Unsupervised Feature Learning and Deep Learning Tutorial of stanford.edu (UFLDL)

Material contributed by: Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, Sameep Tandon

http://ufldl.stanford.edu/tutorial/

Linear regression with one variable

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

Parameters: θ_0, θ_1

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

Goal: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$?

Gradient descent

- Given a function $f(x)$, our objective is

$$\min_x f(x)$$

- Repeat until convergence{

$$x := x - \alpha \frac{\partial}{\partial x} f$$

}

Gradient descent for linear regression (one variable)

Repeat until convergence{

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

}

Question: which algorithm is correct?

Repeat until convergence{

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

Correct

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

$\theta_0 := temp0$

$\theta_1 := temp1$

Repeat until convergence{

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

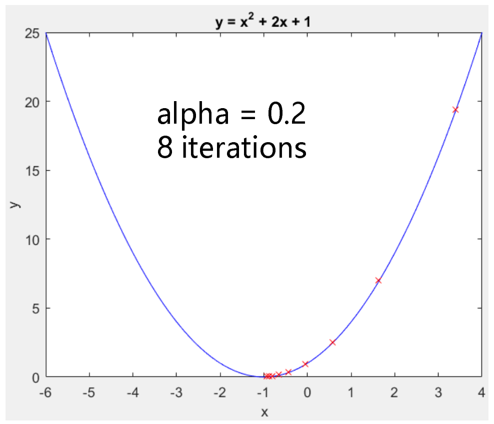
$\theta_0 := temp0$

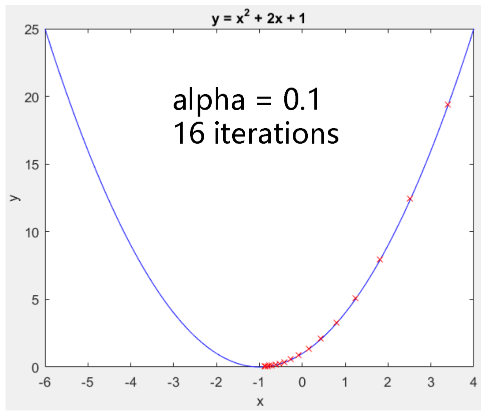
$$temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

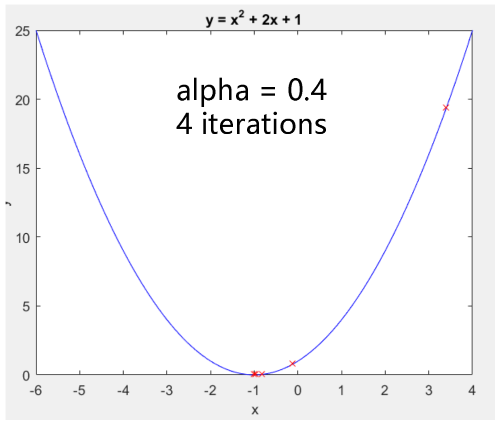
$\theta_1 := temp1$

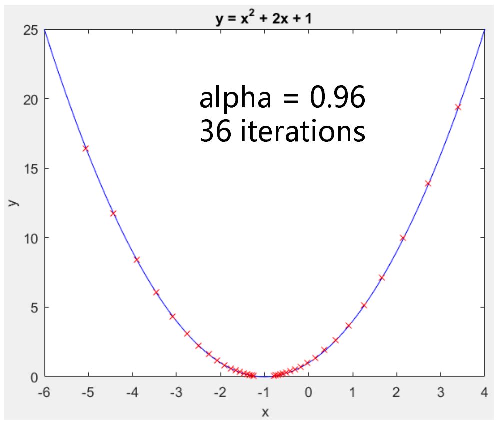
gradesc1d.m

```
x = -6:0.1:4;
fx = x.^2 + 2*x + 1; % y = (x+1)^2
figure(1)
plot(x,fx,'b-');
hold on;
xlabel('x');
ylabel('y');
title('y = x^2 + 2x + 1');

x = 3.4; % initalize
fx = x.^2 + 2*x + 1;% derivative: 2x+2
plot(x,fx,'rx');
alpha = 0.2; % learning rate
iternum = 0;
while 1
    iternum = iternum + 1;
    grad = 2*x+2;
    x = x - alpha*grad;
    y_new = x.^2 + 2*x + 1;
    plot(x,y_new,'rx');
    if abs(fx-y_new) < 0.01
        disp([num2str(iternum) ' iterations']);
        break;
    end
    fx = y_new;
    pause(0.5);
end
```



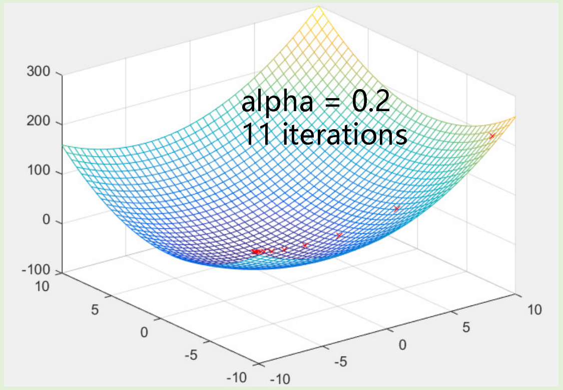




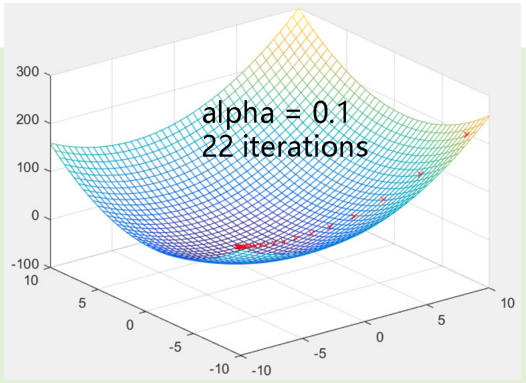


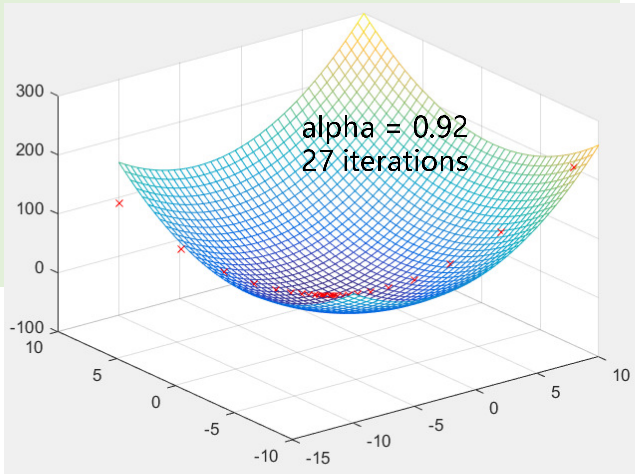
gradesc2d.m

```
x = -10:0.5:10;
fx = -10:0.5:10;
%z = x.^2+y.^2 + 2*x + 7*y + 9;
z = zeros(length(x),length(fx));
for i = 1:length(x)
    for j = 1:length(fx)
        z(i,j) = x(i)^2+fx(j)^2 + 2*x(i) + 7*fx(j) + 9;
    end
end

% gx: 2x + 2 gy: 2y + 7
figure(1)
mesh(x,fx,z)
%colormap gray
hold on;


x = -9;
fx = 9;
z = x^2 + fx^2 + 2*x + 7*fx + 9;
plot3(fx,x,z,'rx');
alpha = 0.2; % learning rate
iternum = 0;
```



```
while 1
    iternum = iternum + 1;
    gradx = 2*x+2;
    grady = 2*fx+7;
    x = x - alpha*gradx;
    fx = fx - alpha*grady;
    z_new = x^2 + fx^2 + 2*x + 7*fx + 9;
    plot3(fx,x,z_new,'rx');
    if abs(z-z_new) < 0.01
        disp([num2str(iternum) ' iterations']);
        break;
    end
    z = z_new;
    pause(0.5);
end
%}
```





Gradient descent for linear regression (one variable)

Repeat until convergence{

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

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

}

Hypothesis

- Hypothesis for multiple features:

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

- If we define $x_0 = 1$, then we have

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \qquad h_\theta = \theta^T x$$

Linear regression with multiple variables

Area of site (1000 square feet)	Size of living place (1000 square feet)	Number of rooms	Ages in years	Selling price
3.472	0.998	7	42	25.9
3.531	1.500	7	62	29.5
$x^{(3)}$ $\underline{2.275}$	$\underline{1.175}$	$\underline{6}$	$\underline{40}$	27.9
4.050	$x_2^{(4)}$ $\underline{1.232}$	6	54	25.9
...	...	...	...	...

