Machine Learning

Machine Learning

- Week1
  - What is Machine Learning?
  - Parameter Learning
    - 1. Algorithm to calculate J_{min} : Gradient descent(梯度下降)
    - 2. Gradient Descent Intuition
    - 3. Linear Algebra
- Week 2
  - 1. Multivariate Linear Regression
    - (1) Multiple Features
    - (2) Feature Scaling
    - (3) Learning Rate α
    - (4) Features and Polynomial Regression
  - 2. Computing Parameters Analytically
    - (1) Normal Equation
    - (2) Normal Equation Noninvertibility
  - 3. Ocatave/Matlab Tutorial
    - (1) Basic operations
    - (2) Moving Data Around
    - (3) Computing on Data
    - (4) Plotting Data
    - (5) Control Statement : `for` `while` `if`
    - (6) Vectorization
    - (7) Method learnt in the assignment
- Week 3
  - 1. Classification -> logistic regression
  - 2. Hypothesis Representation (function used)
    - (1) Logistic Regression Model
    - (2) Interpretation of Hpothesis Output
    - (3) Decision Boundary
  - 3. Logistic Regression Model
    - (1) Cost function
    - (2) Simplified cost function and gradient descent
    - (3) Advanced optimization
  - 4. Multi-class classification: one-vs-all
  - 5. Solving the problem of overfitting
    - (1) The problem of overfitting
    - (2) Cost function
    - (2) regularized linear regression
    - (3) regularized logistic regression
- Week4
  - 1. Motivations
  - 2. Neural Networks
    - (1) Model Representation I
    - (2) Model representation II
  - 3. Applications
    - (1) Example and Intuitions I
    - (2) Example and Intuitions II
- Week 5
  - 1. Cost Function and Backpropagation
    - (1) Cost function
    - (2) Backpropagation Algorithm
    - (3) Backpropagation Intuition

(4) Backpropagation in Practice

Week1

What is Machine Learning?

- Field of study that gives computer the ability to learn without explicity programmed
- A computer program learn from experience E with respect to some task T and some performance measure P, if its Performance of T, measured by P improves with E.
  - T: play check
  - P: probability to win
  - E: play thousands of games
- Machine learning algorithms
  - Supervised learning : teach computer how to do something
    - give the computer data (right answer) use the data to do something
    - regression problem: fit a line to predict
    - Classification: discrete valued output(0 or 1)
  - Unsupervised learning : Data has no label (clustering algorithm)
  - Reinforcement learning
  - recommender systems

Model and Cost Function

1. Model representation

- Some definations
  - Data set -> Training set
  - m = training set . size()
  - x's "input" variables
  - y's "output" variables
  - (x,y) -> a training example
  - (x(i),y(i)) -> i_{th} training example
- Process
  - training set -> Learning algorithm -> h（function as output of learning algorithms）hypothesis
  - x -> h -> y
  - $$h_\theta(x) = \theta_0 + \theta_1 x \tag{1}$$
  - θ is parameter and the function represent a line： named as **linear regression** with one variable -> **Univariate(one var) linear regression**

2. Method

- Idea : choose $\theta_0\theta_1$ that $h_\theta(x)$ is close to y for our training examples (x,y)

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

$$J(\theta_1,\theta_2)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)^2 \quad (3)$$

- J is the cost function with input θ, we need find J_{min}
  - Suppose θ_0 is 0, then draw the graph of J -> a bow shipe graph
  - for two parameters we use contour plots(figures)
    - X-axis : θ_0 Y-axis : θ_1
    - Every circle represent same value of $J(\theta_0,\theta_1)$ (等高线, 3-D 图的投影)

Parameter Learning

---

1. Algorithm to calculate J_{min} : Gradient descent(梯度下降)

- Outline
  - Start with some θ_0,θ_1 (say the two parameters are both zero)
  - Keep changing θ_0,θ_1 to reduce$J(\theta_0,\theta_1)$ until end up at a minimum

repeat until convergence {

$$\theta_j := \theta_j - \alpha\frac{\delta}{\delta\theta_j}J(\theta_0,\theta_1)\ (for\ j=0\ and\ j=1) \quad (4)$$

}

> := assignment
>
> α is learning rate : size of steps down the hill
>
> update θ_0,θ_1 at the same time (simultaneous update)

```
temp0 = theta0 - something(theta0,theta1);
temp1 = theta1 - something(theta0,theta1);
// update by old value first
theta0 = temp0;
theta1 = temp1;
```

---

2. Gradient Descent Intuition

- first suppose $\theta_0 = 0$
  - Use $\alpha\frac{\delta}{\delta\theta_j}J(\theta_0,\theta_1)$ if delta <(>) 0 then go right(left) , when =0 stop
  - α: small->slow, large->fail to converge
- Used in Linear Regression

$$\frac{\delta}{\delta\theta_j}J(\theta_0,\theta_1)=\frac{\delta}{\delta\theta_j}\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)^2=\frac{\delta}{\delta\theta_j}\frac{1}{2m}\sum_{i=1}^{m}(\theta_0+\theta_1x^i-y^i)^2 \quad (5)$$

$$j = 0 : \frac{\delta}{\delta\theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2 \tag{6}$$

$$j = 1 : \frac{\delta}{\delta\theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2 * x^i \tag{7}$$

    - J is Convex function
- **"Batch" Gradient Descent**: Batch is each step of gradient descent uses **all** the training examples

---

3. Linear Algebra

- Matrix in Matlab/Octave

```
% The ; denotes we are going back to a new row.
A = [1, 2, 3; 4, 5, 6; 7, 8, 9; 10, 11, 12]

% Initialize a vector
v = [1;2;3]

% Get the dimension of the matrix A where m = rows and n = columns
[m,n] = size(A)

% You could also store it this way
dim_A = size(A)

% Get the dimension of the vector v
dim_v = size(v)

% Now let's index into the 2nd row 3rd column of matrix A
A_23 = A(2,3)
% Initialize matrix A and B
A = [1, 2, 4; 5, 3, 2]
B = [1, 3, 4; 1, 1, 1]

% Initialize constant s
s = 2

% See how element-wise addition works
add_AB = A + B

% See how element-wise subtraction works
sub_AB = A - B

% See how scalar multiplication works
mult_As = A * s

% Divide A by s
div_As = A / s

% What happens if we have a Matrix + scalar?
add_As = A + s

% The ; denotes we are going back to a new row.
A = [1, 2, 3; 4, 5, 6; 7, 8, 9; 10, 11, 12]

```

```
% Initialize a vector
v = [1;2;3]

% Get the dimension of the matrix A where m = rows and n = columns
[m,n] = size(A)

% You could also store it this way
dim_A = size(A)

% Get the dimension of the vector v
dim_v = size(v)

% Now let's index into the 2nd row 3rd column of matrix A
A_23 = A(2,3)

B=A*v
% Initialize a 3 by 2 matrix
A = [1, 2; 3, 4;5, 6]

% Initialize a 2 by 1 matrix
B = [1; 2]

% We expect a resulting matrix of (3 by 2)*(2 by 1) = (3 by 1)
mult_AB = A*B

% Make sure you understand why we got that result

% Initialize random matrices A and B
A = [1,2;4,5]
B = [1,1;0,2]

% Initialize a 2 by 2 identity matrix
I = eye(2)

% The above notation is the same as I = [1,0;0,1]

% What happens when we multiply I*A ?
IA = I*A

% How about A*I ?
AI = A*I

% Compute A*B
AB = A*B

% Is it equal to B*A?
BA = B*A

% Note that IA = AI but AB != BA

%transpose and invert
A_trans = A'
A_invA = inv(A)
```

- for $\vec{x} = [x_1; x_2; \ldots; x_n]$, we use matrix to find θ_0, θ_1
  - m represents number of sample

$$\begin{bmatrix} 1 & w_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{8}$$

Week 2

1. Multivariate Linear Regression

(1) Multiple Features

- Features -> $\{x_1, x_2, \ldots, x_n\}$ -> y -> Price
- Notation:
  - n = number of features
  - $x^{(i)}$ = input (features) of i^{th} training example (n dimentional vector) : 第i组sample
  - $x_j^{(i)}$ = value of feature j in i^{th} training example： 第j个feature
- Hypothesis:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \theta^T X \tag{9}$$

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

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \theta^T X \tag{10}$$

$$J(\theta) = J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2 \tag{11}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) \cdot x_j^{(i)} = \ (for\ j \in [0, n]) \tag{12}$$

(2) Feature Scaling

- When the two parameter have a great difference (say size(0-2000 $feet^2$),number of bedrooms(1-5)) The contour plots may be **tall and slim** makes it difficult to find the local minimum
- Feature Scaling
  - Use

$$x_1 = \frac{size(0 - 2000 feet^2)}{2000}, x_2 = \frac{number\ of\ bedrooms}{5}\ (x_1, x_2 \in [0, 1]) \tag{13}$$

