# Machine Learning Self-Study with Octave

Jiyuan Shen

----------**DataLog**----------

- About Octave
  - Installation Octave on Mac OSX
  - More about Octave Preparations
  - Octave Preparations List
- Linear Regression Implementation
  - Gradient Descent with one variable
  - Linear Regression with multiple variables
  - Discussion on Learning Rate
- Logistic Regression Implementation
  - Non-regularized logistic regression
  - Regularized logistic regression
- One-vs-All Logistic Regression and Neural Networks
  - Multi-class Classification
  - Neural Network (forward and backward)

# About Octave

## Installation Octave on Mac OSX

1. Download the octave 3.8.0 installer (https://sourceforge.net/projects/octave/files/Octave%20MacOSX%20Binary/2013-12-30%20binary%20installer%20of%20Octave%203.8.0%20for%20OSX%2010.9.1%20%28beta%29/GNU_Octave_3.8.0-6.dmg/download).
2. Open the dmg and follow the instructions.
3. After installation, you will find *octave-cli* and *octave-gui* in your apps.

## More About Octave Preparations

**Q1:** I have installed octave and gnuplot via homebrew, and download auaterm.dmg. But when I try to plot, I get the following message:

```
>>plot(x,y)
gnuplot> set terminal aqua enhanced title "Figure 1" font "*,6"
```

Ans: There are many solutions considered and you may find one works for you:

(s1) Add a line `setenv( "GNUTERM" ," qt" )` to `/usr/local/octave/3.8.0/share/octave/site/m/startup/octaverc`

(s2) Add a line `setenv( "GNUTERM" ," x11" )` to `~/.octaverc`

(s3) `brew reinstall gnuplot –with-aquaterm`

[recommend use vim or vi to edit files, nano is also acceptable.]

Run 'sombrero' in the octave terminal to check if image display is fixed.

## Octave Operations List

Basic Operations && Matrix Representations:

(1) Exp.: [1 2; 2 3; 3 4]  1:0.1:2  A(:)  C=[A B]  C=[A,B]  C=[A;B]

(2) rand(). randn(). ones(). zeros(). eye(). A(x,y). A_{x,y}. A(2,:). A=[A,[x;y;z…]].

(3) size(A). size(A,1). length(A). size(A,2). length(v). length([1;2;3;4;5]).

(4) help. pwd. who. whos. clear

(5) +. -. *. /. ^. ==. ~=. &&. ||. xor()

(6) PS1( '>> ' )

(7) disp(). disp(sprint())

(8) sin(). pi

## Moving Data Around:

(1) load. Exp.: *load feature.dat* or *load( 'feature.dat' )*

(2) Save Exp.: *save hello.mat v;* or *save hello.txt v –ascii;*

## Computing Matrix Data:

* .* .^ ./ log() exp() abs() A' val=max() [val,ind]=max()

< find(A<3) [x,y]=find(A<7) sum() prod() floor() ceil()

max(A,[],1)==max(A) max(A,[],2) max(A(:)) sum(A,1) sum(A,2)

sum(sum(A.*eye() )) sum(sum(A.*flipud(eye() ))) pinv()

## Plotting Data:

plot(x,y). plot(x,y,' r' ) subplot(xr,yr,#) axis([x1 x2 y1 y2])

hold on;

xlabel( '' ) ylabel( '' ) legend( '' ,' ' ,…) title( '…' )

print –dpng '….png' close

figure(1); clf; imagesc(A) colorbar colormap gray

## Control Statements && Function:

```
>> for i=1:10,
        >v(i)=2^i;
        >end;
>> while i<=5,                    >>while true,
        >v(i)=100;                      >v(i)=999;
        >i=i+1;                         >i=i+1; ……(if)……
        >end;                           >end;
>> 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;
 >> function J = costFunctionJ(x,y,theta)
```

Vectorization Example:

Unvectorized implementation:

```
prediction=0.0;
for j=1:n+1,    % Note: the start of vector should be 1 rather than 0
    prediction=prediction+theta(j)*x(j)
end;
```

Vectorized implementation:

```
prediction=theta' *x;    % computation also optimized
```

More can be found in gnu octave:

http://www.gnu.org/software/octave/doc/interpreter/

# Linear Regression Implemention

## Gradient Descent with one variable

In this part, given the dataset, we need to fit the linear regression parameters theta using gradient descent on octave. First, let us recall the algorithm.