- n : number of features
- $x^{(i)}$: input features of i-th training example
- $x_j^{(i)}$: value of feature j in i-th training example

Linear regression with multiple variables

Hypothesis: $h_\theta = \theta^T x$

Parameters: θ

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

Gradient descent:

- Repeat until convergence{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad j = 0, 1, \ldots, n$$

}

Gradient descent for linear regression (multiple variables)

Repeat{

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad j=0,1,\ldots,n$$

}

Feature normalization

- Feature scaling:

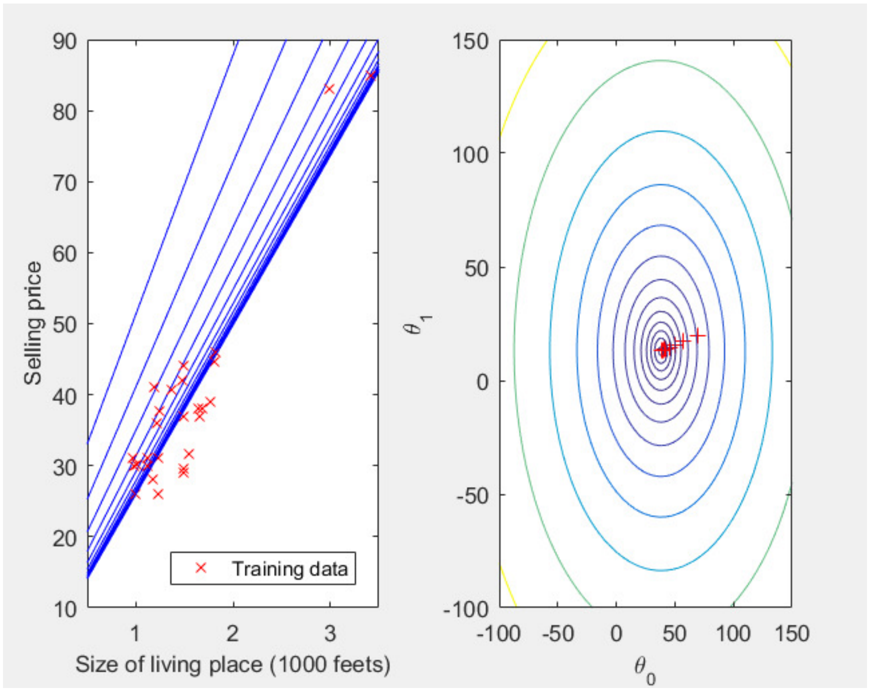
$$x_j^{(i)} = \frac{x_j^{(i)} - \max(x_j^{(k)})}{\max(x_j^{(k)}) - \min(x_j^{(k)})} \quad k=1,2,\ldots,m \quad j=1,2,\ldots,n$$

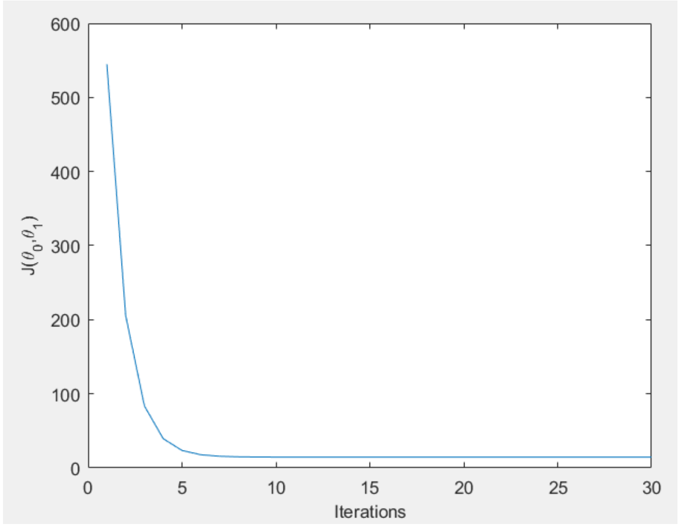
- Standard score:

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j} \quad j=1,2,\ldots,n$$

gradesclinreg.m

```
% This script demonstrates how gradient descent works in linear
regression % with one variable
normark = 1; % 0 - no normalization; 1 - normalization
% load and process data
load('data.mat', 'house_price');
data = table2array(house_price); %size(data)=28*5
data = data(:,[2 5]); %size(data)=28*2, 取两个特征
m = size(data,1); % training example number, m=28
x = data(:,1); %取第一列的特征, 单变量回归分析
if normark == 1
    mu = mean(x);
    sigma = std(x);
    x = (x - mu)/sigma; % normalization 标准化
end
x = [ones(m,1) x]; %在[28*1]矩阵第一列前加1=[1,1,…,1]ᵀ [28*1], 对应截距
y = data(:,2); %y是一个[28*1]的列向量
theta = [70; 20]; % inital theta theta初始值
alpha = 0.4; % learning rate

iternum = 30;
J_history  = zeros(iternum,1); %30个损失函数的历史值
```





```
for iter = 1:iternum  %loop30次

    hx = linspace(-5,5,10);
    %因为数据分布的范围在（-5,5），这里用生成数据hx画出本阶段的回归直线
    if normark == 1
        h = [ones(size(hx)); (hx-mu)/sigma]'*theta;  %数据标准化条件下的用现有的theta计算y
    else
        h = [ones(size(hx)); hx]'*theta; %数据非标准化条件下的用现有的theta计算y
    end
    subplot(1,2,1) % draw left figure
    plot(data(:,1),data(:,2),'rx'); hold on
     %x=data(:,1),y=data(:,2),size(x)=28*1,size(y)=28*1
    xlabel('Size of living place (1000 feets)','FontSize',10);
    ylabel('Selling price','FontSize',10);
    axis([0.5 3.5 10 90]);
    legend('Training data','Location','southeast');

    plot(hx,h,'b'); %利用现在的theta,画出回归直线
    hold off

    subplot(1,2,2) % draw right figure
    % Grid over which we will calculate J
    theta0_vals = linspace(-100, 150, 100);
    theta1_vals = linspace(-100, 150, 100);
```

```
% initialize J_vals to a matrix of 0's
J_vals = zeros(length(theta0_vals), length(theta1_vals));
% Fill out J_vals
for i = 1:length(theta0_vals)
    for j = 1:length(theta1_vals)
        t = [theta0_vals(i); theta1_vals(j)];
        J_vals(i,j) = computeCost(x, y, t);
       %J = sum((X*t - y).*(X*t - y))/(2*m);


    end
end
J_vals = J_vals';
```

```
 % contour plot
   contour(theta0_vals, theta1_vals, J_vals,logspace(1, 10,
30)); hold on %保持等高线图形
   xlabel('\theta_0');
   ylabel('\theta_1');zlabel('J(\theta_0,\theta_1)');
   plot(theta(1),theta(2),'r+'); %每次画出新的theta点

   J_history(iter) = computeCost(x, y, theta);
 % Gradient descent
   theta = theta - alpha*x'*(x*theta-y)/m;
%求出新的theta

   pause(0.2);

end
```

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

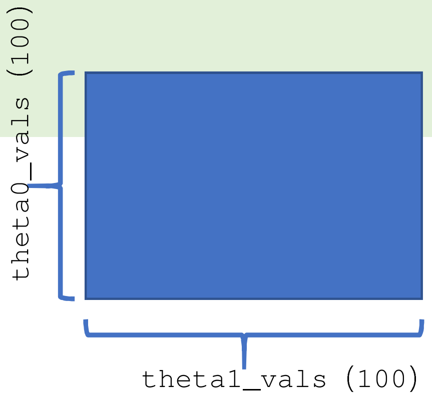
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

```
figure;
plot(J_history);
xlabel('Iterations','FontSize',10);
ylabel('J(\theta_0,\theta_1)','FontSize',10);
hold off;
```

[X,Y] = meshgrid(1:3,10:14)

$$X = \begin{matrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{matrix} \qquad Y = \begin{matrix} 10 & 10 & 10 \\ 11 & 11 & 11 \\ 12 & 12 & 12 \\ 13 & 13 & 13 \\ 14 & 14 & 14 \end{matrix}$$

$$(x,y) = \begin{matrix} (1,10) & (2,10) & (3,10) \\ (1,11) & (2,11) & (3,11) \\ (1,12) & (2,12) & (3,12) \\ (1,13) & (2,13) & (3,13) \\ (1,14) & (2,14) & (3,14) \end{matrix}$$



注1: meshgrid的形式，故用J_val的转置

注2: contour(theta0_vals, theta1_vals, J_vals,logspace(1, 10, 30))
Contour()函数是根据meshgrid(theta0_vals, theta1_vals)形成的网格来画等高线的
这里logspace(1, 10, 30) 是控制等高线的密度和颜色

Logistic regression

- Hypothesis: $h_\theta(x)=\dfrac{1}{1+e^{-\theta^T x}}$
- Parameter: θ
- Goal: $\min\limits_\theta J(\theta)$
- Cost Function: $J(\theta)=-\dfrac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log(h_\theta(x^{(i)}))+(1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right)$

Gradient descent

- Given cost function $J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log(h_\theta(x^{(i)}))+(1-y^{(i)})\log(1-h_\theta(x^{(i)})))$, our objective is $\min\limits_\theta J(\theta)$
- Repeat{

  $\theta_j:=\theta_j-\alpha\dfrac{\partial}{\partial\theta_j}J(\theta)\ \ j=0,1,\ldots,n$ (simultaneously update all θ_j)

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

  }

Identical to linear regression except the hypothesis is different!