  - Only needed when the scale is slightly different
- Mean Normalization
  - Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (do not apply to $x_0 = 1$)

$$x_1 = \frac{size - 1000}{2000}, x_2 = \frac{no.\ bedrooms - 2}{5} \tag{14}$$

$$x_i = \frac{x_i - \mu_i}{s_i}\ (\mu\ is\ average, s\ is\ range = max - min) \tag{15}$$

(3) Learning Rate α

- two point
  - ***Debugging*** : How to make sure gradient descent is working correctly
  - How to choose α
- Debugging use a new function
  - Y-axis : $minJ(\theta)$ X-axis: No. of iterations
  - $minJ(\theta)$ should decrease after every iteration

---

(4) Features and Polynomial Regression

- 2 Points
  - choice of features
  - Polynomial Regression
- Defining new features

$$h_\theta(x) = \theta_0 + \theta_1 * frontage + \theta_2 * depth \tag{16}$$

$$frongtage * depth = Area = x \tag{17}$$

$$h_\theta(x) = \theta_0 + \theta_1 x \tag{18}$$

- Polynomial regression

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \tag{19}$$

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 \sqrt{x} \tag{20}$$

> use cubic function because house price will not go down as size increase

We can simply use

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \tag{21}$$

$$x_1 = size,\ x_2 = size^2,\ x_3 = size^3 \tag{22}$$

Don't forget Feature scaling

---

2. Computing Parameters Analytically

---

(1) Normal Equation

- Solve for $minJ(\theta)$ at one step analytically
- Intuition: if 1D ($\theta \in R$)

$$J(\theta) = a\theta^2 + b\theta + c \tag{23}$$

$$\frac{\delta}{\delta\theta} J(\theta) = \ldots -> 0 \tag{24}$$

But it is too difficult to implement

$$\theta = (X^T X)^{-1} X^T \vec{y} \tag{25}$$

  - Explain

    X is the training set m*n (m sets of data,n features) $\vec{y}$ is the right answer

$$X\vec{\theta} = \vec{y} \text{ is not always solvable} \tag{26}$$

$$We \text{ find a similar answer } \theta \text{ such that } X\hat{\theta} = \vec{p} \tag{27}$$

$$\vec{p} \text{ is a projection of } \vec{y} \text{ on the column space of } X, \text{ to make it solvable} \tag{28}$$

$$\hat{\theta} = X^{-1}\vec{p} \tag{29}$$

$$\vec{p} \text{ and column space of } X \text{ are orthogonal to } \vec{y} - \vec{p} \Longrightarrow X^T \cdot (\vec{y} - \vec{p}) = \vec{0} \Longrightarrow X^T \cdot (\vec{y} - X\hat{\theta}) = 0 \tag{30}$$

$$\hat{\theta} = (X^T X)^{-1} X^T \vec{y} \tag{31}$$

$$\vec{p} = X\hat{\theta} = X(X^T X)^{-1} X^T \vec{y} = P\vec{y} \, (P \text{ is the projection matrix}) \tag{32}$$

$$P = X(X^T X)^{-1} X^T \tag{33}$$

  - $P^T = P, P^2 = P$
  - P * Anyone $\in$ The column space of X
- Implementation (no need feature scaling)

```
pinv(x'*x)*x'*y
```

- Pros
  - No need α
  - No need iterate
- Cons : if n is large(features), it is slow [O(n^3) while gradient descent is O(kn^2) fit for(n<10000)]

---

(2) Normal Equation Noninvertibility

- In octave
  - `pinv()` : pseudo invert (has value **even not invertable**)
  - `inv()`: real invert
- XX^T is not invertable
  - Redundant features (dependent)
  - To many featrues(e.g. m<=n)

    > Delete some features, or use **regularization**

---

3. Ocatave/Matlab Tutorial

---

(1) Basic operations

```
1~=2    %not equal sign
xor(1,0)  %yihuo
1&&0    %AND
1 || 0    %or
PS1("somestring");  %change the string in front of command line
x = 3;    %semicolon supressing output(not print the answer)
a=pi
disp(a);  %display
```

```
disp(sprintf('2 decimals: %0.2f',a))  %display in two decimal place
v=1:0.1:2 %quick way to get row vector from 1 to 2 every column increase 0.1
v=1:6   %from 1 to 6 increase 1 by default
w=ones(2,3) %all entries is 1
w=zero(2,3)
w=rand(2,3)
w=randn(1,3)%Gaussian distribution
w = -6+sqrt(10)*(randn(1,10000));
hist(w)   %draw histogram
hist(w,50)  %histogram with 50 bins
eye(4)    %4by4 identity matrix
help eye
help rand %show the detail of method

```

(2) Moving Data Around

```
sz=size(A)  % show size of matrix sz is also a matrix
size(A,1) % row of A
size(A,2) % column of A
length(A) % longer dimention max(m,n)
pwd     % current path you are in
cd      % change the pass
ls      % list the file
load featureX.dat %load the file
load('featuresX.dat')
who     % show all the variables
whos    % show in detail
clear featuresX %delete the variable
v = priceY(1:10)  % store first 10 columns of priceY in the v
save hello.mat v  %save v in the new file
save hello.txt v-ascii  %save as txt
A(3,2)    %i=3,j=2 elements of A
A(2,:)    %: means the whole row or column
A([1 3],:)  %all 1 rows and 3 rows
A(:,2)=[10;11;12] %assign
A=[A,[100;101;102]] %add a column
A(:)        %put all elements of A into a single vector

```

(3) Computing on Data

```
A.*B  %get matrix c that c(i,j)=a(i,j)*b(i,j)
v = [1;2;3]
1./v  %get matrix c that c(i,j) = 1/a(i,j)
log(v)
exp(v)
abs(v)  % similar
% add 1 to every entries
v+1
v+ones(length(v),1)
A'  % A transpose
A = [1 15 2 0.5]
val = max(A)  % biggest number
max(A,[],1)

```

```
max(A,[],2)   % row and col max respectively
[val,ind]=max(a)% assign the position to ind
A < 3     % return true or false as a matrix
find(A<3)   % return all true entrys
A = magic(3)  % all the rows and columns are diagonals sum to a same number
[r,c]=find(A>=7)% A(ri,ci)>=7
sum(A)      % add all colmn or row for m = 1 matrix
sum(A,1)    % sum all the row
sum(A,2)    % sum all the col
sum(sum(A.*eyes(size(A),1)))  % sum the diagnal of one side
sum(sum(A.*flipud(eyes(size(A),1))) % sum the diagnal of other side
prod(A)     % multiple
floor(A)
ceil(A)     %for all entries


```

(4) Plotting Data

```
plot(t,y1)    % t-x axis y-y axis
plot(t,y2,'r')  % print y2 in a different color
xlabel('time')
ylabel('value')
legend('sin','cos')
title('my plot')
cd 'your path'; print -dpng 'myPlot.png'

% plot in different pictures
figure(1); plot(t,y1);
figure(2); plot(t,y2);

% Divides plot a 1*2 gird, access first element
subplot(1,2,1);
plot(t,y1);
subplot(1,2,2);
plot(t,y2);
axis([0.5 1 -1 1])  %change range of x and y
clf;        % clear figure
A = magic(5);
imagesc(A);     %represent in different colors
% present the color bar and use black white style
imagesc(A),colorvar,colormap gray;

```

(5) Control Statement : `for` `while` `if`

```
% for loop
v = zeros(1,10)
for i = 1:10,
  v(i) = 2*i;
end;
% or
indicies = 1:10;
for i = indicies,
  disp(i);
```

```
end;
% while loop
i = 1;
while i<=5,
  v(i++) = 100;
end;
% or
i = 1;
while true,
  v(i++) = 99;
  if i == 6,
    break;
  end;
end;
% if - else
if v(1) == 1,
  disp('The value is one');
elseif v(1) == 2,
  disp('The value is two');
else
  disp('The value is not one or two');
end;


```

- Create Function
  - Create a new file named as 'the name of function.m'

```
function y = squareThisNumber(x)
y = x^2;
% use
addpath('path of the function')
squareThisNumber(5)

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

```

  - $J(\theta)$ in the exmaple

```
% use the function
>> x = [1 1;1 2;1 3]
>> y = [1;2;3]
>> theta = [0;1];

```

```
% the function file 'costFunctionJ'
function J = costFunctionJ(X,y,theta)
m = size(X,1)
predictions = X*theta;        % m*2 . 2*1
sqrErrors = (predictions-y).^2;   % turn every entries into its square

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

```

(6) Vectorization

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j = \theta^T X \tag{34}$$

- silly version

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

- Vectorized version

```
prediction = theta'*X;
```

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) \cdot x_j^{(i)} = \ (for\ j \in [0, n]) \tag{35}$$