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

$$\textit{Hypothesis: } h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 \quad \cdots\cdots (2)$$

$$\text{Update: } \theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \quad \cdots\cdots (3)$$

Then, the given dataset named ex1data1.txt should refers x to the population size in 10,000s, refers y to the profit in $10,000s. Here, let us do it. Based on module principle, make files ex1.m, plotData.m, computeCost.m and gradientDescent.m.

(1) plotData.m := function to display the dataset

```
function plotData(x,y)
figure;    % open a new figure window
plot(x,y,'  rx'  ,'  MarkerSize'  ,10);
xlabel(  'population of City in 10,000s'  );
ylabel(  'profit of City in 10,000s'  );
end
```

(2) computeCost.m := function to compute the cost of linear regression

```
function J = computeCost(X, y, theta)
m = length(y)   % # of training examples
J = 0;
hypothesis = X * theta;
error = hypothesis – y;
J = 1 / (2 * m) * error'   * error;
end
```

(3) gradientDescent.m := function to run gradient descent

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
m = length(y);
J_history = zeros(num_iters, 1);
for iter = 1:num_iters
    hypothesis = X * theta;
    prediction = 1 / m * X'   * (hypothesis - y);
    theta = theta – alpha * prediction;
    J_history(iter) = computeCost(X, y, theta);
end
end
```

(4) ex1.m := the script that steps us

```
%% ==============Initialization============
clear; close all; clc;
```

```
%% ==============Part 1: Plotting============
fprintf(   'Plotting Data ···\n'   )
data = load(   'ex1data1.txt'   );
X = data(: , 1); y = data(: , 2);
m = length(y);
plotData(X, y);    % call the function written above
fprintf(   'Program paused. Press enter to continue.\n'   ); pause;
```

```
%% ==========Part 2: Gradient Descent=========
fprintf( 'Running Gradient Descent ···\n' );
X = [ones(m, 1), data(: , 1)];  % add a column of ones to X
theta = zeros(2, 1);  % initialize fitting parameters
iterations = 1500;
alpha = 0.01  % learning rate
computeCost(X, y, theta)
theta = gradientDescent(X, y, theta, alpha, iterations);
fprintf( 'theta found by gradient descent: ' );
fprintf( '%f %f \n' ,theta(1), theta(2));
```

```
%% ============Part 3: Plotting============
hold on;
plot(X(: , 2), X*theta,  '-' )
legend( 'Training data' ,' Linear Regression' )
hold off
```