- Solution: given parameter θ, we have code that can compute:
  - Cost function $J(\theta)$
  - Partial derivative $\dfrac{\partial}{\partial\theta_j}J(\theta)\ \ j=0,1,\ldots,n$
- Gradient descent
  - Repeat{

    $\theta_j:=\theta_j-\alpha\dfrac{\partial}{\partial\theta_j}J(\theta)\ \ j=0,1,\ldots,n$

    }

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

Hypothesis 1: $\boldsymbol{h_\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \sum_{i=1}^{n} \theta_i x_i, \quad x_0=1$

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

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})=\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h_\theta(x^{(i)})\right)\frac{\partial h_\theta(x^{(i)})}{\partial\theta} = \frac{-1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h_\theta(x^{(i)})\right)\frac{\partial h_\theta(\boldsymbol{x}^{(i)})}{\partial\theta} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\boldsymbol{x}^{(i)}$$

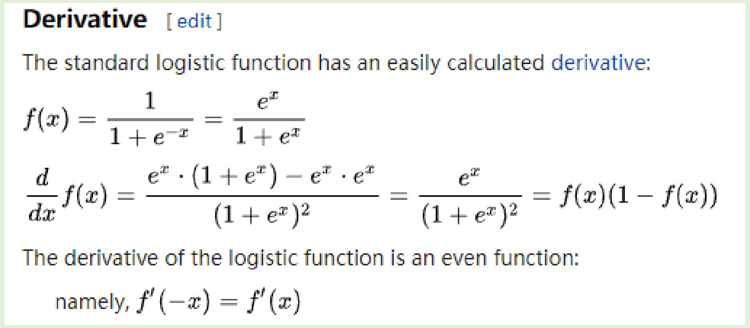
Hypothesis 2: $\boldsymbol{h_\theta}(x) = \dfrac{1}{1+e^{(\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots+\theta_n x_n)}} = \dfrac{1}{1+\boldsymbol{e^{\theta^T x}}}$

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

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})=\frac{-1}{m}\sum_{i=1}^{m}\left(y^{(i)}\frac{1}{\boldsymbol{h_\theta}(\boldsymbol{x}^{(i)})}\frac{\partial h_\theta(\boldsymbol{x}^{(i)})}{\partial\theta} - (1-y^{(i)})\frac{1}{1-\boldsymbol{h_\theta}(\boldsymbol{x}^{(i)})}\frac{\partial h_\theta(\boldsymbol{x}^{(i)})}{\partial\theta}\right)$$

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

$$=\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)}\right)\boldsymbol{x}^{(i)}=\left[\boldsymbol{x}^{(1)}\ \ \boldsymbol{x}^{(2)}\ \ \ldots\boldsymbol{x}^{(m)}\right]\begin{bmatrix}\epsilon_1\\ \vdots\\ \varepsilon_m\end{bmatrix}=\boldsymbol{X}^T(\boldsymbol{h_\theta}(\boldsymbol{X})-\boldsymbol{y})=\begin{cases}\boldsymbol{X}^T(\boldsymbol{X\theta}-\boldsymbol{y})\\ \boldsymbol{X}^T(\boldsymbol{h_\theta}(\boldsymbol{X})-\boldsymbol{y})\end{cases}$$



Derivative [edit]

The standard logistic function has an easily calculated derivative:

$$f(x)=\frac{1}{1+e^{-x}}=\frac{e^x}{1+e^x}$$

$$\frac{d}{dx}f(x)=\frac{e^x\cdot(1+e^x)-e^x\cdot e^x}{(1+e^x)^2}=\frac{e^x}{(1+e^x)^2}=f(x)(1-f(x))$$

The derivative of the logistic function is an even function:

namely, $f'(-x)=f'(x)$

Regularization

- Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$
  - "Simpler" hypothesis, thus less prone to overfitting
- Regularization for linear regression

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

Note j start from 1 not 0!

- Here lambda is the regularization parameter, control the tradeoff between two different goals: fitting the training set well and keeping the parameter small
  - What would happen if lambda is too small?
  - What would happen if lambda is too large?
  - How to tune it? (Later in this course)

Regularized linear regression

Gradient descent:

- Repeat until convergence{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad j = 0, 1, \ldots, n$$

}

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

Gradient descent:

- Repeat until convergence{

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

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

}

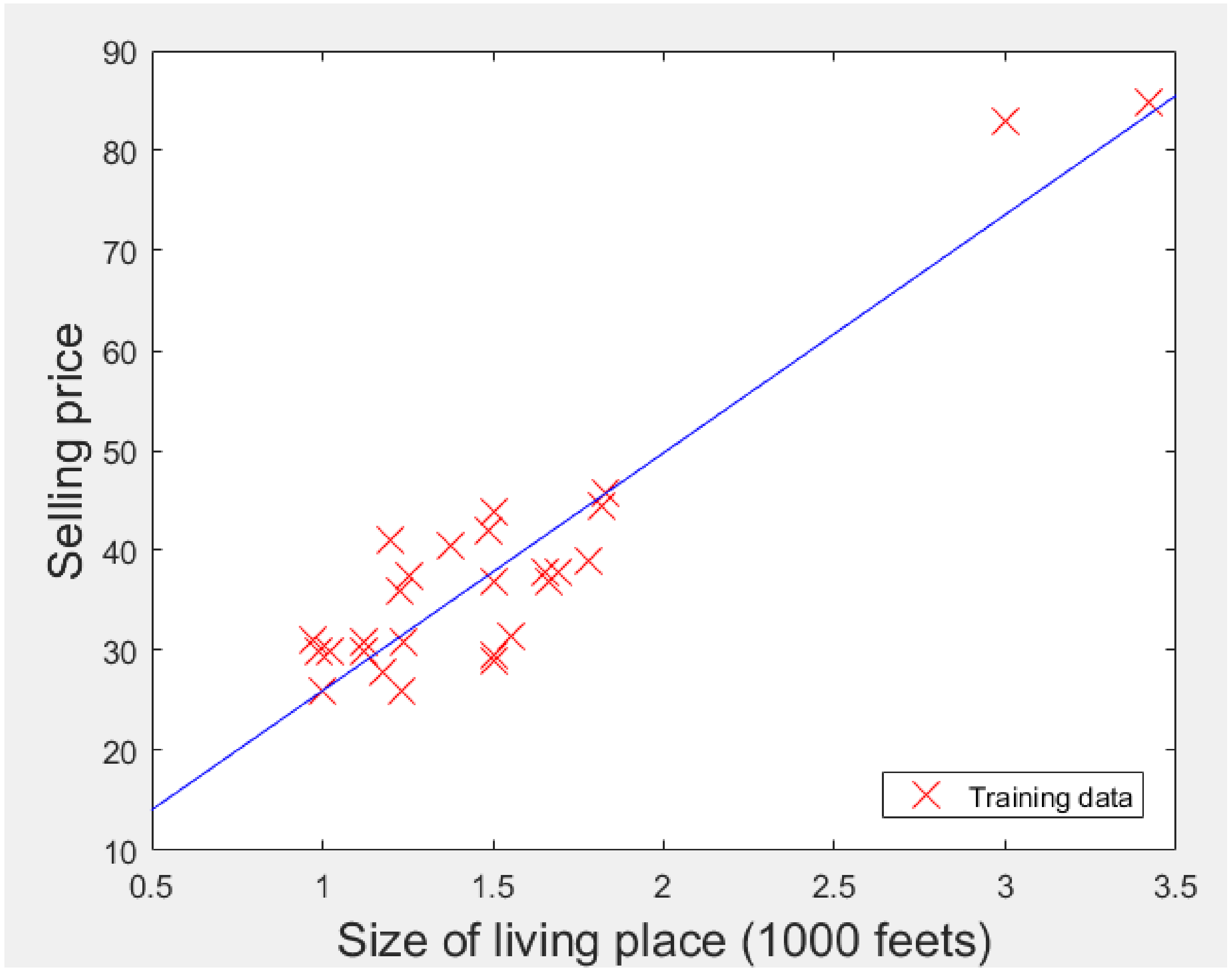
Additional term here

advalgs.m

```matlab
load('data.mat', 'house_price');
data = table2array(house_price);
data = data(:,[2 5]);

plot(data(:,1),data(:,2),'rx','Markersize',12); hold
xlabel('Size of living place (1000
feets)','FontSize',15);
ylabel('Selling price','FontSize',15);
axis([0.5 3.5 10 90]);
legend('Training data','Location','southeast');
m = size(data,1); % training example number m=28

x = [ones(m,1) data(:,1)]; %加一列1对应bais
y = data(:,2);


initial_theta = zeros(size(x,2),1);
% Set options for fminunc
options = optimset('GradObj', 'on', 'MaxIter', 400);
% Run fminunc to obtain the optimal theta
% This function will return theta and the cost
% Here you also need to pass extra parameter x,y to
fminunc, for more
% details, please see help fminunc
[theta, cost] = ...
fminunc(@(t)(costFunction(t, x, y)), initial_theta,
options);

hx = linspace(-5,5,10);
h = [ones(size(hx)); hx]'*theta;
plot(hx,h,'b-');
```

```matlab
function [J, grad] = costFunction(theta, X, y)
%COSTFUNCTION Compute cost and gradient for logistic regression
%   J = COSTFUNCTION(theta, X, y) computes the cost of using
theta as the
%   parameter for logistic regression and the gradient of the
cost
%   w.r.t. to the parameters.

% Initialize some useful values
m = length(y); % number of training examples

% You need to return the following variables correctly
J = 0;
grad = zeros(size(theta));

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the cost of a particular choice of
theta.
%               You should set J to the cost.
%               Compute the partial derivatives and set grad to
the partial
%               derivatives of the cost w.r.t. each parameter in
theta
%
% Note: grad should have the same dimensions as theta
%
J = sum((X*theta - y).*(X*theta - y))/(2*m);

grad = X'*(X*theta - y)/m; %[2*28]*[28*1]=[2*1]

% =============================================================

end
```

$$=\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right)\boldsymbol{x}^{(i)}=\left[\boldsymbol{x}^{(1)}\ \ \boldsymbol{x}^{(2)}\ \ \ldots\boldsymbol{x}^{(m)}\right]\begin{bmatrix}\epsilon_1\\ \vdots\\ \varepsilon_m\end{bmatrix}=X^T(\boldsymbol{h_\theta}(\boldsymbol{X})-\boldsymbol{y})=\begin{cases}X^T(\boldsymbol{X\theta}-\boldsymbol{y})\\ X^T(\boldsymbol{h_\theta}(\boldsymbol{X})-\boldsymbol{y})\end{cases}$$



Selling price
Size of living place (1000 feets)
Training data
90
80
70
60
50
40
30
20
10
0.5
1
1.5
2
2.5
3
3.5

Advanced optimization

- Other advanced algorithms:
  - Conjugate gradient
  - BFGS
  - L-BFGS
  - ...
- Advantages
  - No need to pick learning rate manually
  - Often faster than gradient descent
- Disadvantages:
  - More complex to implement

Implementation is out of scope in the course, but you can still use them in Matlab!

Implementation in Matlab

1. Write your own cost function:

```
function [jVal, gradient] = costFunction(theta)
      jVal = [...code to compute J(theta)...];
      gradient = [...code to compute derivative of J(theta)...];
end
```

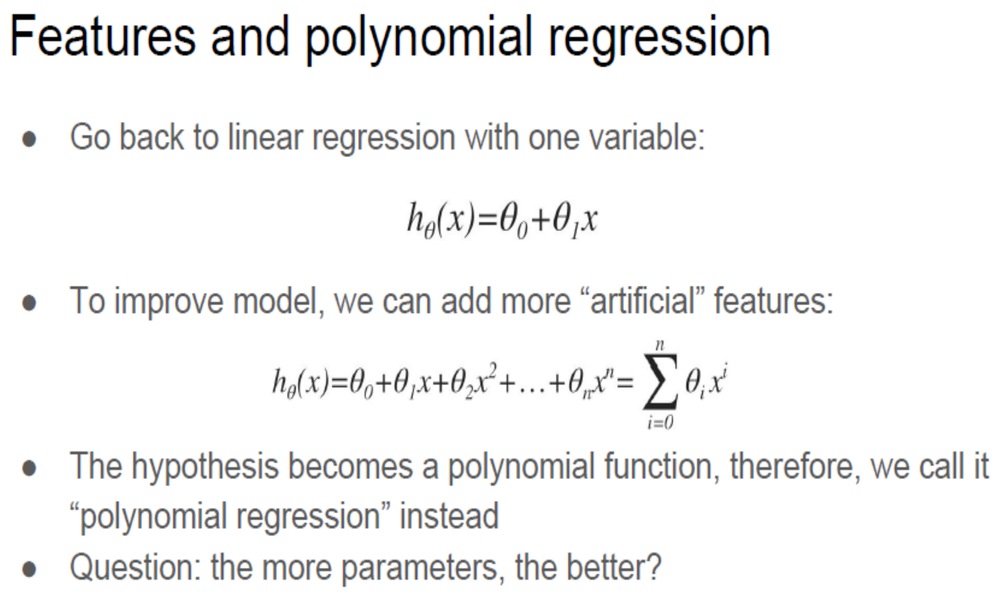
2. Use the function fminunc():

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

3. Example of linear regression

Nonlinear

- What is “nonlinear”？
  - The change of the output is not proportional to the change of the input.
- Nonlinear function:
  - Polynomial, Gaussian,...



Features and polynomial regression

- Go back to linear regression with one variable:

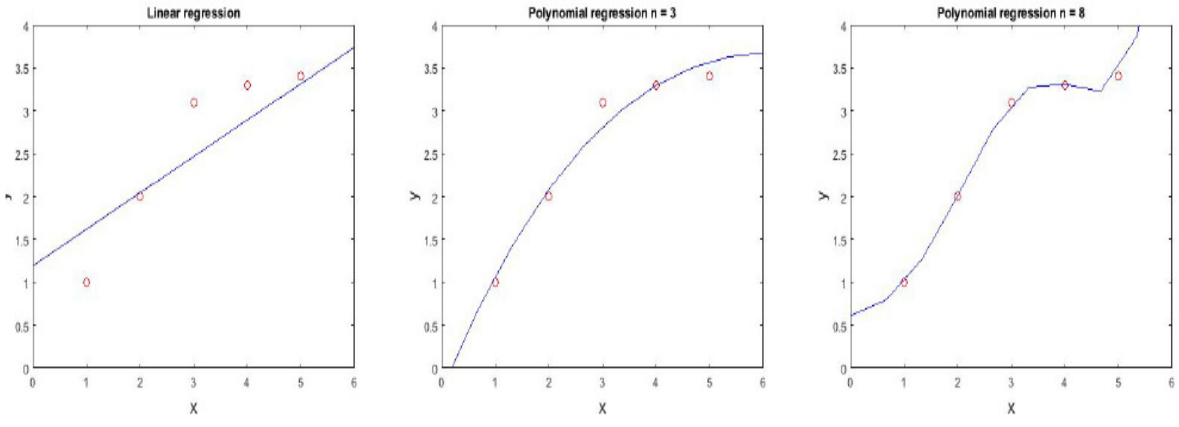
$$h_\theta(x)=\theta_0+\theta_1 x$$

- To improve model, we can add more “artificial” features:

$$h_\theta(x)=\theta_0+\theta_1 x+\theta_2 x^2+\ldots+\theta_n x^n=\sum_{i=0}^{n}\theta_i x^i$$

- The hypothesis becomes a polynomial function, therefore, we call it “polynomial regression” instead
- Question: the more parameters, the better?

Example: linear/polynomial regression



Underfitting, high bias　　Just right　　Overfitting, high variance

Underfitting and Overfitting

- Underfitting (high bias)
  - A phenomenon that the ML model maps poorly to the trend of the data
  - Occurs when your hypothesis is too simple or use too few features
- Overfitting (high variance)
  - A phenomenon that the ML Model fit the training set very well, but fail to generalize to test set. In other words, an overfitting model performs well in training set, but quit bad in test set.
  - Occurs when your hypothesis is excessively complex, e.g, too many parameters or too many features

Address Underfitting and Overfitting

- Underfitting
  - Add more features ○ Use a more complex hypothesis
- Overfitting ○ Reduce number of features
  - Manually select which features to keep (still questionable) ■ Model selection algorithm (later in this course)
  - Regularization
    - Keep all the features, but reduce values of parameters
    - Regularization works well when we have lots of slightly useful features

Regularization

- Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$
    - "Simpler" hypothesis, thus less prone to overfitting
- Regularization for linear regression

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

Note j start from 1 not 0!