- Implementation

```
prediction = X*theta;   % (m,1)
Error = prediction - Y; % (m,1)
delta = (1/m)*X'*Error;   % (n+1,m)*(m,1)=(n+1,1)
theta=theta-alpha*delta;

theta = theta-alpha*X'*(X*theta-Y)
```

(7) Method learnt in the assignment

```
data = load('filename');  % open the file and store it as a matrix
linspace - 生成线性间距向量

    此 MATLAB 函数 返回包含 x1 和 x2 之间的 100 个等间距点的行向量。

    y = linspace(x1,x2)
    y = linspace(x1,x2,n)


logspace - 生成对数间距向量

    此 MATLAB 函数 生成一个由在 10^a 和 10^b (10 的 N 次幂) 之间的 50 个对数间距点组成的行向量 y。logspace
    函数对于创建频率向量特别有用。该函数是 linspace 和:运算符的对数等价函数。

    y = logspace(a,b)
    y = logspace(a,b,n)
```

```
    y = logspace(a,pi)
```

- plot

```
% two axis of 100 grids from (-10,10) (-1,4) respectively
theta0_vals = linspace(-10,10,100);
theta1_vals = linspace(-1,4,100);
% draw the graph
J_vals = zeros(length(theta0_vals),length(theta1_vals));
% Fill
for i = 1:length(theta0_vals)
  for j = 1:length(theta1_vals)
    t = [theta0_vals(i);theta1_vals(j)];  %a pair of theta
    J_vals(i,j) = computeCost(x,y,t);
  end
end
% Surface plot
figure;
surf(theta0_vals, theta1_vals, J_vals)
xlabel('\theta_0'); ylabel('\theta_1');

% Contour plot
figure;
% Plot J_vals as 15 contours spaced logarithmically between 0.01 and 100
contour(theta0_vals, theta1_vals, J_vals, logspace(-2, 3, 20))
xlabel('\theta_0'); ylabel('\theta_1');
hold on;
plot(theta(1), theta(2), 'rx', 'MarkerSize', 10, 'LineWidth', 2);
```

- Normalization for multiple features

```
% normalization : after normalize you need to store the mean and standard(when predict new value we need to change its scale again)
mean(X);  % return average of each column
std(X);   % return standard deviation of each column
repmat(A,m,n) % treat A as an element create a matrix m by n of A as a entry
% the code
X_norm = X;
mu = zeros(1, size(X, 2));
sigma = zeros(1, size(X, 2));
mu = mean(X)
X_norm = X_norm-repmat(mu,size(X,1),1)
sigma = std(X)
X_norm = X_norm./repmat(sigma,size(X,1),1)
```

- Gradient Descent for multiple features
  - cost function

```
function J = costFunctionJ(X,y,theta);
J = 1/(2*size(X,1))*((X*theta-y)'*(X*theta-y));
```

  - Gradient iteration

```
function [theta, J_history] = gradientDescentMulti(X, y, theta, alpha, num_iters);
% Create a column vector of J
J_history = zeros(num_iters, 1);
% do gradient descent
theta = theta - (1/length(y))*alpha*X'*(X*theta-y);
% compute new J for debug
J_history(iter) = computeCostMulti(X, y, theta);
end
```

- Normal Equation

```
function [theta] = normalEqn(X, y)
theta = zeros(size(X, 2), 1);
theta = pinv(X'*X)*X'*y;
end
```

Week 3

1. Classification -> logistic regression

- Email: spam/not spam
- Online Transactions: Fraudulent (Yes/ No)
- Tumor: Malignant / Benign

  $y \in \{0, 1\}$ 0: "Negative Class"; 1: "Positive Class"
- Why not use linear regression? : It is easily affected by noisy points

2. Hypothesis Representation (function used)

(1) Logistic Regression Model

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

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{36}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{37}$$

g is Sigmoid (Logistic) function

(2) Interpretation of Hpothesis Output

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

$h_\theta(x) = 0.7$ => Tell patient that 70% chance of tumor being malignant

$$P(y = 0 \mid x; \theta) = 1 - P(y = 1 \mid x; \theta) \tag{38}$$

(3) Decision Boundary

- property of the g
  - Suppose predict "y=1" if $h_\theta(x) \geq 0.5$
  - Predict "y=0" if $h_\theta(x) < 0.5$
  - $g(z) \geq 0.5$ when $z \geq 0$ ($\theta^T x \geq 0$)
- Decision Boundary

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \tag{39}$$

$$\theta = [-3, 1, 1] \tag{40}$$

Predict "y=1" if $-3 + x_1 + x_2 \geq 0$ $x_1 + x_2 = 3$ is a line (Decision boundary)

Cut the $x_1 x_2$ plane into two pieces

- What if the training set is more complex (cannot use a line) : Add polynomial parameter

---

3. Logistic Regression Model

(1) Cost function

- Problem

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

  m examples: $x \in [x_0, x_1, \ \dots \ x_n]$ $x_0 = 1, y \in \{0, 1\}$

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

  How to choose θ
- Logistic regression cost function

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) \ if \ y = 1 \\ -log(1 - h_\theta(x)) \ if \ y = 0 \end{cases} \tag{41}$$

  - If $y = 1, \ h_\theta(x) = 1$ then Cost = 0
  - But as $h_\theta(z) \to 0$,$Cost \to \infty$
  - 如果猜对了， 奖励这个算法， 猜错了则增大cost

(2) Simplified cost function and gradient descent

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}, y^{(i)}) \tag{42}$$

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) \ if \ y = 1 \\ -log(1 - h_\theta(x)) \ if \ y = 0 \end{cases} \tag{43}$$

Note y =0 or 1 always

$$Cost(h_\theta(x), y) = -ylog(h_\theta(x)) - (1 - y)log(1 - h_\theta(x))) \tag{44}$$

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))] \tag{45}$$

- Gradient Descent

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

  Repeat{

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

}

$$\frac{\delta}{\delta\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \tag{46}$$

Which is similar to linear regression

(3) Advanced optimization

Given θ we have code that can compute $J(\theta)$ and $\frac{\delta}{\delta\theta_j}J(\theta)$

- Optimization algorithm
  - Conjugate gradient
  - BFGS
  - L-BFGS
  - Advantages
    - No need to manually pick α
    - Often faster than gradient descent
  - Disadvantages
    - More complex
- Example (Use derivative)

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

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

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

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

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

```
% option is a data structure which can store your options
options = optimset('GradObj','on','MaxIter','100');
% set gradient objective parameter to on, and set the maximum number of interation
initialTheta = zero(2,1);
% @represent a pointer to costFunction
[optTheta,functionVal,exitFlag] = fminunc(@costFunction,initialTheta,options);
```

---

4. Multi-class classification: one-vs-all

- Multiclass classification
  - Email folding/tagging: Work, friend, family, hobby
  - Medical: Not ill, cold, flu
  - Weather: Sunny,Cloudy,Rain,Snow
- One-vs-all (one-vs-rest) : use more binary classification

$$h_\theta^{(i)}(x) = P(y = i|x;\theta)(i = 1, 2, 3) for\ three\ cluster \tag{47}$$

- On a new x, to make a prediction, pick the class i that maximizes $max\ h_\theta^{(i)}(x)$

