detection
→ $x^{(i)}$ = features of user i's activities
→ Model $p(x)$ from data
→ Identify unusual users by checking which have $p(x) < \varepsilon$
Example: Monitoring computers in a data centre
Example: Manufacturing - identifying faulty produce

15.2 Gaussian distribution

$$p(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)}-\mu)^2$$

15.3 Algorithm

Density estimation

Training set: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ each $x^{(i)} \in \mathbb{R}^n$
$p(x) = p(x_1;\mu_1,\sigma_1^2)\,p(x_2;\mu_2,\sigma_2^2)\dots p(x_n;\mu_n,\sigma_n^2) = \prod_{j=1}^{n} p(x_j;\mu_j,\sigma_j^2)$

Anomaly detection algorithm

1. Choose features x_i that you think might be indicative of anomalous examples
2. Fit parameters $\mu_1, \mu_2, \dots, \mu_n, \sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$

$$\rightarrow \mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

$$\rightarrow \sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$$

3. Given a new example x compute $p(x)$:

$$\rightarrow p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \varepsilon$

15.4 Developing and evaluating an anomaly detection system

Real number evaluation metric

- When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm
- Assume we have some labelled data of anomalous and non-anomalous examples ($y = 1$ $y = 0$ respectively) i.e. some of our data is labelled
- Assume Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ are non-anomalous examples (exclude examples known to be anomalous)
- Cross validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), (x_{cv}^{(2)}, y_{cv}^{(2)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
- Test set: $(x_{test}^{(1)}, y_{test}^{(2)}), (x_{test}^{(2)}, y_{test}^{(2)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

Example: 10000 good and 20 anomalous
Training set: 6000 good
Cross validation set: 2000 good 10 anomalous
Test set: 2000 good 10 anomalous

Possible evaluation metrics
- True +ve, False +ve, True -ve, False -ve
- Precision / Recall
- F_1 score

Can use cross-validation set to choose ε

15.5 Anomaly detection vs. supervised learning

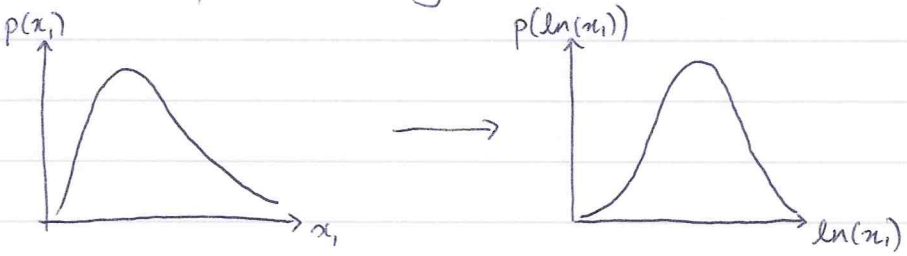
Anomaly Detection	vs.	Supervised Learning
* Very small number of positive example ($y=1$)		* Large number of positive and negative examples
* Large number of -ve examples		
* Many different "types" of anomalies - hard for an algorithm to learn what +ve examples look like; future anomalies are not like past ones		* Enough +ve examples for the algorithm to get a sense of what +ve examples look like; future +ve examples likely to be similar to ones in the training set
* Examples - Fraud detection - Manufacturing - Monitoring machines in a data centre	in some cases →	* Examples - Email - spam detection - Weather prediction - Cancer classification

15.6 Choosing what features to use

Non-Gaussian features

Plot a histogram
- can use hist function in octave

Examples of transforming data to be more Gaussian



can use other functions too
- $\ln(x + c)$
- x^n

hist(x, n)
↳ number of buckets

Most common problem:
p(x) is comparable for normal and anomalous features
One can try to find functions which will make these stand out rather than

Choose features which take on unusually large or small values in the event of an anomaly

Example: Computer monitoring in a data centre
x_1 = memory use of computer x_3 = CPU Load
x_2 = # disk accesses/sec x_4 = network traffic
maybe ... $x_5 = x_3 / x_4$? or $x_6 = x_3^2 / x_4$?

