Standford University

Week1

Machine Learning

-Grew out of work in AI

- New capability for computers

Examples:

-Database mining

Large datasets from growth of automation/web.

e.g. Web click data, medical records, biology, engineering

Applications can’t program by hand.

E.g. Autonomous helicopter, handwriting recognition, most of Natural Language Processing(NLP), Computer Vision

Self-customizing Programs

E.g. Amazon, Netflix product recommendations

Understanding human learning (brain, real AI)

Class on machine learning

Machine learning is one of the most exciting recent technologies.

Here we

Learn state of the art and also gain practice implementing and deploying these algorithms yourself.

Unknowingly, you’re using dozens of learning algorithms in your day-to-day life.

Web search engine uses learning algorithm developed by Google or Microsoft, has learned how to rank web pages.

Facebook or Apple’s photo typing application and it recognizes your friend’s photos, that’s also machine learning

Every time you Read email and your spam filter saves your from having to wade through tons of spam emails, that’s also learning algorithm.

For me one of the reasons I’m excited is the AI dream of someday building machines as intelligent as you or me.

We’re long way away from that goal, but many AI researchers believe that the best way to towards that goal is through learning algorithms that try to mimic how the human brain learns.

I’ll tell you a little bit about that too in this content.

Learn about state-of-the-art machine learning algorithms.

Andrew Ng

What is Machine Learning?

Machine Learning Definition

* Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

Checkers Players

Here's a definition of what is machine learning as due to Arthur Samuel.

He defined machine learning as the field of study

that gives computers the ability to learn without being explicitly learned.

Samuel's claim to fame was that back in the 1950, he wrote a checkers playing

program and the amazing thing about this checkers playing program

was that Arthur Samuel himself wasn't a very good checkers player.

But what he did was he had to programmed maybe tens of thousands of games against

himself, and by watching what sorts of board positions tended to lead to wins and

what sort of board positions tended to lead to losses,

the checkers playing program learned over time what are good board positions and

what are bad board positions.

And eventually learn to play checkers better than the Arthur Samuel

himself was able to.

This was a remarkable result.

Arthur Samuel himself turns out not to be a very good checkers player.

But because a computer has the patience to play tens of thousands of

games against itself, no human has the patience to play that many games.

By doing this, a computer was able to get so much checkers playing experience

that it eventually became a better checkers player than Arthur himself.

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

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T in this setting?

* Classifying emails as spam or not spam. --------🡪T
* Watching you label emails as spam or not spam. -----🡪E
* The number (or fraction) of emails correctly classified as spam/not spam. ---🡪P
* None of the above-this is not a machine learning problem

Absolutely, the correct ans is classifying emails as spam or not spam. --🡪T

Machine Learning Algoritihms:

* Supervised learning
* Unsupervised learning

Others: Reinforcement learning, recommender systems.

Also talk about: Practical advice for applying learning algorithms.

Review: What is Machine Learning?

Two definitions of ML are offered.

Arthur Samuel described it as: “the field of study that gives computers the ability to learn without being explicitly programmed.” This is an older, informal definition.

Tom Mitchell provides a more modern definition: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Example: playing checkers.

E = the experience of playing many games of checkers.

T = the task of playing checkers.

P = the probability that the program will win the next game.

In general, any machine learning problem can be assigned to one of the two broad classifications:

* Supervised learning
* Unsupervised learning

Supervised learning:

**Supervised Learning**

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

**Example 1:**

Given data about the size of houses on the real estate market, try to predict their price. Price as a function of size is a continuous output, so this is a regression problem.

We could turn this example into a classification problem by instead making our output about whether the house "sells for more or less than the asking price." Here we are classifying the houses based on price into two discrete categories.