5. Solving the problem of overfitting

(1) The problem of overfitting

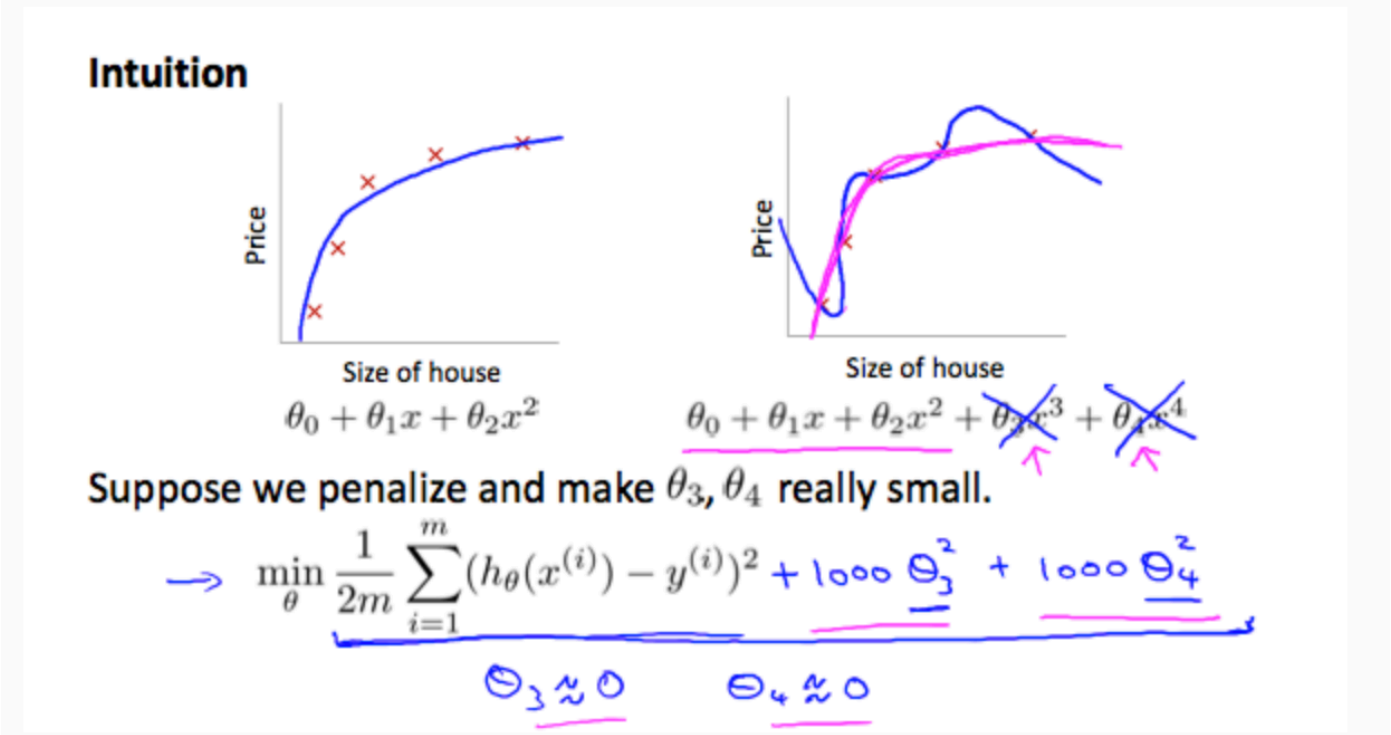
- Three kinds of fitting
  - Cannot fit very well => "Underfit" or "High bias" (偏见，偏差)
  - Fit well => "Just right"
  - 100% fitting (Although fitting, but the curve is meaningless) => "Overfit" "High variance"
- Overfitting : If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples)
- Addressing overfitting:
  - draw the graph
  - Options

    (1) Reduce number of features
    - Manually select which features to keep
    - Model selection algorithm

    (2) Regularization
    - Keep all the features, but reduce magnitude/values of parameters θ_j
    - Works well when we have a lot of features, each of which contributes a bit to predicting y .

(2) Cost function

- modify the cost function by add quadratic of overfitting θ , and recalculate the minimum, to make them contribute less than before



- Regularization : small value for parameters $\theta_0, \theta_1, \ldots, \theta_n$
  - "Simpler" hypothesis
  - Less prone to overfitting
  - In real use, we don't know to shrink which parameter => shrink all parameters (λ is regularization parameter)

$$J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} {\theta_j}^2] \tag{48}$$

- If λ is too large => cause underfitting (i.e. $h_\theta(x) = \theta_0$)

(2) regularized linear regression

- Gradient descent (with regularization) => treat θ_0 differently

  Repeat{

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

  $\theta_j := \theta_j - \alpha\frac{1}{m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \lambda\theta_j](j \neq 0)$

  Or $\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}(j \neq 0)((1 - \alpha\frac{\lambda}{m})\ is\ always\ less\ than\ 1)$

  }

- Normal equation

$$\theta = (X^TX + \lambda L)^{-1}X^Ty \tag{49}$$

  Where L is a matrix

$$\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{50}$$

  - Use when the matrix X^TX is singular

(3) regularized logistic regression

- Cost function

$$J(\theta) = -[\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(x^{(i)})) + (1 - y^{(i)})log(1 - h_\theta(x^{(i)}))]] + \frac{\lambda}{2m}\sum_{j=1}^{n}{\theta_j}^2(j = 1, 2, \ldots, n) \tag{51}$$

- Advanced optimization

Week4

1. Motivations

- There may be a lot of features => it is not efficient to use all feature
- Neural network => build brain

---

2. Neural Networks

(1) Model Representation I

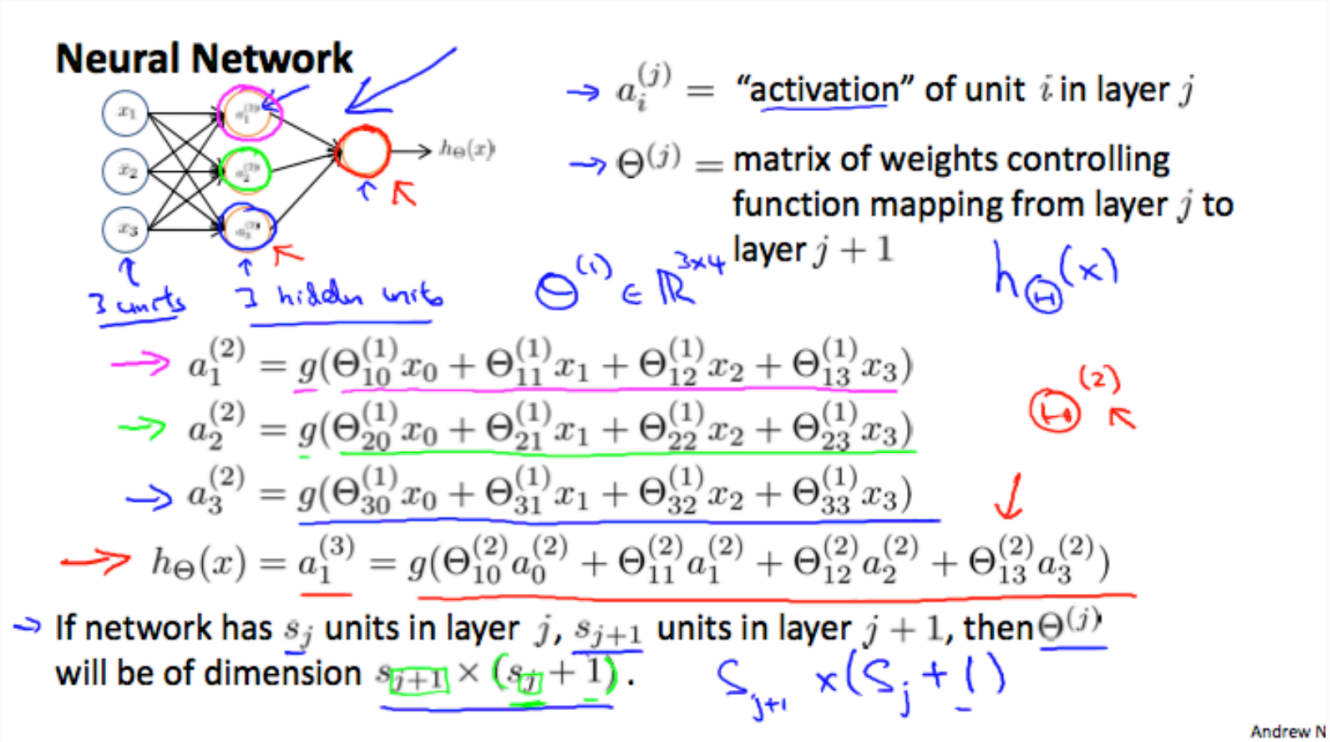
- Neuron model : logistic unit

$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_m \end{bmatrix} => [\ ] => h_\theta(x) \tag{52}$$

- Sigmoid function => activation function $g(z)$
- θ => weights or parameters
- We can have intermediate layers of nodes between the input and output layers called the "hidden layers."