```
%% ============Part 4: Predicting============
predict1 = [1, 3.5] * theta;
fprintf( 'For population = 35,000, we predict a profit of %f\n' ,···
    predict1*10000);
predict2 = [1, 7] * theta
fprintf( 'For population = 70,000, we predict a profit of %f\n' ,···
    predict2*10000);
```

After finishing this coding, you can run 'ex1' in the octave terminal (make sure you are in the ex1 directory). We can check the our program by

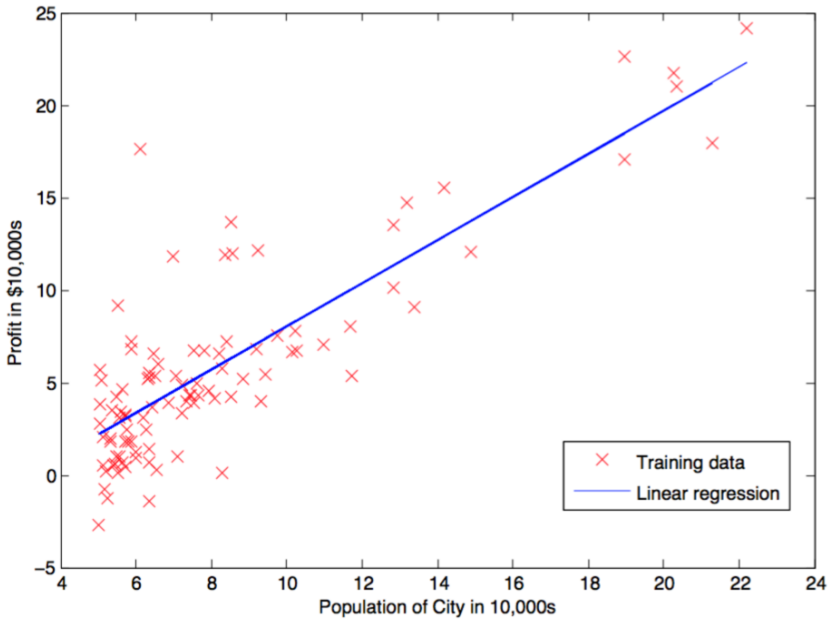
(1) cost should initialy be 32.07;

(2) carefully deal with the matrices(print the dimensions can help);

(3) the value of $J(\theta)$ should decrease each iteration.

Notice: by default octave interprets math operators to be matrix operators. If you do not want matrix multiplication, you need to add the (DOT) notation to specify this to octave (it is exactly similar to matlab).

If all done successfully, you will have the figure as follows:




## Linear Regression with multiple variable

In this part, we will implement linear regression with multiple variables to predict the prices of houses. The given file ex1data2.txt contains a training set of housing prices. The first column is the size of the house (in square foot), the second column is the number of bedrooms, and the third column is the price of the house.

1. Feature Normalization

   When features differ by orders of magnitude, first performing feature scaling can make gradient descent converge much more quickly.

   For implementation, denote mean in mu, denote standard deviation in sigma.

```
Function [X_norm, mu, sigma] = featureNormalize(X)
X_norm = X;
mu = zeros(1, size(X, 2));    sigma = zeros(1, size(X, 2));
```

```
for i=1:size(X_norm, 2)
    mu(i) = mean(X_norm(: , i));
    X_norm(: , i) -= mu(i);
    sigma(i) = std(X_norm(: , i));
    X_norm(: , i) ./= mu(i);
end
```

2. Gradient Descent

Compared to univariate implementation, here the unique difference is the feature number increases. That is the cost function and update function are all remain unchanged. Therefore, let us just get started to implementation.

Notice that when I implemented univariate version, I use matrix exactly, which means it can also work well for our multi one. So the functions in compute -CostMulti.m and gradientDescentMulti.m are same as before.

Then let us consider the ex1_multi.m:

```
%% ==============Initialization============
clear; close all; clc;
```

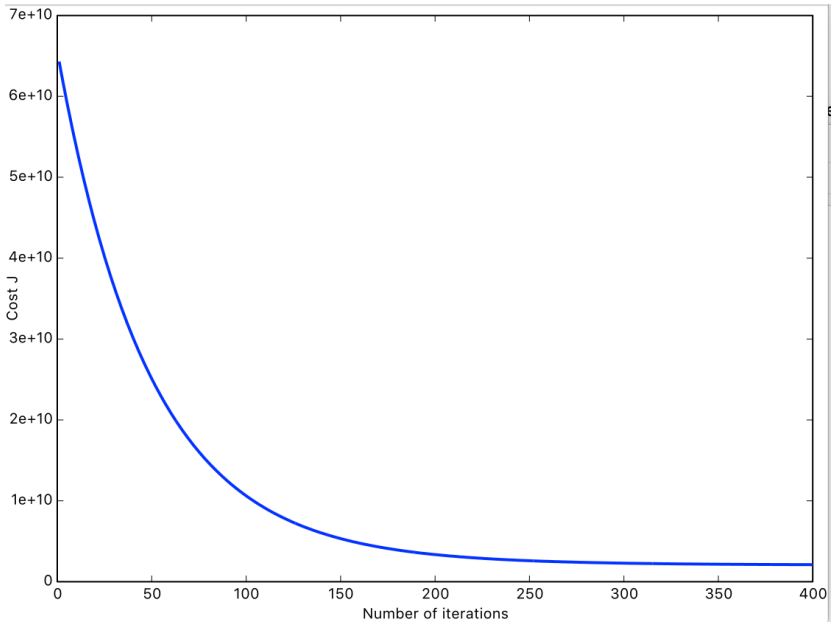
```
%% ===============load data=============
fprintf( 'Loading data ···\n' );
data = load( ': , 1:2' );
X = data(: , 1:2);
y = data(: , 3);
m = length(y);
```