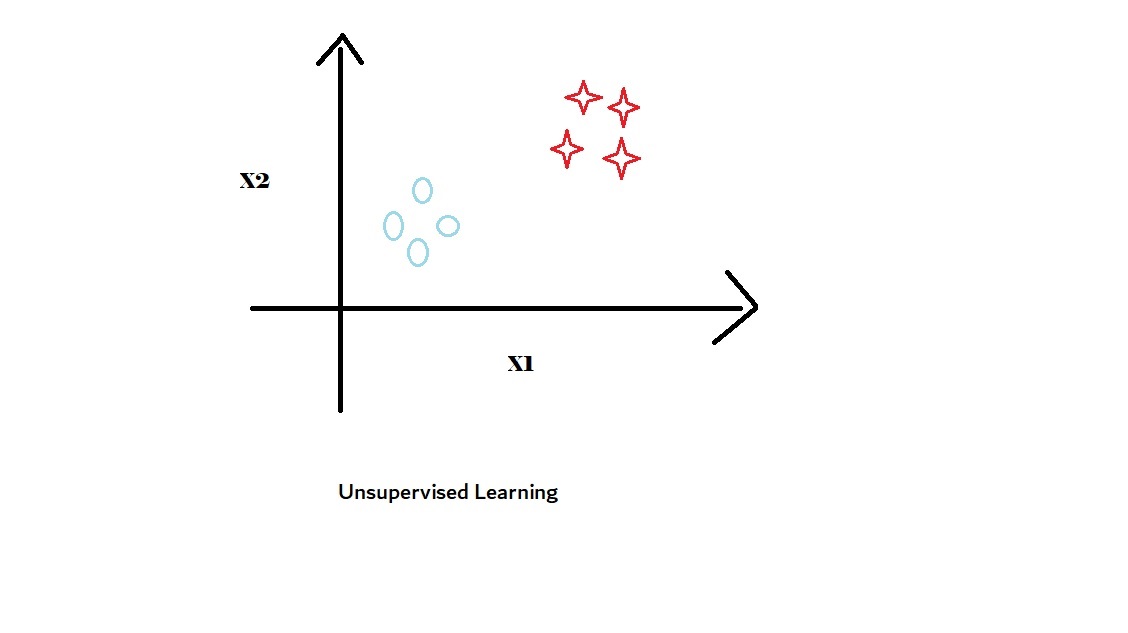
**Example 2**:

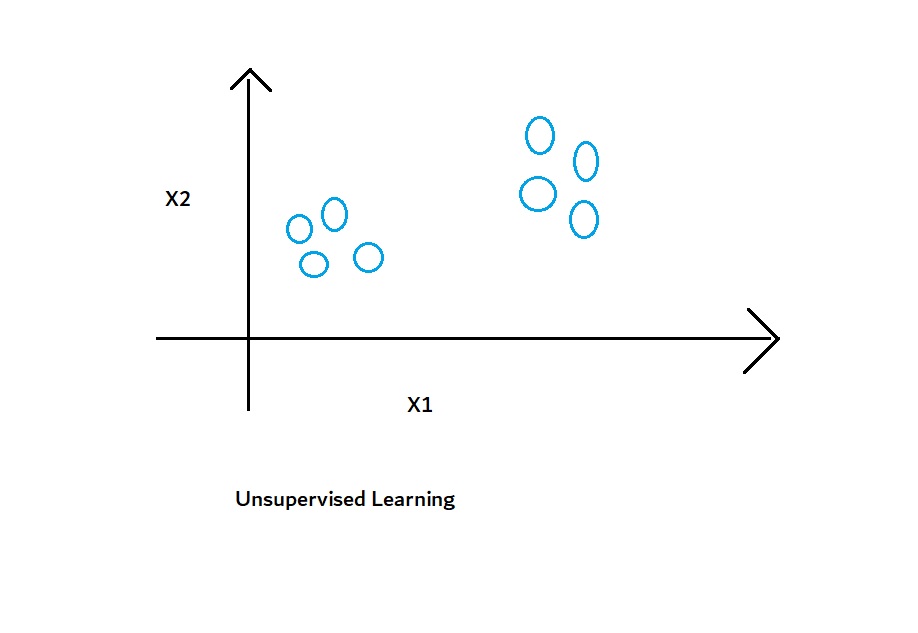
(a) Regression - Given a picture of a person, we have to predict their age on the basis of the given picture

(b) Classification - Given a patient with a tumor, we have to predict whether the tumor is malignant or benign.