- Here lambda is the regularization parameter, control the tradeoff between two different goals: fitting the training set well and keeping the parameter small
    - What would happen if lambda is too small?
    - What would happen if lambda is too large?
    - How to tune it? (Later in this course)

Part I Linear Regression

1 LMS algorithm

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

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

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

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

let’ s consider the gradient descent algorithm

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

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}.$$

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\ &= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\ &= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\ &= (h_\theta(x) - y)\, x_j \end{aligned}$$

the LMS update rule (LMS stands for “least mean squares”),

Exercise 1A: Linear Regression

For this exercise you will implement the objective function and gradient calculations for linear regression in MATLAB.

In the ex1/ directory of the starter code package you will find the file ex1_linreg.m which contains the makings of a simple linear regression experiment. This file performs most of the boiler-plate steps for you:

1.The data is loaded from housing.data. An extra '1' feature is added to the dataset so that θ_1θ1 will act as an intercept term in the linear function.

2.The examples in the dataset are randomly shuffled and the data is then split into a training and testing set. The features that are used as input to the learning algorithm are stored in the variables train.X and test.X. The target value to be predicted is the estimated house price for each example. The prices are stored in "train.y" and "test.y", respectively, for the training and testing examples. You will use the training set to find the best choice of θθ for predicting the house prices and then check its performance on the testing set.

3.The code calls the minFunc optimization package. minFunc will attempt to find the best choice of θθby minimizing the objective function implemented in linear_regression.m. It will be your job to implement linear_regression.m to compute the objective function value and the gradient with respect to the parameters.

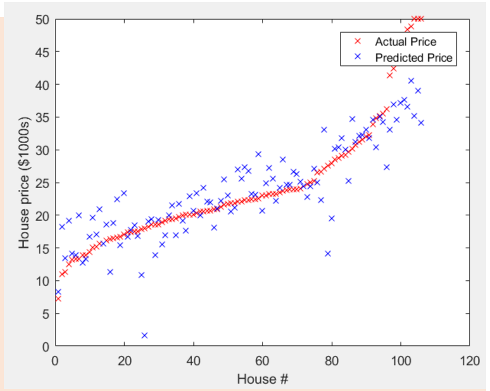
4.After minFunc completes (i.e., after training is finished), the training and testing error is printed out. Optionally, it will plot a quick visualization of the predicted and actual prices for the examples in the test set.

```
Repeat until convergence {
    $\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$    (for every $j$).
}
```

```
Loop {
    for i=1 to m, {
        $\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$    (for every $j$).
    }
}
```

stochastic gradient descent

```
train.X = data(1:end-1,1:400);
train.y = data(end,1:400);

test.X = data(1:end-1,401:end);
test.y = data(end,401:end);

m=size(train.X,2);
n=size(train.X,1);
```



```
options = struct('MaxIter', 200);
theta = minFunc(@linear_regression, theta, options,
train.X, train.y);
%theta = minFunc(@linear_regression, theta, , train.X,
train.y);
%[error,theta] = linear_regression(theta, train.X,
train.y)
fprintf('Optimization took %f seconds.\n', toc);
```

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

```
linear_regression(theta, X, y)
 m=size(X,2);
 n=size(X,1);
 f=0;
 g=zeros(size(theta));
 for k=1:m
   h = theta' * X(:,k) - y(k);
   f = f + 0.5*(h)^2;
   g = g + X(:,k)*h;
  end
end
```

2 The normal equations

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}. \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

$$\begin{aligned} X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(m)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}. \end{aligned}$$

$$\begin{aligned} \nabla_A \mathrm{tr} AB &= B^T \\ \nabla_{A^T} f(A) &= (\nabla_A f(A))^T \\ \nabla_A \mathrm{tr} ABA^TC &= CAB + C^TAB^T \\ \nabla_A |A| &= |A|(A^{-1})^T. \end{aligned}$$

$$\begin{aligned} \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) &= \frac{1}{2}\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= J(\theta) \end{aligned}$$

$$\mathrm{tr} ABC = \mathrm{tr} CAB = \mathrm{tr} BCA,$$
$$\mathrm{tr} ABCD = \mathrm{tr} DABC = \mathrm{tr} CDAB = \mathrm{tr} BCDA.$$

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\ &= \frac{1}{2}\nabla_\theta \left(\theta^TX^TX\theta - \theta^TX^T\vec{y} - \vec{y}^TX\theta + \vec{y}^T\vec{y}\right) \\ &= \frac{1}{2}\nabla_\theta \,\mathrm{tr}\left(\theta^TX^TX\theta - \theta^TX^T\vec{y} - \vec{y}^TX\theta + \vec{y}^T\vec{y}\right) \\ &= \frac{1}{2}\nabla_\theta \left(\mathrm{tr}\,\theta^TX^TX\theta - 2\mathrm{tr}\,\vec{y}^TX\theta\right) \\ &= \frac{1}{2}\left(X^TX\theta + X^TX\theta - 2X^T\vec{y}\right) \\ &= X^TX\theta - X^T\vec{y} \end{aligned}$$

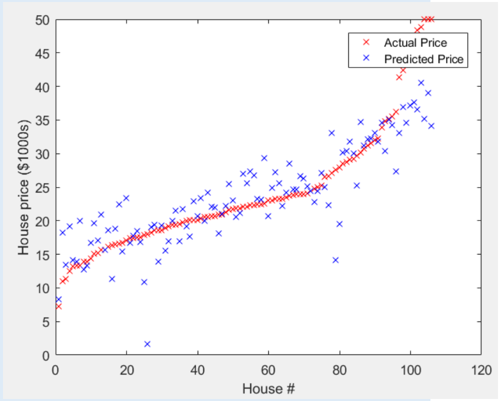
normal equations:

$$X^TX\theta = X^T\vec{y}$$

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

```
theta = rand(n,1);
tic;
theta = minFunc(@linear_regression_vec, theta, options, train.X, train.y);
fprintf('Optimization took %f seconds.\n', toc);
```

```
function [f,g] = linear_regression_vec(theta, X,y)
  %
  % Arguments:
  %   theta - A vector containing the parameter values to optimize.
  %   X - The examples stored in a matrix.
  %       X(i,j) is the i'th coordinate of the j'th example.
  %   y - The target value for each example.  y(j) is the target for example j.
  %
  m=size(X,2);

  % initialize objective value and gradient.
  f = 0;
  g = zeros(size(theta));

  %
  % TODO:  Compute the linear regression objective function and gradient
  %        using vectorized code.  (It will be just a few lines of code!)
  %        Store the objective function value in 'f', and the gradient in 'g'.
  %
%%% YOUR CODE HERE %%%
  h = theta' * X - y;
  f = h * h';
  g = X * h';
```



3 Probabilistic interpretation

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}, \quad p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right).$$

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right). \end{aligned}$$

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

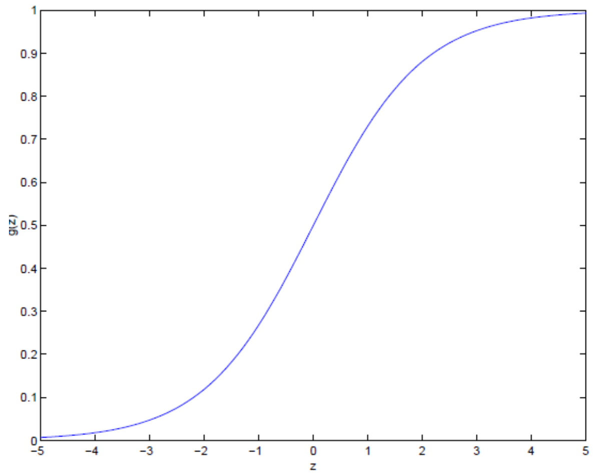
$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2. \end{aligned}$$

Part II Classification and logistic regression

5 Logistic regression

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

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



$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\ &= \frac{1}{(1 + e^{-z})^2} \left(e^{-z}\right) \\ &= \frac{1}{(1 + e^{-z})} \cdot \left(1 - \frac{1}{(1 + e^{-z})}\right) \\ &= g(z)(1 - g(z)). \end{aligned}$$

$$\begin{aligned} P(y = 1 \mid x; \theta) &= h_\theta(x) \\ P(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X; \theta) \\ &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \end{aligned}$$

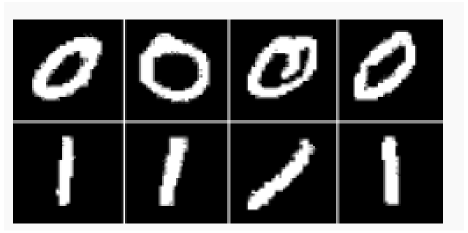
$$\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= \left(y(1 - g(\theta^T x)) - (1 - y) g(\theta^T x) \right) x_j \\
&= (y - h_\theta(x))\, x_j
\end{aligned}$$