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

- Neuron network



  - dimension of Θ : $s_{j+1} * (s_j + 1)$ where s_n is the number of variables in nth layer, 1 is the constant column
  - $A^{(j)} * Theta^{(j)T} = A^{(j+1)}$
- More specific description
  - Input layer:
    - W matrix: Each row of weight represent a neural, column represent features accept
    - X matrix : Each row represent a feature, each column represent a sample
    - WX+B = Z: Each row of Z represent a neural, each column of z represent a sample(Neural*feature.feature*sample), where B is bias
    - A = g(Z), activate output

- Hidden layer:
  - W matrix: Each row represent a neural of this layer, each column represent a neural of last layer, the first column represent the bias of last layer
  - WA(last)+B = Z: Each row of Z represent a neural in this layer, each column of z represent a sample(Neural* (last Neural+1).(last Neural+1)*sample)
- Output layer:
  - similar to Hidden layer, instead each row represent an output for this sample

(2) Model representation II

- Forward propagation: Vectorized implementation

$$z^{(j+1)} = \Theta^{(j)} a^{(j)} \tag{53}$$

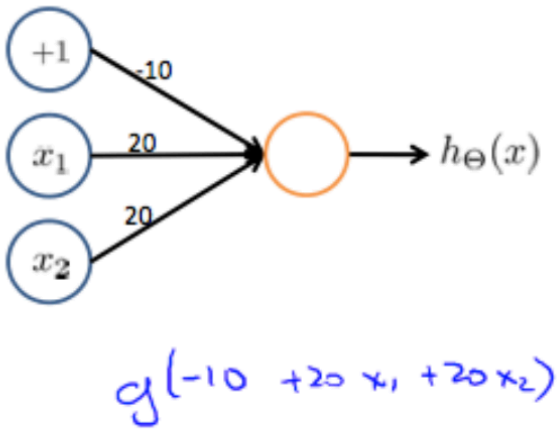
$$h_\theta(x) = a^{(j+1)} = g(z^{(j+1)}) \tag{54}$$

---

3. Applications

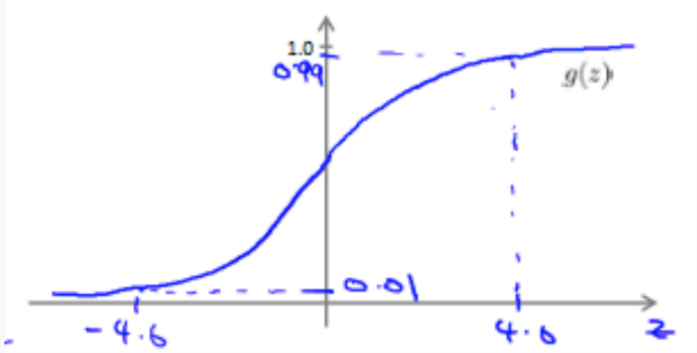
(1) Example and Intuitions I

Example: OR function

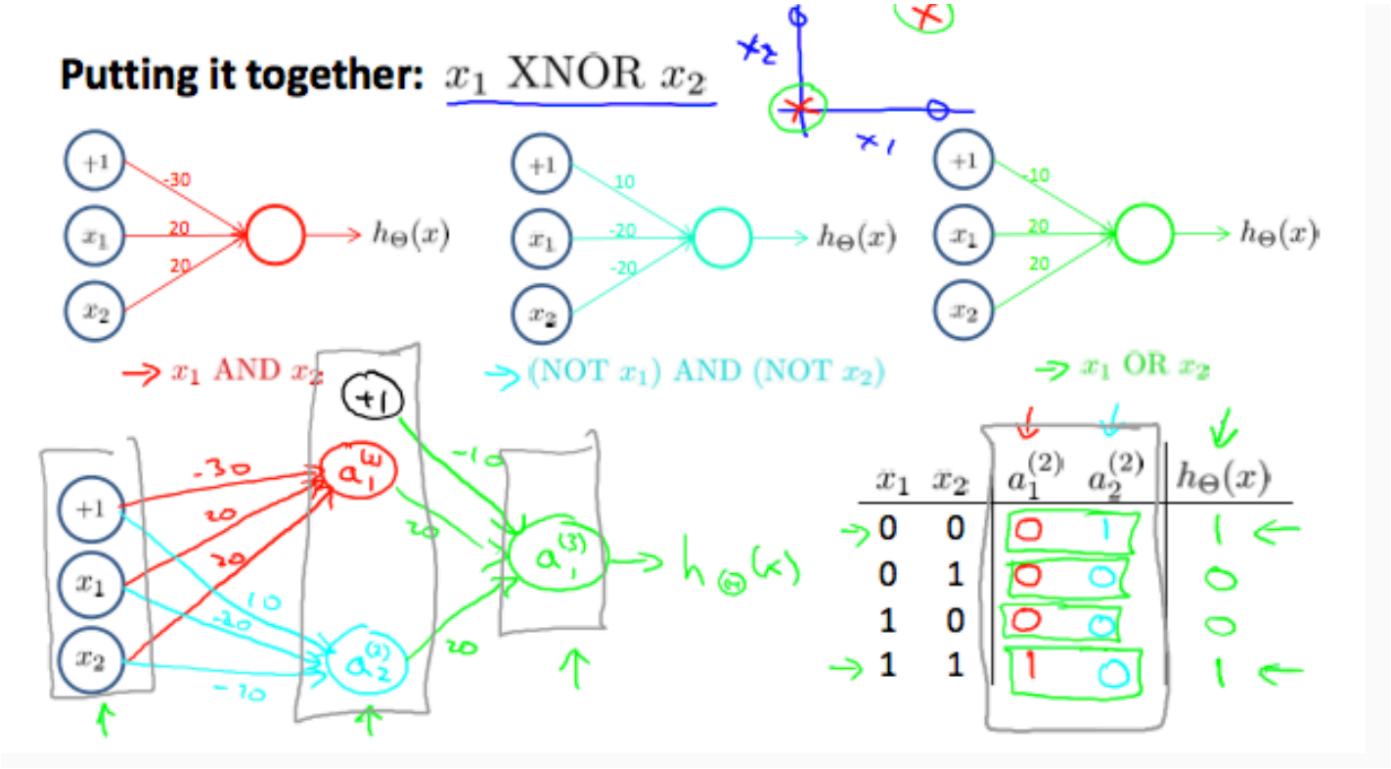


x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	≈ 1
1	1	≈ 1

Where g(z) is the following:



(2) Example and Intuitions II



Week 5

1. Cost Function and Backpropagation

(1) Cost function

- Cost functions
  - Denotes
    - L = total number of layers in the network
    - S_l = number of units (not counting bias unit) in layer l
    - K = number of output units/classes (channels)
  - Logistic regression

$$J(\theta) = -[\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(x^{(i)})) + (1-y^{(i)})log(1-h_\theta(x^{(i)}))]] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j{}^2 (j=1,2,\dots,n) \quad (55)$$

  - Neural network (for multi-class problem there are K clusters) $h_\Theta(x) \in R^K$ $(h_\Theta(x))_i = i^{th}$ *output*
    - The first part add the cost of all the m-by-K results
    - The second part add all the theta used (every column, row, layer (L))
    - regularization doesn't include the 0^{th} weight

$$J(\Theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})log(1-h_\Theta(x^{(i)}))_k] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}(\Theta_{ji}^l)^2 \quad (56)$$

(2) Backpropagation Algorithm

- Proof the backpropagation

  权重和偏执项 $W^{[3]} = \begin{bmatrix} w_{11}^{[3]} & w_{12}^{[3]} & w_{13}^{[3]} \\ w_{21}^{[3]} & w_{22}^{[3]} & w_{23}^{[3]} \end{bmatrix}_{2\times 3}$ ， $B^{[3]} = \begin{bmatrix} b_1^{[3]} \\ b_2^{[3]} \end{bmatrix}_{2\times 1}$

  该层的线性计算

  $Z^{[3]} = W^{[3]}A^{[2]} + B^{[3]} = \begin{bmatrix} w_{11}^{[3]}a_1^{[2](1)} + w_{12}^{[3]}a_2^{[2](1)} + w_{13}^{[3]}a_3^{[2](1)} + b_1^{[3]} & w_{11}^{[3]}a_1^{[2](2)} + w_{12}^{[3]}a_2^{[2](2)} + w_{13}^{[3]}a_3^{[2](2)} + b_1^{[3]} \\ w_{21}^{[3]}a_1^{[2](1)} + w_{22}^{[3]}a_2^{[2](1)} + w_{23}^{[3]}a_3^{[2](1)} + b_1^{[3]} & w_{21}^{[3]}a_1^{[2](2)} + w_{22}^{[3]}a_2^{[2](2)} + w_{23}^{[3]}a_3^{[2](2)} + b_1^{[3]} \end{bmatrix}_{2\times 2}$

  记作 $Z^{[3]} = \begin{bmatrix} z_1^{[3](1)} & z_1^{[3](2)} \\ z_2^{[3](1)} & z_2^{[3](2)} \end{bmatrix}_{2\times 2}$

  激活输出 $A^{[3]} = \sigma(Z^{[3]}) = \begin{bmatrix} a_1^{[3](1)} & a_1^{[3](2)} \\ a_2^{[3](1)} & a_2^{[3](2)} \end{bmatrix}_{2\times 2} = \begin{bmatrix} \sigma(z_1^{[3](1)}) & \sigma(z_1^{[3](2)}) \\ \sigma(z_2^{[3](1)}) & \sigma(z_2^{[3](2)}) \end{bmatrix}_{2\times 2}$

  输出结果是 $Y = \begin{bmatrix} y_1^{(1)} & y_1^{(2)} \\ y_2^{(1)} & y_2^{(2)} \end{bmatrix}_{2\times 2} = A^{[3]}$ ，