```
%% ===========feature normalization=========
fprintf( 'Normalizing Feature ···\n' );
[X mu sigma] = featureNormalize(X);
X = [ones(m , 1) X];   % add intercept term to X
```

```
%% =============gradient descent==========
alpha = 0.01; num_iters = 400; theta = zeros(3, 1);
[theta, J_history] = gradientDescentMulti(X, y, theta, alpha, num_iters);
```

```
%% ==========plot convergence graph=========
figure;
plot(1:numel(J_history), J_history,  '-b' ,  'LineWidth' , 2);
xlabel( 'Number of iterations' );
ylabel( 'Cost J' );
```

If all done successfully, you will have the figure as follows:




## Discussion on Learning Rate

To compare how different learning rate affect convergence, it is helpful to plot J for several learning rates on the same figure.

# Logistic Regression Implemention

## Non-regularized Logistic Regression

Our question is that given two scores, we want to determine each applicant' s chance of admission. Therefore we need to build a classification model that estimates the probability. Let us get started to implementation.

First, we need to visualize the training data, make a function in plotData.m.

```
%==========Find Indices of Positive and Negative Examples==========
pos = find(y==1);    neg = find(y==0);
%====================Plot Examples====================
plot(X(pos,1), X(pos, 2),    'k+'    ,'    LineWidth'    ,2,'    MarkerSize'    ,7);
plot(X(neg,1), X(neg, 2),    'ko'    ,'    MarkerFaceColor'    ,'    y'    ,'    MarkerSize'    ,7);
```

Before we code the algorithm, we should recall the model:

$$hypothesis: \quad h_\theta(x) = g(\theta^T x)$$

$$sigmoid\ function: \quad g(z) = \frac{1}{1+e^{-z}}$$

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

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

Notice here the difference between linear and logistic algorithms is the hypothesis

(1) Sigmoid Function in sigmoid.m (the matrix version):

```
Function g = sigmoid(z)
g = zeros(size(z));
t = ones(size(z));
g = t ./ (t + e .^(-z))
end
```

Test the codes: sigmoid(0)=0.5

(2) Cost Function & Gradient in costFunction.m (the matrix version):

```
Function [J, grad] = costFunction(theta, X, y)
J = 0;     grad = zeros(zise(theta));
```

```
% ====================cost function====================
h = sigmoid(X * theta);
t = sum((-y'  * log(h)) – ((1-y)'  * log(1-h)) );
J = 1 / m * t
```

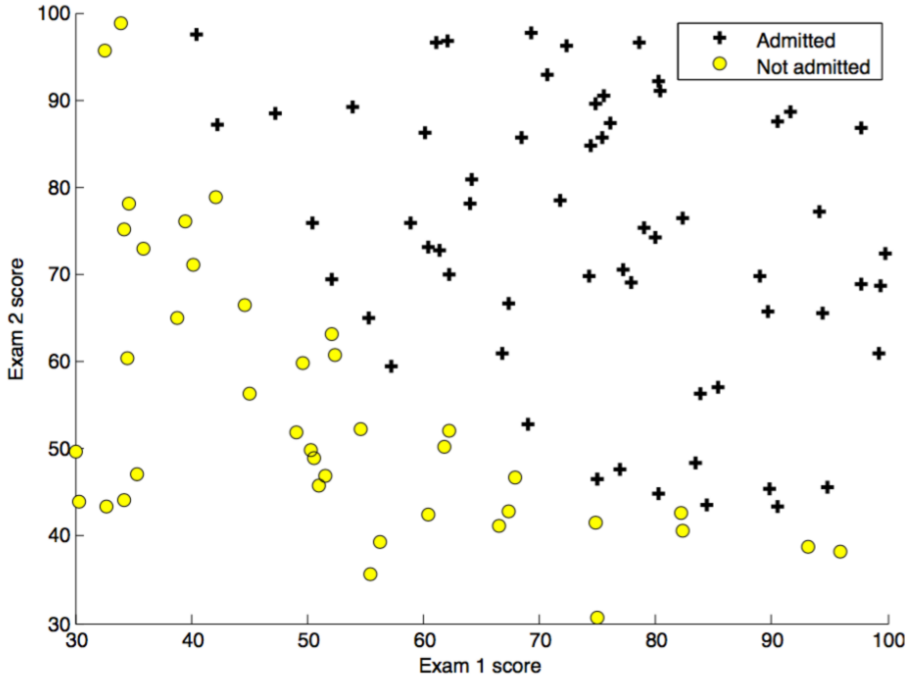
```
% ====================gradient====================
for i=1:size(X, 2)
    grad(:, i) = 1 / m * sum((h-y)'  * X(:, i))
end
```

Then when we successfully have work done, you can get the cost 0.693.

(3) learning parameters using *fminunc*