15.7 Multivariate Gaussian Distribution

$x \in \mathbb{R}^n$ ... x_i are not independent

Parameters: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix)

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} \underbrace{|\Sigma|}_{\text{determinant of } \Sigma}^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

Recall - diagonal entries give variance

15.8 Anomaly detection using the multivariate Gaussian distribution

As before:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad \Sigma = \frac{1}{m}\sum (x^{(i)}-\mu)(x^{(i)}-\mu)^T$$

Given a new example

$$p(x) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

flag as an anomaly if $p(x) < \varepsilon$

Original Model	vs.	Multivariate Gaussian
* Manually create features to capture where features combined give indicators		* Automatically captures correlations between features
* Computationally cheaper - scales better to large n (n = 10K, 100K)		* Computationally expensive - requires calculation of Σ^{-1}
* Okay even if m is small		* Must have $m > n$ else Σ is non-invertible. ($m \geq 10n$)

16 Recommender Systems

16.1 Example: Predicting movie ratings

Notation

n_u = # users

n_m = # movies

$r(i,j) = 1$ if user j has rated movie i

$y^{(i,j)}$ = rating given by user j to movie i (defined $\Leftrightarrow r(i,j) = 1$)

Users rate movies with zero - five stars ← usually this is one but we use zero for mathematical convenience

16.2

movie \ users	(1) Alice	(2) Bob	(3) Carol	(4) Dave	Romance	Action
Love at last	1	0	?	5	0.99	0.01
Romance 4ever	0	0	5	4	0.98	0.02
Love puppies	0	?	4	0	0.85	0.05
Car chases	4	5	0	?	0.05	0.95
Big action	5	5	?	0	0.01	0.99

idea is we want to fill in the gaps so we can make recommendations

$\theta^{(j)}$ = parameter vector for user j $\in \mathbb{R}^{n+1}$

$x^{(i)}$ = feature vector for movie i

Predicted rating of user j for movie i

$$h_\theta(x^{(i)}) = (\theta^{(j)})^T x^{(i)}$$

$m^{(j)}$ = # movies rated by user j

To learn $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \left\{ \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \right\}$$

To learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)},\theta^{(2)},\dots,\theta^{(n_u)}} \underbrace{\left\{ \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(\theta_k^{(j)} \right)^2 \right\}}_{J(\theta^{(1)},\dots,\theta^{(n_u)})}$$

Gradient descent update

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} \qquad \text{for } k=0$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \underbrace{\left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)}_{\frac{\partial}{\partial \theta_k^{(j)}} \left\{ J(\theta^{(1)},\dots,\theta^{(n_u)}) \right\}} \qquad \text{for } k \neq 0$$

16.3 Collaborative filtering

Learning features:

* Given $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \left\{ \frac{1}{2} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2 \right\}$$

* Given $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ to learn $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$:

$$\min_{x^{(1)},x^{(2)},\dots,x^{(n_m)}} \left\{ \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 \right\}$$

Given $x^{(1)}, \dots, x^{(n_m)} \rightarrow$ estimate $\theta^{(1)}, \dots, \theta^{(n_u)}$

Given $\theta^{(1)}, \dots, \theta^{(n_u)} \rightarrow$ estimate $x^{(1)}, \dots, x^{(n_m)}$

Can do this recursively $\theta \rightarrow x \rightarrow \theta \rightarrow x \rightarrow \dots$

16.4 Collaborative filtering algorithm

Rather than solving for θ and then x recursively we do them simultaneously.