  对应的真实标签值记是 $\tilde{Y} = \begin{bmatrix} \tilde{y_1}^{(1)} & \tilde{y_1}^{(2)} \\ \tilde{y_2}^{(1)} & \tilde{y_2}^{(2)} \end{bmatrix}_{2\times 2}$

  设每个神经元的激活函数为最常用的Sigmoid函数： $\sigma(z) = \dfrac{1}{1+e^{-z}}$

  - each number in a "()" represent the sample id
  -

$$L=-(\tilde{Y}log(Y)+(1-\tilde{Y})log(1-Y))=\begin{bmatrix}-(\tilde{y_1}^{(1)}log(y_1^{(1)})+(1-\tilde{y_1}^{(1)})log(1-y_1^{(1)}) & -(\tilde{y_1}^{(2)}log(y_1^{(2)})+(1-\tilde{y_1}^{(2)})log(1-y_1^{(2)}))\\ -(\tilde{y_2}^{(1)}log(y_2^{(1)})+(1-\tilde{y_2}^{(1)})log(1-y_2^{(1)}) & -(\tilde{y_2}^{(2)}log(y_2^{(2)})+(1-\tilde{y_2}^{(2)})log(1-y_2^{(2)}))\end{bmatrix}_{2\times 2}$$

简记为：

$$L=\begin{bmatrix}l_1^{(1)} & l_1^{(2)}\\ l_2^{(1)} & l_2^{(2)}\end{bmatrix}$$

则由简单的链导法则可有：

$$\frac{dL}{dZ^{[3]}}=\frac{dL}{dA^{[3]}}\frac{dA^{[3]}}{dZ^{[3]}}=\begin{bmatrix}\frac{dl_1^{(1)}}{da_1^{[3](1)}}\frac{da_1^{[3](1)}}{dz_1^{[3](1)}} & \frac{dl_1^{(2)}}{da_1^{[3](2)}}\frac{da_1^{[3](2)}}{dz_1^{[3](2)}}\\ \frac{dl_2^{(1)}}{da_2^{[3](1)}}\frac{da_2^{[3](1)}}{dz_2^{[3](1)}} & \frac{dl_2^{(2)}}{da_2^{[3](2)}}\frac{da_2^{[3](2)}}{dz_2^{[3](2)}}\end{bmatrix}_{2\times 2}=\begin{bmatrix}dz_1^{[3](1)} & dz_1^{[3](2)}\\ dz_2^{[3](1)} & dz_2^{[3](2)}\end{bmatrix}_{2\times 2}$$

记 $\frac{dL}{dZ^{[3]}}=dZ^{[3]}$ ， $\frac{dL}{dA^{[3]}}=dA^{[3]}$ ， $\frac{dA}{dZ^{[3]}}=\sigma^{'}(Z^{[3]})$ ，则，显然有：

$dZ^{[3]}=dA^{[3]}*\sigma^{'}(Z^{[3]})$ ，即：

$$\begin{bmatrix}dz_1^{[3](1)} & dz_1^{[3](2)}\\ dz_2^{[3](1)} & dz_2^{[3](2)}\end{bmatrix}_{2\times 2}=\begin{bmatrix}da_1^{[3](1)} & da_1^{[3](2)}\\ da_2^{[3](1)} & da_2^{[3](2)}\end{bmatrix}_{2\times 2}*\begin{bmatrix}d\sigma(dz_1^{[3](1)}) & d\sigma(dz_1^{[3](2)})\\ d\sigma(dz_2^{[3](1)}) & d\sigma(dz_2^{[3](2)})\end{bmatrix}_{2\times 2}$$

其中的 $*$ 表示逐元素相乘（比如MatLAB里的A.*B，Python里的矩阵 A*B）。这一块涉及的函数求导就不赘述了，然后就可很容易计算出来下面的结果：

$dZ^{[3]} = A^{[3]} - \tilde{Y}$ ，

$$\frac{dL}{dW^{[3]}} = \frac{dL}{dZ^{[3]}}\frac{dZ^{[3]}}{dW^{[3]}} = \begin{bmatrix} \frac{dl_1^{(1)}}{dw_{11}^{[3]}} + \frac{dl_1^{(2)}}{dw_{11}^{[3]}} & \frac{dl_1^{(1)}}{dw_{12}^{[3]}} + \frac{dl_1^{(2)}}{dw_{12}^{[3]}} & \frac{dl_1^{(1)}}{dw_{13}^{[3]}} + \frac{dl_1^{(2)}}{dw_{13}^{[3]}} \\ \frac{dl_1^{(1)}}{dw_{21}^{[3]}} + \frac{dl_1^{(2)}}{dw_{21}^{[3]}} & \frac{dl_1^{(1)}}{dw_{22}^{[3]}} + \frac{dl_1^{(2)}}{dw_{22}^{[3]}} & \frac{dl_1^{(1)}}{dw_{23}^{[3]}} + \frac{dl_1^{(2)}}{dw_{23}^{[3]}} \end{bmatrix}_{2\times 3}$$

$$dW^{[3]} = \begin{bmatrix} \frac{dl_1^{(1)}}{dz_1^{[3](1)}}\frac{dz_1^{[3](1)}}{dw_{11}^{[3]}} + \frac{dl_1^{(2)}}{dz_1^{[3](2)}}\frac{dz_1^{[3](2)}}{dw_{11}^{[3]}} & \frac{dl_1^{(1)}}{dz_1^{[3](1)}}\frac{dz_1^{[3](1)}}{dw_{12}^{[3]}} + \frac{dl_1^{(2)}}{dz_1^{[3](2)}}\frac{dz_1^{[3](2)}}{dw_{12}^{[3]}} & \frac{dl_1^{(1)}}{dz_1^{[3](1)}}\frac{dz_1^{[3](1)}}{dw_{13}^{[3]}} + \frac{dl_1^{(2)}}{dz_1^{[3](2)}}\frac{dz_1^{[3](2)}}{dw_{13}^{[3]}} \\ \frac{dl_2^{(1)}}{dz_2^{[3](1)}}\frac{dz_2^{[3](1)}}{dw_{21}^{[3]}} + \frac{dl_2^{(2)}}{dz_2^{[3](2)}}\frac{dz_2^{[3](2)}}{dw_{21}^{[3]}} & \frac{dl_2^{(1)}}{dz_2^{[3](1)}}\frac{dz_2^{[3](1)}}{dw_{22}^{[3]}} + \frac{dl_2^{(2)}}{dz_2^{[3](2)}}\frac{dz_2^{[3](2)}}{dw_{22}^{[3]}} & \frac{dl_2^{(1)}}{dz_2^{[3](1)}}\frac{dz_2^{[3](1)}}{dw_{23}^{[3]}} + \frac{dl_2^{(2)}}{dz_2^{[3](2)}}\frac{dz_2^{[3](2)}}{dw_{23}^{[3]}} \end{bmatrix}_{2\times 3}$$

上式计算后，结果为：

$$dW^{[3]} = \begin{bmatrix} dz_1^{[3](1)}a_1^{[2](1)} + dz_1^{[3](2)}a_1^{[2](2)} & dz_1^{[3](1)}a_2^{[2](1)} + dz_1^{[3](2)}a_2^{[2](2)} & dz_1^{[3](1)}a_3^{[2](1)} + dz_1^{[3](2)}a_3^{[2](2)} \\ dz_2^{[3](1)}a_1^{[2](1)} + dz_2^{[3](2)}a_1^{[2](2)} & dz_2^{[3](1)}a_2^{[2](1)} + dz_2^{[3](2)}a_2^{[2](2)} & dz_2^{[3](1)}a_3^{[2](1)} + dz_2^{[3](2)}a_3^{[2](2)} \end{bmatrix}_{2\times 3}$$

从上面的式子可以看出，每个权重的梯度是每个样本得到的梯度之和，因此，这里都除以样本个数，求出平均梯度。整理一下，我们就得到：

$$dW^{[3]} = \frac{1}{2}\begin{bmatrix} dz_1^{[3](1)} & dz_1^{[3](2)} \\ dz_2^{[3](1)} & dz_2^{[3](2)} \end{bmatrix}_{2\times 2}\begin{bmatrix} a_1^{[2](1)} & a_2^{[2](1)} & a_3^{[2](1)} \\ a_1^{[2](2)} & a_2^{[2](2)} & a_3^{[2](2)} \end{bmatrix}_{2\times 3}$$

即 $dW^{[3]} = \frac{1}{2}dZ^{[3]}{A^{[2]}}^T$

- m = 2 thus use $\frac{1}{2}$

二、隐含层（Hidden）

权重和偏执项 $W^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} \\ w_{31}^{[2]} & w_{32}^{[2]} \end{bmatrix}_{3\times 2}, B^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ b_3^{[2]} \end{bmatrix}_{3\times 1}$