```
% ============set options for fminunc============
options = optimset( 'GradObj' ,' on' ,' MaxIter' ,400);
% ============obtain optimal theta============
[theta, cost] = fminunc(@(t)(costFunction(t, X, y)), initial_theta, options );
```

Then we call the costFunction using the optimal parameters of theta. We should see the cost be 0.203. For your reference, the original data visual is as:




## Regularized Logistic Regression

First let us recall the regularized model:

***cost function:***

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

***gradient:***

$$\frac{\partial J(\theta)}{\partial \theta_j}=\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right){x_j}^{(i)} \qquad (for\ j\ =\ 0)$$

$$\frac{\partial J(\theta)}{\partial \theta_j}=\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right){x_j}^{(i)}+\frac{\lambda}{m}\theta_j \qquad (for\ j\ \geq 0)$$

Our question is that given two tests, we will determine whether they should be accepted or rejected. We will build a logistic regression model on training data.

(1) Visualizing the training data (same as above)

(2) Feature Mapping

The objective is to map the two input features to quadratic features used in the regularization exercise. It will return a bew feature array with more features, comprising of X1, X2, X1.^2, X2.^2, X1*X2, X1*X2.^2, etc.. Notice here inputs X1, X2 must be the same size.

After mapping, our vector of two features has been transformed into a 28 one.

```
function out = mapFeature(X1, X2)
degree = 6;
out = ones(size(X1(: , 1)));
for i=1:degree
    for j=0:i
        out(:, end+1) = (X1.^(i-j).*(X2.^j));
    end
end
end
```

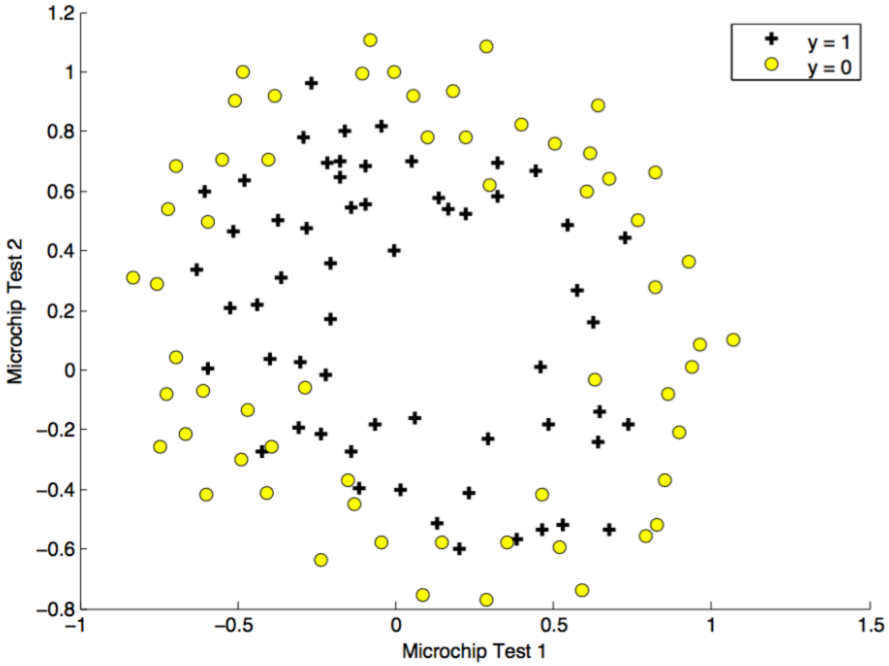
(3) Cost Function and Gradient

Now we will implement the formula mentioned above, there are few changes