Minimising over $x^{(1)}, \ldots, x^{(n_m)}$ and $\theta^{(1)}, \ldots, \theta^{(n_u)}$ simultaneo

$$J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}) =$$
$$= \frac{1}{2} \sum_{(i,j):r(i,j)} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$$\min_{\substack{x^{(1)}, \ldots, x^{(n_m)} \\ \theta^{(1)}, \ldots, \theta^{(n_u)}}} \left\{ J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}) \right\}$$

Use $x^{(i)} \in \mathbb{R}^n$ and $\theta^{(j)} \in \mathbb{R}^n$ i.e. x_0 and θ_0 not needed

Algorithm

1. Initialize
   $x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$
   to small random values
2. Minimise $J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)})$ using gradient descent (or other advanced algorithm). Eg. for every $j = 1, \ldots, n_u$ and $i = 1, \ldots, n_m$:
   $$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$
   $$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$
3. For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x = (\theta^{(j)})^T x^{(i)}$

16.5 Vectorization: Low rank matrix factorization

$$\begin{pmatrix} (\theta^{(1)})^T x^{(1)} & (\theta^{(1)})^T x^{(1)} & \cdots & (\theta^{(n_u)})^T x^{(1)} \\ (\theta^{(1)})^T x^{(2)} & (\theta^{(2)})^T x^{(2)} & \cdots & (\theta^{(n_u)})^T \\ \vdots & \vdots & & \vdots \\ (\theta^{(1)})^T x^{(n_m)} & (\theta^{(2)})^T x^{(n_m)} & \cdots & (\theta^{(n_u)})^T x^{(n_m)} \end{pmatrix} = Y$$

Finding related movies

For each product, i, we learn a feature vector $x^{(i)} \in \mathbb{R}$.
→ x_1 = romance, x_2 = action, x_3 = comedy, ...
How to find movies j related to movie i
→ small $\|x^{(i)} - x^{(j)}\| \Rightarrow$ movies i and j are "similar"

16.6 Implementation detail: mean normalisation

We mean normalise but do not do feature scaling since the features are already on the same scales

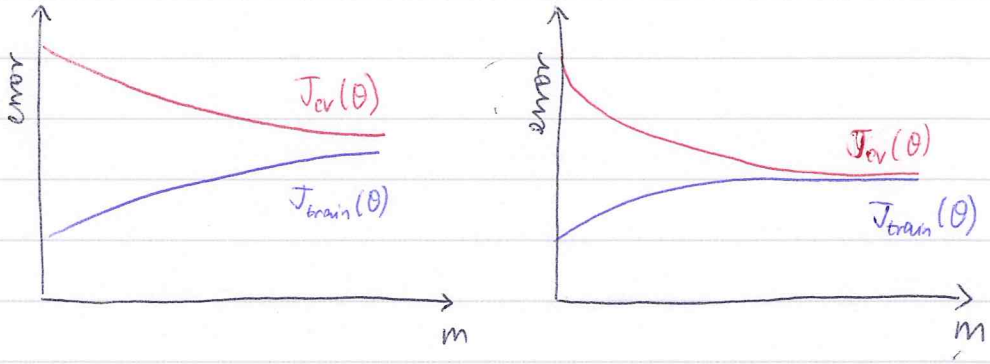
Mean normalisation means that our fitted parameters would give a result consistent with predicting the average ratings for a user for which we have no data

17 Large Scale Machine Learning

17.1 Learning with large data-sets

Recall
"It's not who has the best algorithm who wins. It's who has the most data"

Suppose you have a very large dataset, say $m = 100mn$ it's a good idea to train it on a small randomly chosen subset of the data first to see if more data will help - i.e. $m = 1000$ and then plots learning curves of $J_{train}(\theta)$ and $J_{cv}(\theta)$ if they indicate high variance for small m



17.2 Stochastic Gradient Descent

When we have large datasets the regression algorithms we have looked at so far become computationally expensive

Recall:

Linear regression with gradient descent

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \quad , \quad J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x) - y^{(i)} \right)^2$$

Repeat { $\theta_j := \theta_j - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}}_{= \frac{\partial}{\partial \theta_j} J_{train}(\theta)}$

for every $j = 0, 1, \ldots, n$ }

also known as batch gradient descent because we compute the term $\frac{\partial}{\partial \theta_j} J_{train}(\theta)$ for each $j = 0, 1, \ldots, n$

In Stochastic Gradient Descent we instead train on each data pair $(x^{(i)}, y^{(i)})$ at a time i.e. define

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

Then

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

1. Randomly shuffle dataset.

2. Repeat {

for $i = 1, 2, \dots, m$ {

$\theta_j := \theta_j - \alpha \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$ $\leftarrow \frac{\partial}{\partial \theta_j} \text{cost}(\theta, (x^{(i)}, y^{(i)}))$

(for $j = 0, 1, \dots, n$)

}

}

$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

Unlike batch gradient the algorithm does not monotonically converge to the minimum it can however be much faster for large data-sets.