该层的线性计算

$$Z^{[2]} = W^{[2]}A^{[1]} + B^{[2]} = \begin{bmatrix} w_{11}^{[2]}a_1^{[1](1)} + w_{12}^{[2]}a_2^{[1](1)} + b_1^{[2]} & w_{11}^{[2]}a_1^{[1](2)} + w_{12}^{[2]}a_2^{[1](2)} + b_1^{[2]} \\ w_{21}^{[2]}a_1^{[1](1)} + w_{22}^{[2]}a_2^{[1](1)} + b_2^{[2]} & w_{21}^{[2]}a_1^{[1](2)} + w_{22}^{[2]}a_2^{[1](2)} + b_2^{[2]} \\ w_{31}^{[2]}a_1^{[1](1)} + w_{32}^{[2]}a_2^{[1](1)} + b_3^{[2]} & w_{31}^{[2]}a_1^{[1](2)} + w_{32}^{[2]}a_2^{[1](2)} + b_3^{[2]} \end{bmatrix}_{3\times 2}$$

记作 $Z^{[2]} = \begin{bmatrix} z_1^{[2](1)} & z_1^{[2](2)} \\ z_2^{[2](1)} & z_2^{[2](2)} \\ z_3^{[2](1)} & z_3^{[2](2)} \end{bmatrix}_{3\times 2}$

激活输出 $A^{[2]} = \sigma(Z^{[2]}) = \begin{bmatrix} a_1^{[2](1)} & a_1^{[2](2)} \\ a_2^{[2](1)} & a_2^{[2](2)} \\ a_3^{[2](1)} & a_3^{[2](2)} \end{bmatrix}_{3\times 2} = \begin{bmatrix} \sigma(z_1^{[2](1)}) & \sigma(z_1^{[2](2)}) \\ \sigma(z_2^{[2](1)}) & \sigma(z_2^{[2](2)}) \\ \sigma(z_3^{[2](1)}) & \sigma(z_3^{[2](2)}) \end{bmatrix}_{3\times 2}$

- 现在输出层的都求出来，然后就再往回一层，求隐含层的梯度，因此，中间链导需要经过 $A^{[2]}$ ：

$$dA^{[2]} = \begin{bmatrix} dz_1^{[3](1)}\frac{dz_1^{[3](1)}}{da_1^{[2](1)}} + dz_2^{[3](1)}\frac{dz_2^{[3](1)}}{da_1^{[2](1)}} & dz_1^{[3](2)}\frac{dz_1^{[3](2)}}{da_1^{[2](2)}} + dz_2^{[3](2)}\frac{dz_2^{[3](2)}}{da_1^{[2](2)}} \\ dz_2^{[3](1)}\frac{dz_2^{[3](1)}}{da_2^{[2](1)}} + dz_2^{[3](1)}\frac{dz_2^{[3](1)}}{da_2^{[2](1)}} & dz_1^{[3](2)}\frac{dz_1^{[3](2)}}{da_2^{[2](2)}} + dz_2^{[3](2)}\frac{dz_2^{[3](2)}}{da_2^{[2](2)}} \\ dz_1^{[3](1)}\frac{dz_1^{[3](1)}}{da_3^{[2](1)}} + dz_2^{[3](1)}\frac{dz_2^{[3](1)}}{da_3^{[2](1)}} & dz_1^{[3](2)}\frac{dz_1^{[3](2)}}{da_3^{[2](2)}} + dz_2^{[3](2)}\frac{dz_2^{[3](2)}}{da_3^{[2](2)}} \end{bmatrix}_{3\times 2}$$

$$dA^{[2]} = \begin{bmatrix} dz_1^{[3](1)}w_{11}^{[3]} + dz_2^{[3](1)}w_{21}^{[3]} & dz_1^{[3](2)}w_{11}^{[3]} + dz_2^{[3](2)}w_{21}^{[3]} \\ dz_2^{[3](1)}w_{12}^{[3]} + dz_2^{[3](1)}w_{22}^{[3]} & dz_1^{[3](2)}w_{12}^{[3]} + dz_2^{[3](2)}w_{22}^{[3]} \\ dz_1^{[3](1)}w_{13}^{[3]} + dz_2^{[3](1)}w_{23}^{[3]} & dz_1^{[3](2)}w_{13}^{[3]} + dz_2^{[3](2)}w_{23}^{[3]} \end{bmatrix}_{3\times 2}$$

  即：

$$dA^{[2]} = {W^{[3]}}^T dA^{[3]}$$

  然后就可以推导隐含层的梯度了，过程和上面是一样的，所以直接给出结果：

$$dW^{[2]} = \frac{1}{2} dZ^{[2]} {A^{[1]}}^T$$

$$dB^{[2]} = \frac{1}{2} sum(dZ^{[2]}, axis = 1)$$

- 然后更新权重即可：

$$W^{[i]} = W^{[i]} + \eta dW^{[i]}$$
$$B^{[i]} = B^{[i]} + \eta dB^{[i]}$$
$$i = 1, 2, 3$$

  最后把上面的推导总结一下，对于第 l 层：

$$Input: dA^{[l]}, A^{[l-1]}, W^{[l]}, Z^{[l]}$$

$$\cdots\cdots \text{ 若该层是输出层，则 } dZ^{[l]} = A^{[l]} - \tilde{Y} \text{ ，否则 } dZ^{[l]} = dA^{[l]} * \sigma^{'}(Z^{[l]})$$

$$\cdots\cdots dW^{[l]} = \frac{1}{N} dZ^{[l]} {A^{[l-1]}}^T$$

$$\cdots\cdots dB^{[l]} = \frac{1}{N} sum(dZ^{[l]}, axis = 1)$$

$$\cdots\cdots dA^{[l-1]} = {W^{[l]}}^T dZ^{[l]}$$

$$Output: dA^{[l-1]}, dW^{[l]}, dB^{[l]}$$

- Gradient computation

- we need : $min_\Theta J(\Theta)$ and $\frac{\delta}{\delta\Theta_{ij}^{(l)}}J(\Theta)$
- Given one training example (x, y)
- Intuition: $\delta_j^{(l)}$ = "errors" of node j in layer l = $\frac{\delta_j^{(l)} J}{\delta Z^{(l)}}$

$$\delta_j^{(4)} = a_j^{(4)} - y_j \tag{57}$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* (a^{(3)} .* (1 - a^{(3)})) \tag{58}$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* (a^{(2)} .* (1 - a^{(2)})) \tag{59}$$

$$\frac{\delta}{\delta\Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \tag{60}$$

- Backpropagation algorithm
  - Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$
  - Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)

    For i=1 to m

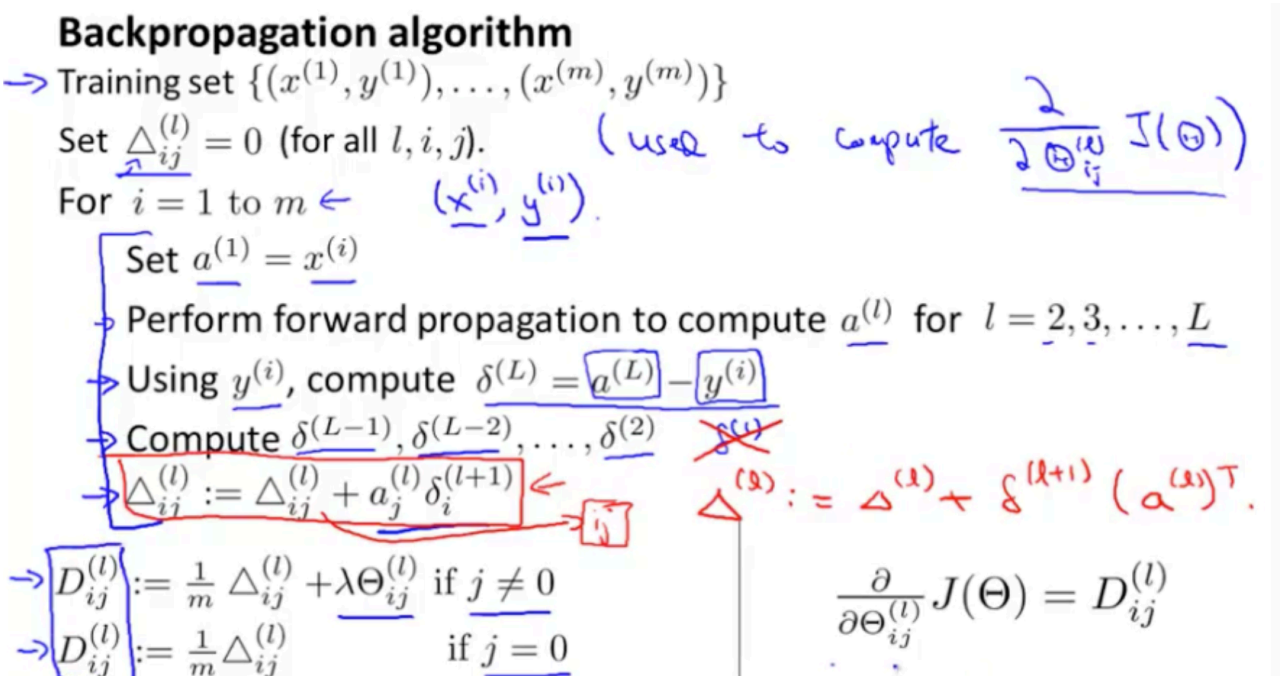
    Set $a^{(1)} = x^{(i)}$

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

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

    Compute $\delta^{(L-1)}, \delta^{(L-2)}, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$