```
function [J, grad]=costFunctionReg(theta, X, y, lambda)
……
t2 = lambda / (2*m) * sum(theta'  *  theta)
J = 1 / m * t1 + t2
……
      grad(1, i) = 1 / m * sum ((h-y)'  * X(:, i))
      for j=2:size(theta, 1)
            grad(1, i) = 1 / m * sum ((h-y)'  * X(:, i)) + lambda / m * theta
      end
……
```

compared to the one in non-regularized version. Therefore, we only list changes:

If all done successfully, you will have the figure as follows:




Check your results with cost about 0.693.

(4) Learning parameters using *fminunc* (same as above)

# One-vs-All Logistic Regression & Neural Networks

## Multi-class Classification

In this part, we will use logistic regression and neural networks to recognize handwritten digits (from 0 to 9).

About DataSet —— you are given a data set in ex3data1.mat that contains 5000 training examples of handwritten digits. Each training example is a 20 pixel by 20 pixel grayscale image of the digit. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. Each of these training examples becomes a single row in our data matrix X. This gives us a 5000 by 400 matrix X where every row is a training example for a handwritten digit image. The second part of the training set is a 5000-dim vector y that contains labels. Since there are no 0 indice in octave, let 0 labeled as 10, and keep 1 to 9 the same.

(1) Visualizing the data

Considering how special the dataset is, we visual a subset of the training set. We randomly select 100 rows from X and pass those rows to the displayData. You will see an image like:

(2) Vectorizing the cost function and gradient for regularized style

(3) One-vs-All Classification

Let denote K (here 10) classes in our dataset, we will train multiple regularized logistic regression classifiers, one for each of the K classes.

```
function [all_theta]=oneVsAll(X, y, num_labels, lamda)
m = size(X, 1)
n = size(X, 2)
all_theta = zeros(num_labels, n+1);
X = [ones(m, 1) X];
Initial_theta=zeros(n+1,1);
Options=optimset( 'GradObj' ,' on' ,' MaxIter' ,50);
for i=1:num_labels
    [theta] = …
        fmincg(@(t)(lrCostFunction(t, X, (y==i), lamda)),…
            initial_theta, options);
        all_theta(i, :) = theta' ;
end
```

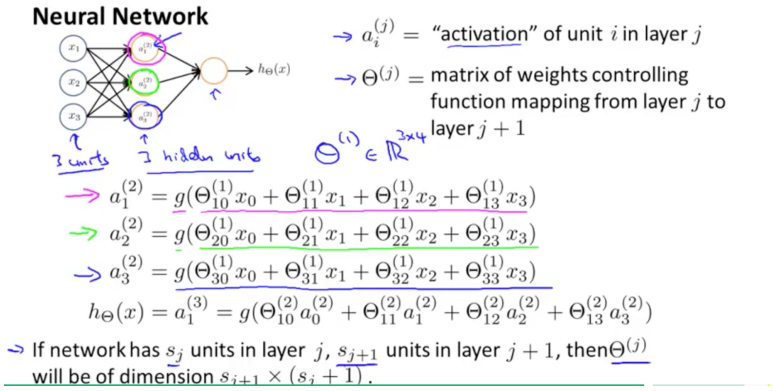
If all done successfully, you will see the result:

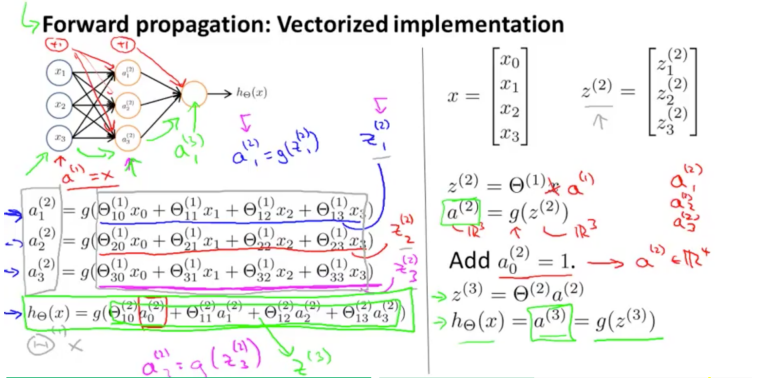
```
Jiyuan — octave-3.8.0 — gnuplot ◂ octave-cli-3.8.0 — 80×24
Unknown or ambiguous terminal name 'X11'
Loading and Visualizing Data ...
Program paused. Press enter to continue.