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

Exercise 1B

Starter code for this exercise is included in the Starter Code GitHub Repo in the ex1/ directory.
In this exercise you will implement the objective function and gradient computations for logistic regression and use your code to learn to classify images of digits from the MNIST dataset as either “0” or “1”. Some examples of these digits are shown below:



Each of the digits is is represented by a 28x28 grid of pixel intensities, which we will reformat as a vector $x_{(i)}$x(i) with 28*28 = 784 elements. The label is binary, so $y_{(i)} \in \{0,1\}$y(i)∈{0,1}. You will find starter code for this exercise in the ex1/ex1b_logreg.m file. The starter code file performs the following tasks for you:

```
binary_digits = true;
[train,test] = ex1_load_mnist(binary_digits);

train.X = [ones(1,size(train.X,2)); train.X];
test.X = [ones(1,size(test.X,2)); test.X];
tic;
theta=minFunc(@logistic_regression, theta, options,
train.X, train.y);
%fprintf('Optimization took %f seconds.\n', toc);

theta = rand(n,1)*0.001;
tic;
theta=minFunc(@logistic_regression_vec, theta,
options, train.X, train.y);
fprintf('Optimization took %f seconds.\n', toc);

% Print out training accuracy.
tic;
accuracy =
binary_classifier_accuracy(theta,train.X,train.y);
fprintf('Training accuracy: %2.1f%%\n',
100*accuracy);

% Print out accuracy on the test set.
accuracy =
binary_classifier_accuracy(theta,test.X,test.y);
fprintf('Test accuracy: %2.1f%%\n', 100*accuracy);
```

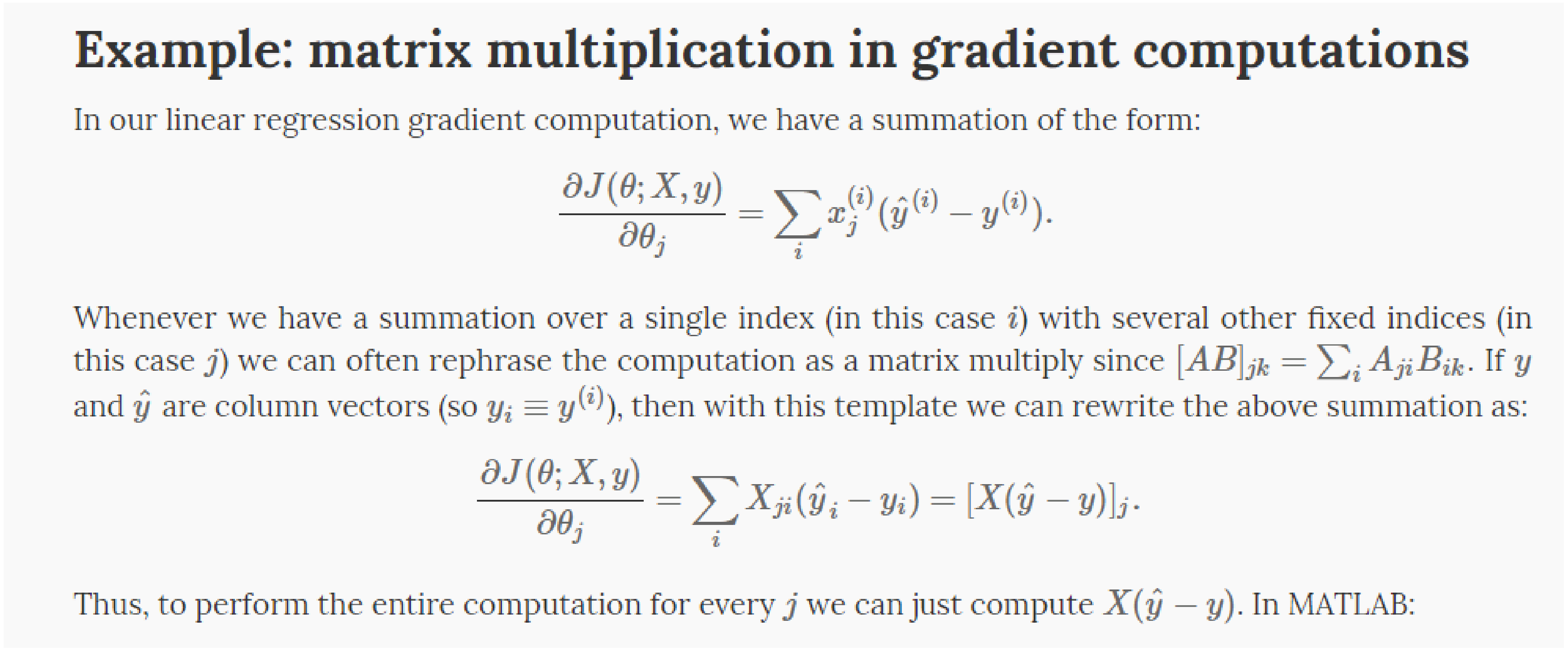
```
function [f,g] = logistic_regression(theta, X, y)
%
  % TODO:  Compute the gradient of the objective
by looping over the dataset and summing
  %        up the gradients (df/dtheta) for each
example. Store the result in 'g'.
  %
%%% YOUR CODE HERE %%%
  for k = 1:m
    h = sigmoid(theta'*X(:,k));
    f = f - (y(k)*log2(h) + (1-y(k))*log2(1-h));
    g = g + X(:,k) * (h - y(k));
  end
end
```

```
function [f,g] = logistic_regression_vec(theta,
X,y)
%
%%% YOUR CODE HERE %%%
  h = sigmoid(theta'*X);
  f = -(y*log2(h') + (1-y)*log2(1-h'));
  g = X*(h-y)';
end
```

Vectorization

linear_regression_vec.m

logistic_regression_vec.m



Example: matrix multiplication in gradient computations

In our linear regression gradient computation, we have a summation of the form:

$$\frac{\partial J(\theta; X, y)}{\partial \theta_j} = \sum_i x_j^{(i)} (\hat{y}^{(i)} - y^{(i)}).$$

Whenever we have a summation over a single index (in this case i) with several other fixed indices (in this case j) we can often rephrase the computation as a matrix multiply since $[AB]_{jk} = \sum_i A_{ji} B_{ik}$. If y and $\hat{y}$ are column vectors (so $y_i \equiv y^{(i)}$), then with this template we can rewrite the above summation as:

$$\frac{\partial J(\theta; X, y)}{\partial \theta_j} = \sum_i X_{ji} (\hat{y}_i - y_i) = [X(\hat{y} - y)]_j.$$

Thus, to perform the entire computation for every j we can just compute $X(\hat{y} - y)$. In MATLAB:

Debugging: Gradient Checking

Recall the mathematical definition of the derivative as:

$$\frac{d}{d\theta}J(\theta) = \lim_{\epsilon \to 0} \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}.$$

Thus, at any specific value of θ, we can numerically approximate the derivative as follows:

$$\frac{J(\theta + \mathrm{EPSILON}) - J(\theta - \mathrm{EPSILON})}{2 \times \mathrm{EPSILON}}$$

In practice, we set $\mathrm{EPSILON}$ to a small constant, say around 10^{-4}. (There's a large range of values of $\mathrm{EPSILON}$ that should work well, but we don't set $\mathrm{EPSILON}$ to be "extremely" small, say 10^{-20}, as that would lead to numerical roundoff errors.)

Thus, given a function $g(\theta)$ that is supposedly computing $\frac{d}{d\theta}J(\theta)$, we can now numerically verify its correctness by checking that

$$g(\theta) \approx \frac{J(\theta + \mathrm{EPSILON}) - J(\theta - \mathrm{EPSILON})}{2 \times \mathrm{EPSILON}}.$$

The degree to which these two values should approximate each other will depend on the details of J. But assuming $\mathrm{EPSILON} = 10^{-4}$, you'll usually find that the left- and right-hand sides of the above will agree to at least 4 significant digits (and often many more).