    $\Delta_{(ij)}^{(l)} += a_j^{(l)} \delta_i^{(l+1)}$



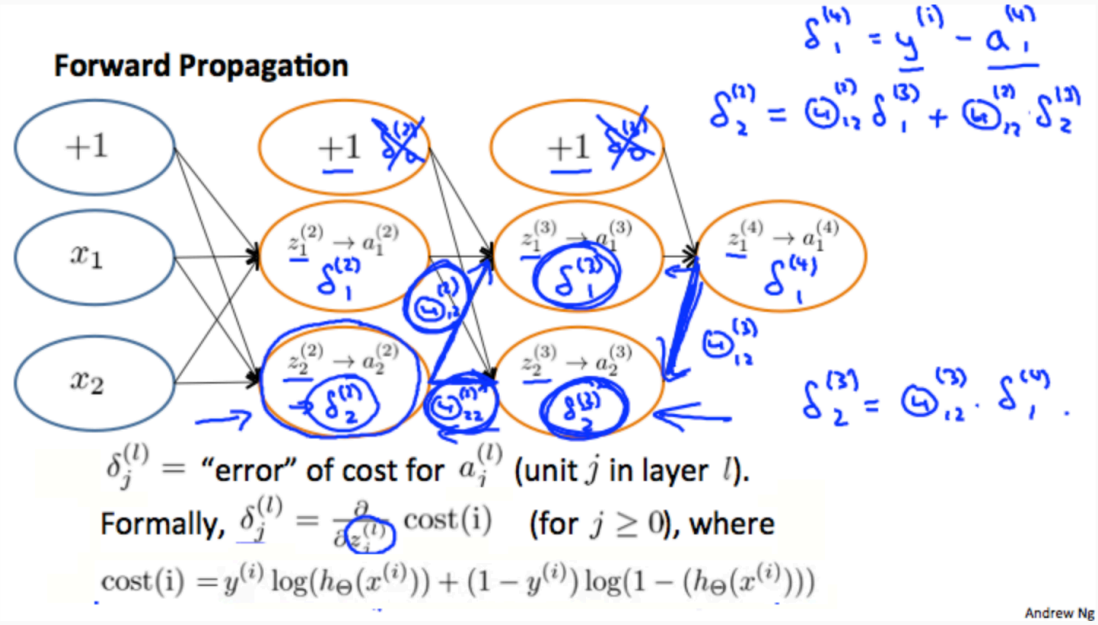
(3) Backpropagation Intuition

- Forward Propagation

$$z_1^{(l)} = \Theta_{10}^{(l-1)} * 1 + \Theta_{11}^{(l-1)} * a_1^{(l-1)} + \Theta_{11}^{(l-1)} * a_2^{(l-1)} \tag{61}$$

Focusing on a single example $x^{(i)}$ $y^{(i)}$ ignore regularization

$$cost = -[y^{(i)} log h_\Theta(x^{(i)}) + (1 - y^{(i)}) log h_\Theta(x^{(i)})] \tag{62}$$

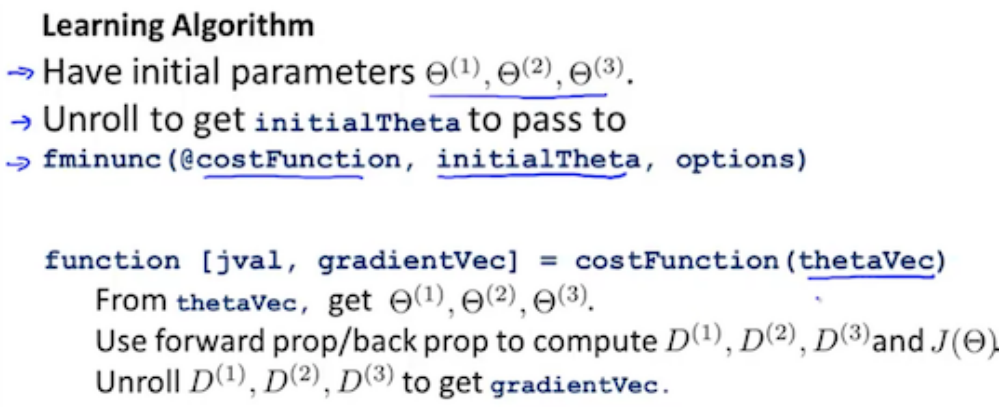


In the image above, to calculate $\delta_2^{(2)}$, we multiply the weights $\Theta_{12}^{(2)}$ and $\Theta_{22}^{(2)}$ by their respective δ values found to the right of each edge. So we get $\delta_2^{(2)} = \Theta_{12}^{(2)} * \delta_1^{(3)} + \Theta_{22}^{(2)} * \delta_2^{(3)}$. To calculate every single possible $\delta_j^{(l)}$, we could start from the right of our diagram. We can think of our edges as our Θ_{ij}. Going from right to left, to calculate the value of $\delta_j^{(l)}$, you can just take the over all sum of each weight times the δ it is coming from. Hence, another example would be $\delta_2^{(3)} = \Theta_{12}^{(3)} * \delta_1^{(4)}$.

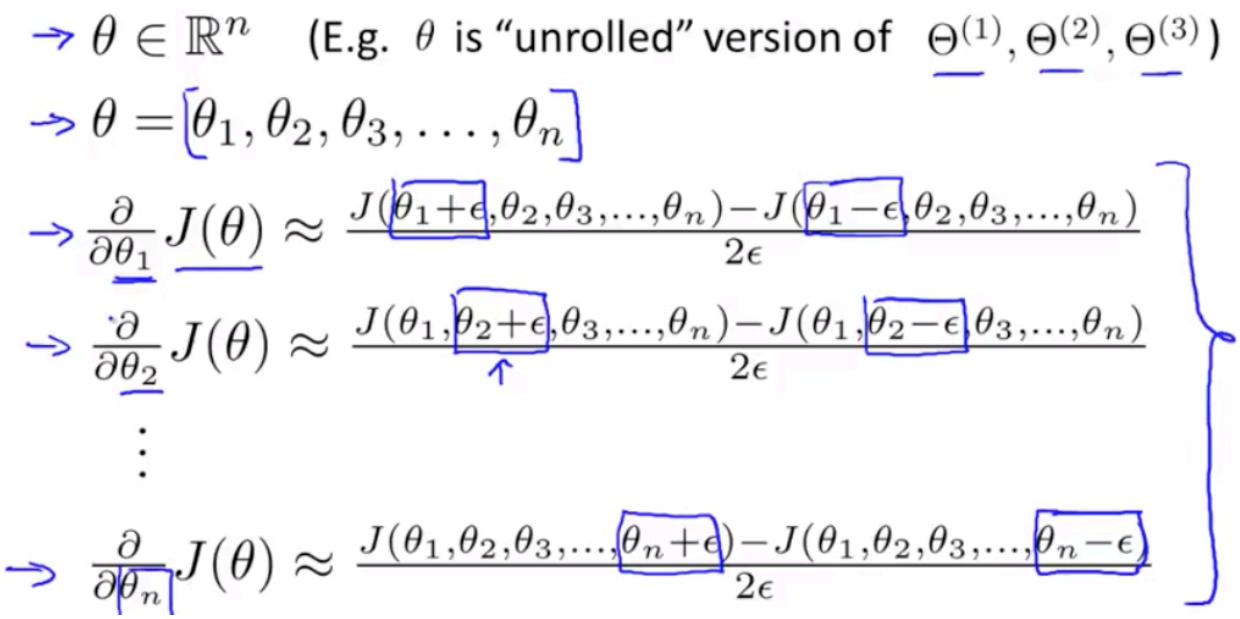
(4) Backpropagation in Practice

- Add all matrices of each layer into one big vector

```
thetaVec = [Theta1(:); Theta2(:); Theta2(:)];
% reshape row 1 to 110 to a 10*11 matrix
Theta1 = reshape(thetaVec(1:110),10,11)
```



- Gradient checking
  - use an approximate value $\frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}$




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

- Check that `gradApprox` ≈ `DVec` (but it is slow)
-