Training One-vs-All Logistic Regression...
Iteration    50 | Cost: 1.381258e-02
Iteration    50 | Cost: 5.725227e-02
Iteration    50 | Cost: 6.380756e-02
Iteration    50 | Cost: 3.630427e-02
Iteration    50 | Cost: 6.154967e-02
Iteration    50 | Cost: 2.210121e-02
Iteration    50 | Cost: 3.588050e-02
Iteration    50 | Cost: 8.406781e-02
Iteration    50 | Cost: 7.984389e-02
Iteration    50 | Cost: 9.818996e-03
Program paused. Press enter to continue.

Training Set Accuracy: 94.960000
octave:20>
```

# Neural Network

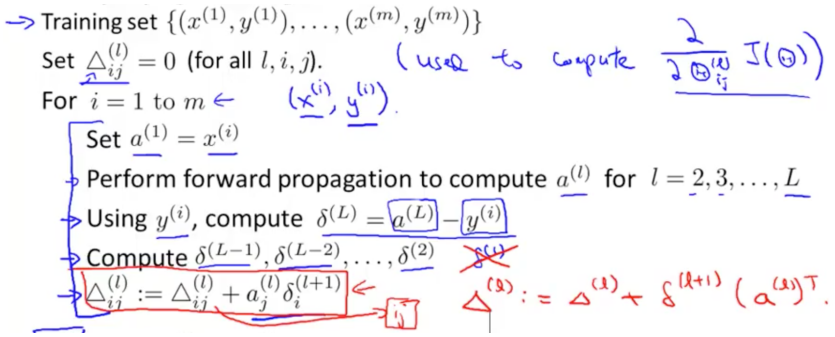
In consideration of computation when we meet large scale dataset, we have a motivation of neural network, the figure below is a screenshot when I take the courser class, it clarifies explicitly:

## Forward Propagation



## BackPropagation（BP）




## More Discussions

***For BackPropagation,*** there are two stages: the first is forward signals; the second is backward errors to update weights. In order to more specifically learn this network, let us take an example:

1. Network Initialization

   (1) Suppose node number of the input layer is ***n***, node number of the hidden layer is ***l***, node number of the output layer is ***m.***

   (2) Suppose weight from layer input to hidden is $\boldsymbol{\omega_{ij}}$, weight from layer hidden to output is $\boldsymbol{\omega_{jk}}$.

   (3) Suppose bias from layer input to hidden is $\boldsymbol{a_j}$, bias from layer hidden to putput is $\boldsymbol{b_k}$.

   (4) Suppose learning rate is $\boldsymbol{\eta}$, activity rule is ***g(x)***, here we select g(x) as sigmoid function, that is $g(x) = \frac{1}{1+e^{-x}}$. (logistic regression)

2. Output of Hidden Layer

$$H_j = g(\sum_{i=1}^{n} w_{ij} x_i + a_j)$$

3. Output of Output Layer

$$O_k = \sum_{j=1}^{l} H_j \omega_{jk} + b_k$$

4. Compute Error

   Let us select error formula as

$$E = \frac{1}{2} \sum_{k=1}^{m} (Y_k - O_k)^2$$

   here, $Y_k$ is the output expectation.

5. Update Weight

$$\begin{cases} \omega_{ij} = \omega_{ij} + \eta H_j(1-H_j)x_i \sum_{k=1}^{m} \omega_{jk} e_k & (input\ to\ hidden) \\ \omega_{jk} = \omega_{jk} + \eta H_j e_k & (hidden\ to\ output) \end{cases}$$

$$( e_k = Y_k - O_k )$$

Explanation of Formula:

In order to minimize the error formula $E$, we use gradient descent.