17.3 Mini Batch Gradient Descent

* Batch Gradient Descent - use all m examples in each iteration
* Stochastic Gradient Descent - use one example in each iteration
* Mini-Batch Gradient Descent - use b examples in each iteration

b = mini-batch size typically $2 \leq b \leq 100$

Repeat {

for $i = 1, 1+b, 1+2b, \dots$ {

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

(for every $j = 0, 1, \dots, n$)

}

}

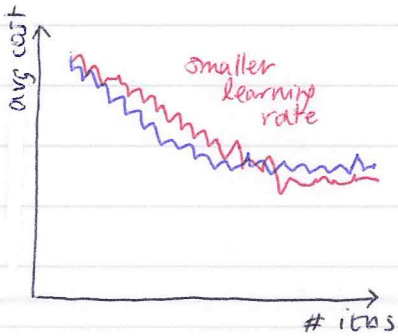
Mini-batch allows us to take advantage of partial vectorization and so can be faster than both Stochastic Gradient Descent and Batch Gradient Descent.

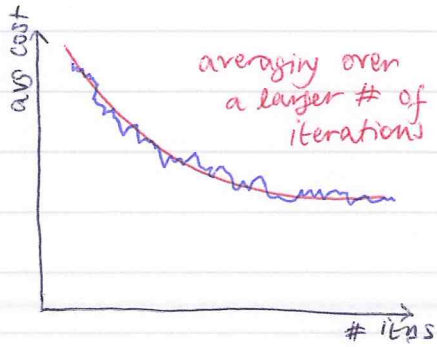
17.4 Stochastic Gradient Descent Convergence

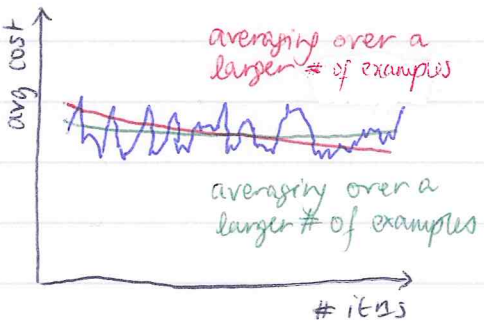
Checking for convergence:

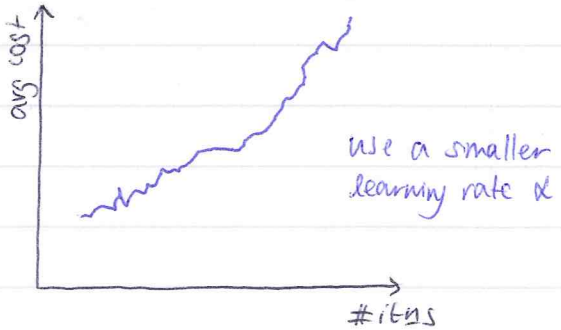
Batch Gradient Descent
- → Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent
- → $J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

Stochastic Gradient Descent
- → $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$
- → During learning compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$
- → Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by the algorithm.









To improve the convergence of the Stochastic Gradient Descent algorithm one can decrease the learning rate as the number of iterations increases. Typically one might use

$$\alpha = \frac{\text{const 1}}{\text{iteration \# + const 2}}$$

Of course this gives you an extra two parameters to play with which can be a hassle but for the right choice one might see smoother convergence.

17.5 Online learning

Example: Shipping service where user specifies origin and destination, service quotes a price and then user purchases ($y=1$) or does not purchase ($y=0$) the service.