grad_check_my.m

```
% Load housing data from file.
data = load('housing.data');% size(data)=506*14
data=data'; % size(data)=14*506
% Include a row of 1s as an additional intercept feature.
data = [ ones(1,size(data,2)); data ]; %data=[1,1,…,1;data]
% Shuffle examples.
data = data(:, randperm(size(data,2)));
% Split into train and test sets
% The last row of 'data' is the median home price.
train.X = data(1:end-1,1:400);% train.X=data(1:14,1:400)
train.y = data(end,1:400);  % train.y=data(15,1:400)
test.X = data(1:end-1,401:end)% test.X=data(1:14,401:506)

test.y = data(end,401:end); % test.y=data(15,401:506)
m=size(train.X,2); % m=400
n=size(train.X,1); % n =14
% Initialize the coefficient vector theta to random values.
theta = rand(n,1); %随机初始化
% Run the minFunc optimizer with linear_regression.m as the
objective.
%
% TODO:  Implement the linear regression objective and gradient
computations
% in linear_regression.m
%
tic;
%options = struct('MaxIter', 200);
%theta = minFunc(@linear_regression, theta, options, train.X,
train.y);
%fprintf('Optimization took %f seconds.\n', toc);
grad_check(@linear_regression, theta, 200, train.X, train.y);
```

function average_error = grad_check(fun, theta0, num_checks, varargin)

```
  delta=1e-3;
  sum_error=0;

  fprintf(' Iter       i             err');
  fprintf(' g_est      g             f\n')

  for i=1:num_checks
    T = theta0;
    j = randsample(numel(T),1);
    T0=T; T0(j) = T0(j)-delta;
    T1=T; T1(j) = T1(j)+delta;

    [f,g] = fun(T, varargin{:});
    f0 = fun(T0, varargin{:});
    f1 = fun(T1, varargin{:});

    g_est = (f1-f0) / (2*delta);
    error = abs(g(j) - g_est);

    fprintf('% 5d  % 6d % 15g % 15f % 15f % 15f\n', ...
            i,j,error,g(j),g_est,f);

    sum_error = sum_error + error;
  end

  average=sum_error/num_checks;
```

Softmax Regression

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes. In logistic regression we assumed that the labels were binary: $y^{(i)} \in \{0,1\}$. We used such a classifier to distinguish between two kinds of hand-written digits. Softmax regression allows us to handle $y^{(i)} \in \{1,\ldots,K\}$ where K is the number of classes.

Recall that in logistic regression, we had a training set $\{(x^{(1)},y^{(1)}),\ldots,(x^{(m)},y^{(m)})\}$ of m labeled examples, where the input features are $x^{(i)} \in \Re^{n}$. With logistic regression, we were in the binary classification setting, so the labels were $y^{(i)} \in \{0,1\}$. Our hypothesis took the form:

$$h_\theta(x) = \frac{1}{1+\exp(-\theta^\top x)},$$

and the model parameters θ were trained to minimize the cost function

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

In the softmax regression setting, we are interested in multi-class classification (as opposed to only binary classification), and so the label y can take on K different values, rather than only two. Thus, in our training set $\{(x^{(1)},y^{(1)}),\ldots,(x^{(m)},y^{(m)})\}$, we now have that $y^{(i)} \in \{1,2,\ldots,K\}$. (Note that our convention will be to index the classes starting from 1, rather than from 0.) For example, in the MNIST digit recognition task, we would have $K=10$ different classes.

Given a test input x, we want our hypothesis to estimate the probability that $P(y=k|x)$ for each value of $k=1,\ldots,K$. I.e., we want to estimate the probability of the class label taking on each of the K different possible values. Thus, our hypothesis will output a K-dimensional vector (whose elements sum to 1) giving us our K estimated probabilities. Concretely, our hypothesis $h_\theta(x)$ takes the form:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

Here $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)} \in \Re^n$ are the parameters of our model. Notice that the term $\frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)}$ normalizes the distribution, so that it sums to one.

For convenience, we will also write θ to denote all the parameters of our model. When you implement softmax regression, it is usually convenient to represent θ as a n-by-K matrix obtained by concatenating $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$ into columns, so that

$$\theta = \left[\begin{array}{cccc} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \cdots & \theta^{(K)} \\ | & | & | & | \end{array}\right].$$

Cost Function

We now describe the cost function that we'll use for softmax regression. In the equation below, $1\{\cdot\}$ is the "'indicator function,'" so that $1\{\text{a true statement}\} = 1$, and $1\{\text{a false statement}\} = 0$. For example, $1\{2+2=4\}$ evaluates to 1; whereas $1\{1+1=5\}$ evaluates to 0. Our cost function will be:

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}\right]$$

Notice that this generalizes the logistic regression cost function, which could also have been written:

$$\begin{aligned} J(\theta) &= -\left[\sum_{i=1}^{m} (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) + y^{(i)} \log h_\theta(x^{(i)})\right] \\ &= -\left[\sum_{i=1}^{m}\sum_{k=0}^{1} 1\left\{y^{(i)} = k\right\} \log P(y^{(i)} = k | x^{(i)}; \theta)\right] \end{aligned}$$

$$P(y^{(i)} = k | x^{(i)}; \theta) = \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}$$

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}\right]$$

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m}\left[x^{(i)}\left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta)\right)\right]$$

$$J(\boldsymbol{\theta}) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} \mathbf{1}\left\{y^{(i)} = k\right\} \log \frac{\exp[\, \boldsymbol{\theta}^{(k)T} \boldsymbol{x}^{(i)}]}{\sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]}\right]$$

$$\nabla_{\boldsymbol{\theta}^{(l)}} J(\boldsymbol{\theta}) = -\nabla_{\boldsymbol{\theta}^{(l)}}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} \mathbf{1}\left\{y^{(i)} = k\right\} \log \frac{\exp[\, \boldsymbol{\theta}^{(k)T} \boldsymbol{x}^{(i)}]}{\sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]}\right]$$

$$= -\nabla_{\boldsymbol{\theta}^{(l)}}\left[\sum_{i=1}^{m}\left(\mathbf{1}\{y^{(i)} = l\} \log \frac{\exp[\, \boldsymbol{\theta}^{(l)T} \boldsymbol{x}^{(i)}]}{\sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]} + \sum_{k}^{K} \mathbf{1}\{y^{(i)} = k \neq l\} \log \frac{\exp[\, \boldsymbol{\theta}^{(k)T} \boldsymbol{x}^{(i)}]}{\sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]}\right)\right]$$

$$= -\nabla_{\boldsymbol{\theta}^{(l)}}\left[\sum_{i=1}^{m}\left(\mathbf{1}\{y^{(i)} = l\} \log \frac{\exp[\, \boldsymbol{\theta}^{(l)T} \boldsymbol{x}^{(i)}]}{\sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]} + \mathbf{1}\{y^{(i)} \neq l\} \log \frac{\exp[\, \boldsymbol{\theta}^{(k)T} \boldsymbol{x}^{(i)}]}{\sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]}\right)\right]$$

$$= -\left[\sum_{i=1}^{m}\left(\nabla_{\boldsymbol{\theta}^{(l)}} \mathbf{1}\{y^{(i)} = l\}\left(\boldsymbol{\theta}^{(l)T} \boldsymbol{x}^{(i)} - \log \sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]\right) + \nabla_{\boldsymbol{\theta}^{(l)}} \mathbf{1}\{y^{(i)} \neq l\}\left(\boldsymbol{\theta}^{(k)T} \boldsymbol{x}^{(i)} - \log \sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]\right)\right)\right]$$

$$= -\left[\sum_{i=1}^{m}\left(\mathbf{1}\{y^{(i)} = l\}\left(\boldsymbol{x}^{(i)} - \frac{\exp[\, \boldsymbol{\theta}^{(l)T} \boldsymbol{x}^{(i)}] \boldsymbol{x}^{(i)}}{\sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]}\right) + \mathbf{1}\{y^{(i)} \neq l\}\left(-\frac{\exp[\, \boldsymbol{\theta}^{(l)T} \boldsymbol{x}^{(i)}] \boldsymbol{x}^{(i)}}{\sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]}\right)\right)\right]$$

$$= -\left[\sum_{i=1}^{m}\left(\mathbf{1}\{y^{(i)} = l\} - \frac{\exp[\, \boldsymbol{\theta}^{(l)T} \boldsymbol{x}^{(i)}]}{\sum_{j=1}^{K} \exp[\, \boldsymbol{\theta}^{(j)T} \boldsymbol{x}^{(i)}]}\right) \boldsymbol{x}^{(i)}\right] = = -\sum_{i=1}^{m}\left(\mathbf{1}\{y^{(i)} = l\} - P\{y^{(i)} = l | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\}\right) \boldsymbol{x}^{(i)}$$

Properties of softmax regression parameterization

$$\begin{aligned} P(y^{(i)} = k | x^{(i)} ; \theta) &= \frac{\exp((\theta^{(k)} - \psi)^\top x^{(i)})}{\sum_{j=1}^K \exp((\theta^{(j)} - \psi)^\top x^{(i)})} \\ &= \frac{\exp(\theta^{(k)\top} x^{(i)}) \exp(-\psi^\top x^{(i)})}{\sum_{j=1}^K \exp(\theta^{(j)\top} x^{(i)}) \exp(-\psi^\top x^{(i)})} \\ &= \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^K \exp(\theta^{(j)\top} x^{(i)})}. \end{aligned}$$

Further, if the cost function $J(\theta)$ is minimized by some setting of the parameters $(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(k)})$, then it is also minimized by $(\theta^{(1)} - \psi, \theta^{(2)} - \psi, \ldots, \theta^{(k)} - \psi)$ for any value of ψ. Thus, the minimizer of $J(\theta)$ is not unique. (Interestingly, $J(\theta)$ is still convex, and thus gradient descent will not run into local optima problems. But the Hessian is singular/non-invertible, which causes a straightforward implementation of Newton's method to run into numerical problems.)