$$\frac{\partial E}{\partial \omega_{jk}} = \sum_{k=1}^{m} (Y_k - O_k)\left(-\frac{\partial O_k}{\partial \omega_{jk}}\right) = (Y_k - O_k)(-H_j) = -e_k H_j$$

Therefore, weight update formula is

$$\omega_{jk} == \omega_{jk} + \eta \left|\frac{\partial E}{\partial \omega_{jk}}\right| = \omega_{jk} + \eta H_j e_k \ldots (hidden\ to\ output)$$

*Notice here*

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial H_j}\frac{\partial H_j}{\partial \omega_{ij}}$$

where,

$$\frac{\partial E}{\partial H_j} = (Y_1 - O_1)\left(-\frac{\partial O_1}{\partial H_j}\right) + \cdots + (Y_m - O_m)\left(-\frac{\partial O_m}{\partial H_j}\right)$$

$$= -(Y_1 - O_1)\omega_{j1} - \cdots - (Y_m - O_m)\omega_{jm}$$

$$= -\sum_{k=1}^{m} (Y_k - O_k)\omega_{jk} = -\sum_{k=1}^{m} \omega_{jk} e_k$$

and,

$$\frac{\partial H_j}{\partial \omega_{ij}} = \frac{\partial g(\sum_{i=1}^{n} \omega_{ij} x_i + a_j)}{\partial \omega_{ij}}$$

$$= g\left(\sum_{i=1}^{n} w_{ij} x_i + a_j\right) \times \left[1 - g\left(\sum_{i=1}^{n} w_{ij} x_i + a_j\right)\right] \times \frac{\partial(\sum_{i=1}^{n} \omega_{ij} x_i + a_j)}{\partial \omega_{ij}}$$

$$= H_j(1 - H_j)x_i$$

Therefore, weight update formula can also denoted as

$$\omega_{ij} = \omega_{ij} + \eta H_j(1 - H_j)x_i \sum_{k=1}^{m} \omega_{jk} e_k \ldots (input\ to\ hidden)$$

6. Update Bias

$$\begin{cases} a_j = a_j + \eta H_j(1 - H_j) \sum_{k=1}^{m} \omega_{jk} e_k & (input\ to\ hidden) \\ b_k = b_k + \eta e_k & (hidden\ to\ output) \end{cases}$$

(hidden to output)

$$\frac{\partial E}{\partial b_k} = (Y_k - O_k)\left(-\frac{\partial O_k}{\partial b_k}\right) = -e_k$$

Therefore, bias update formula is

$$b_k = b_k + \eta e_k \ldots (hidden\ to\ output)$$

(input to hidden)

$$\frac{\partial E}{\partial a_j} = \frac{\partial E}{\partial H_j}\frac{\partial H_j}{\partial a_j}$$

where,

$$\frac{\partial E}{\partial H_j} = (Y_1 - O_1)\left(-\frac{\partial O_1}{\partial H_j}\right) + \cdots + (Y_m - O_m)\left(-\frac{\partial O_m}{\partial H_j}\right)$$

$$= -(Y_1 - O_1)\omega_{j1} - \cdots - (Y_m - O_m)\omega_{jm}$$

$$= -\sum_{k=1}^{m}(Y_k - O_k)\omega_{jk} = -\sum_{k=1}^{m}\omega_{jk}e_k$$

therefore,

$$a_j = a_j + \eta H_j(1 - H_j)\sum_{k=1}^{m}\omega_{jk}e_k \ldots (input\ to\ hidden)$$

7. Judge if the algorithm converge

   Common used one is specify a number and judge if the error is smaller.

@Jiyuan 2016/9/10