Feature vector x captures user properties (origin / destination) and our quote and we want to learn $p(y=1|x;\theta)$ to optimise price

Repeat forever {
  Get (x,y) corresponding to user
  Update θ using (x,y)
  → $\theta_j := \theta_j - \alpha(h_\theta(x)-y)x_j \quad j=0,1,\dots,n$
}

each (x,y) is discarded once it is learnt from

Example: Product Search - Click Through Rate (CTR)
Based on the search query we want to know the probability the user will click on a link for a particular product
Show the user the 10 most likely products they will click

Online learning algorithms can
→ Learn changing user tastes / trends
→ Avoid the need to store huge amounts of data

17.6 Map reduce and data parallelism

Example : Batch Gradient Descent

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

We can split our m training examples into as many equal parts as we have computers - say c. Each computer then calculates the sum

$$temp_j^{(k)} = \sum_{i=1+(k-1)b}^{kb} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad b = m/c = \text{batch size}$$

and then combine the results at a centralised server

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{k=1}^{c} temp_j^{(k)} \qquad j = 0, 1, \ldots, n$$

Map reduce can be used when
→ algorithm can be expressed as a sum of functions over the training set.
→ works for logistic regression too

Multi-core machines:
→ Parallelising over cores rather than seperate computers re issue of network latency
→ Numerical linear algebra libraries may take advantage of parallelisation automatically

Map reduce can be used to train a neural network. In this case each core/computer computes forward propagation and back propogation on its portion of the training data followed by the derivative with respect to its portion.

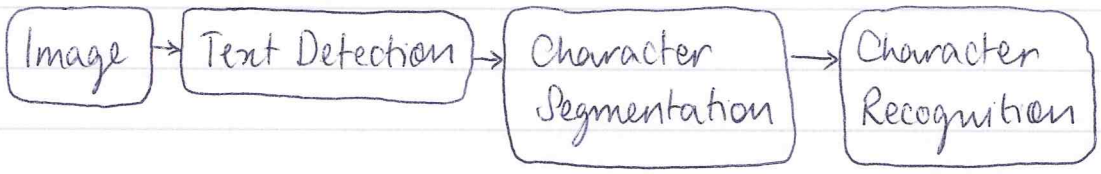
18 Application Example: Photo OCR

18.1 Problem description and pipeline

Photo Optical Character Recognition - Photo OCR

Photo OCR pipeline

1. Text detection
2. Character segmentation
3. Character classification
4. Error correction



18.2 Sliding windows

Learn what text looks like using logistic regression then use a sliding window of a fixed aspect ratio to locate text in the image. Increase the size of the window and repeat. Then use "expansion" on the pixels to identify blocks of text.

Learn what the midpoint between two characters looks like again using logistic regression. Use sliding windows method to perform character segmentation.

Finally we've done character classification using neural networks before!

18.3 Getting Lots of Data and Artificial Data

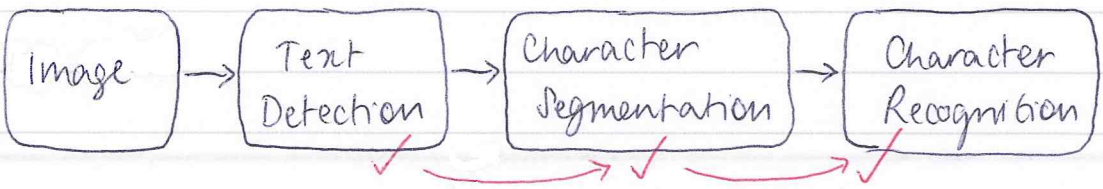
We've talked about how effective it can be to take a low bias algorithm and train it on masses of data. We can get more data by synthesising it.

We can synthesis data by taking real data and introducing distortions. In this way we can amplify the size of our data set.

Distortions should be representative of the kinds we would expect to see in our dataset.

Other ways to get data:
- → collect and label it yourself
- → "crowd sourced" data labelling e.g. "Amazon Mechanical Turk"

18.4 Ceiling Analysis: What part of the pipeline to work on next?



Replace each component in turn (in order) with a perfectly accurate set of results to see the potential gain in overall accuracy

Component	Accuracy	
Overall System	72%	↑ 17%
Text Detection	89%	↑ 1%
Char' Seg'n	90%	↑ 10%
Char' Recog'n	100%	