Notice also that by setting $\psi = \theta^{(K)}$, one can always replace $\theta^{(K)}$ with $\theta^{(K)} - \psi = \vec{0}$ (the vector of all 0's), without affecting the hypothesis. Thus, one could "eliminate" the vector of parameters $\theta^{(K)}$ (or any other $\theta^{(k)}$, for any single value of k), without harming the representational power of our hypothesis. Indeed, rather than optimizing over the $K \cdot n$ parameters $(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)})$ (where $\theta^{(k)} \in \Re^n$), one can instead set $\theta^{(K)} = \vec{0}$ and optimize only with respect to the $K \cdot n$ remaining parameters.

Relationship to Logistic Regression

$$h_\theta(x) = \frac{1}{\exp(\theta^{(1)\top}x) + \exp(\theta^{(2)\top}x^{(i)})}\begin{bmatrix}\exp(\theta^{(1)\top}x)\\ \exp(\theta^{(2)\top}x)\end{bmatrix}$$

Taking advantage of the fact that this hypothesis is overparameterized and setting $\psi = \theta^{(2)}$, we can subtract $\theta^{(2)}$ from each of the two parameters, giving us

$$\begin{aligned}
h(x) &= \frac{1}{\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)}) + \exp(\vec{0}^\top x)}\begin{bmatrix}\exp((\theta^{(1)}-\theta^{(2)})^\top x)\,\exp(\vec{0}^\top x)\end{bmatrix}\\
&= \begin{bmatrix}\frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})}\\ \frac{\exp((\theta^{(1)}-\theta^{(2)})^\top x)}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})}\end{bmatrix}\\
&= \begin{bmatrix}\frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})}\\ 1-\frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})}\end{bmatrix}
\end{aligned}$$

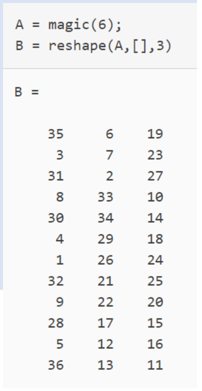
Thus, replacing $\theta^{(2)} - \theta^{(1)}$ with a single parameter vector θ', we find that softmax regression predicts the probability of one of the classes as $\frac{1}{1+\exp(-(\theta')^\top x^{(i)})}$, and that of the other class as $1 - \frac{1}{1+\exp(-(\theta')^\top x^{(i)})}$, same as logistic regression.

ex1c_softmax.m

```
% Load the MNIST data for this exercise.
% train.X and test.X will contain the training and testing images.
%   Each matrix has size [n,m] where:
%      m is the number of examples.
%      n is the number of pixels in each image.
% train.y and test.y will contain the corresponding labels (0 to 9).
binary_digits = false;
num_classes = 10;
[train,test] = ex1_load_mnist(binary_digits); % train = X: [784x60000], y: [1x60000]
test = X: [784x10000], y: [1x10000]
% Add row of 1s to the dataset to act as an intercept term.
train.X = [ones(1,size(train.X,2)); train.X]; % size(train.X)=785*60000
test.X = [ones(1,size(test.X,2)); test.X]; % size(test.X)=785*10000
train.y = train.y+1; % make labels 1-based.% train.y in {1,2,…,10}
test.y = test.y+1; % make labels 1-based .% train.y in {1,2,…,10}

% Training set info
m=size(train.X,2); % m=60000
n=size(train.X,1); % n=785

% Train softmax classifier using minFunc
options = struct('MaxIter', 400);

% Initialize theta.  We use a matrix where each column corresponds to a class,
% and each row is a classifier coefficient for that class.
% Inside minFunc, theta will be stretched out into a long vector (theta(:)).
% We only use num_classes-1 columns, since the last column is always assumed 0.
theta = rand(n,num_classes-1)*0.001; % theta=rand(785,9)*0.001,第10列都为0
```

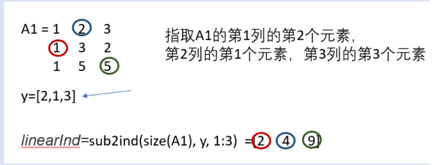
```
% Call minFunc with the softmax_regression_vec.m file as objective.
%
% TODO:  Implement batch softmax regression in the softmax_regression_vec.m
% file using a vectorized implementation.
%
tic;
theta(:)=minFunc(@softmax_regression_vec, theta(:), options, train.X, train.y);
fprintf('Optimization took %f seconds.\n', toc);
theta=[theta, zeros(n,1)]; % expand theta to include the last class. size(theta)=785*(9+1)

% Print out training accuracy.
tic;
accuracy = multi_classifier_accuracy(theta,train.X,train.y);
fprintf('Training accuracy: %2.1f%%\n', 100*accuracy);

% Print out test accuracy.
accuracy = multi_classifier_accuracy(theta,test.X,test.y);
fprintf('Test accuracy: %2.1f%%\n', 100*accuracy);

% % for learning curves
% global test
% global train
% test.err{end+1} = multi_classifier_accuracy(theta,test.X,test.y);
% train.err{end+1} = multi_classifier_accuracy(theta,train.X,train.y);
```

```
function [f,g] = softmax_regression(theta, X, y)
  %size(theta)=(785*9)*1是一个列向量
  %
  % Arguments:
  %   theta - A vector containing the parameter values to
optimize.
  %       In minFunc, theta is reshaped to a long vector.  So we
need to
  %       resize it to an n-by-(num_classes-1) matrix.
  %       Recall that we assume theta(:,num_classes) = 0.
  %
  %   X - The examples stored in a matrix.
  %       X(i,j) is the i'th coordinate of the j'th example.
  %   y - The label for each example.  y(j) is the j'th
example's label.
  %
  m=size(X,2); %m=60000
  n=size(X,1); %n=785
% train.y in {1,2,…,10}, train.y in {1,2,…,10}
% theta=rand(785,9)*0.001,第10列都为0,

  % theta is a vector;  need to reshape to n x num_classes.
%theta=reshape(theta, 785, []);

  theta=reshape(theta, n, []);%size(theta)=[785, 9]
  num_classes=size(theta,2)+1;  % num_classes=10

  % initialize objective value and gradient.
  f = 0;
  g = zeros(size(theta));
```



```
A = magic(6);
B = reshape(A,[],3)

B =

    35     6    19
     3     7    23
    31     2    27
     8    33    10
    30    34    14
     4    29    18
     1    26    24
    32    21    25
     9    22    20
    28    17    15
     5    12    16
    36    13    11
```

```
%
  % TODO:  Compute the softmax objective function and
gradient using vectorized code.
  %        Store the objective function value in 'f', and
the gradient in 'g'.
  %        Before returning g, make sure you form it back
into a vector with g=g(:);
 %%% YOUR CODE HERE %%%
  M = exp(theta'*X); %[9*785]*[785*60000]=[9*60000]
 %wu added begin
  M = [M; ones(1, m)];
  %size(M)=[10*60000],最后一行有60000元素都为1
  p_full = bsxfun(@rdivide, M, sum(M)); %各列归一化,
  %size(p_full)=[10*60000]
  %wu added end

  %wu delete begin
  %p = bsxfun(@rdivide, M, sum(M))
  %p_full = [p; ones(1, m)];
  %wu delete end
 % size(y)=[1x60000]

  I = sub2ind(size(p_full), y, 1:m);%取出p_full中
  f = -sum(sum(log2(p_full(I))));

  one = zeros(size(p_full));
  one(I) = 1;
  g = X*(p_full-one)';
  g = g(:, 1:num_classes-1);

  g=g(:); % make gradient a vector for minFunc
end
```



$$A1 = \begin{matrix} 1 & \textcircled{2} & 3 \\ \textcircled{1} & 3 & 2 \\ 1 & 5 & \textcircled{5} \end{matrix}$$

指取A1的第1列的第2个元素，
第2列的第1个元素，第3列的第3个元素

y=[2,1,3]

linearInd=sub2ind(size(A1), y, 1:3) =[$\textcircled{2}$ $\textcircled{4}$ $\textcircled{9}$]

$$A1 = \begin{matrix} 1 & \enclose{circle}{2} & 3 \\ \enclose{circle}{1} & 3 & 2 \\ 1 & 5 & \enclose{circle}{5} \end{matrix}$$

指取A1的第1列的第2个元素，
第2列的第1个元素，第3列的第3个元素

y=[2,1,3] $\leftarrow$

linearInd=sub2ind(size(A1), y, 1:3) $= [\enclose{circle}{2}\ \ \enclose{circle}{4}\ \ \enclose{circle}{9}]$