Unsupervised Learning

Recall





Cocktail party problem algorithm

[W,s,v] = svd((repmat(sum(x.\*x,1),size(x,1),1).\*x)\*x');

Of the following examples, which would you address using an unsupervised learning algorithm? (Check all that apply.)

1. Given email labeled as spam/not spam, learn a spam filter.
2. Given a set of news articles found on the web, group them into set of articles about the same story.
3. Given a database of customer data, automatically discover market segments and group customers into different market segments.
4. Given a dataset of patients diagnosed as either having diabetes or not, learn to classify new patients as having diabetes or not.

Correct Ans: b and c

## Unsupervised Learning

Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.

We can derive this structure by clustering the data based on relationships among the variables in the data.

With unsupervised learning there is no feedback based on the prediction results.

**Example:**

Clustering: Take a collection of 1,000,000 different genes, and find a way to automatically group these genes into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.

Non-clustering: The "Cocktail Party Algorithm", allows you to find structure in a chaotic environment. (i.e. identifying individual voices and music from a mesh of sounds at a [cocktail party](https://en.wikipedia.org/wiki/Cocktail_party_effect)).

# **Who are Mentors?**

**Mentor Program Overview:**

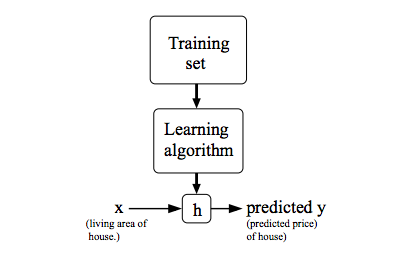
Community Mentors are successful, dedicated Coursera learners who volunteer to assist with support and discussion forum moderation in courses that they have already completed. They have been recruited by Coursera to encourage newer learners, answer questions, set an example by posting thoughtful and timely content, and report platform bugs and inappropriate content to Coursera.

As you use the [discussion areas](https://www.coursera.org/learn/machine-learning/discussions), please be aware that the ideas expressed by participants in this course, including the Mentors, do not represent the views of Stanford University. The mentors are not employed by Stanford University and they have not been vetted by Stanford University as experts on course content or course facilitation.

# Model Representation

To establish notation for future use, we’ll use x^{(i)}*x*(*i*) to denote the “input” variables (living area in this example), also called input features, and y^{(i)}*y*(*i*) to denote the “output” or target variable that we are trying to predict (price). A pair (x^{(i)} , y^{(i)} )(*x*(*i*),*y*(*i*)) is called a training example, and the dataset that we’ll be using to learn—a list of m training examples {(x^{(i)} , y^{(i)} ); i = 1, . . . , m}(*x*(*i*),*y*(*i*));*i*=1,...,*m*—is called a training set. Note that the superscript “(i)” in the notation is simply an index into the training set, and has nothing to do with exponentiation. We will also use X to denote the space of input values, and Y to denote the space of output values. In this example, X = Y = ℝ.

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function h : X → Y so that h(x) is a “good” predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis. Seen pictorially, the process is therefore like this:



When the target variable that we’re trying to predict is continuous, such as in our housing example, we call the learning problem a regression problem. When y can take on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment, say), we call it a classification problem.

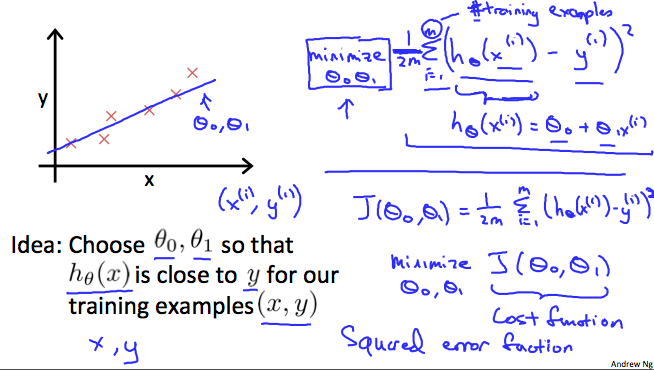
# Cost Function

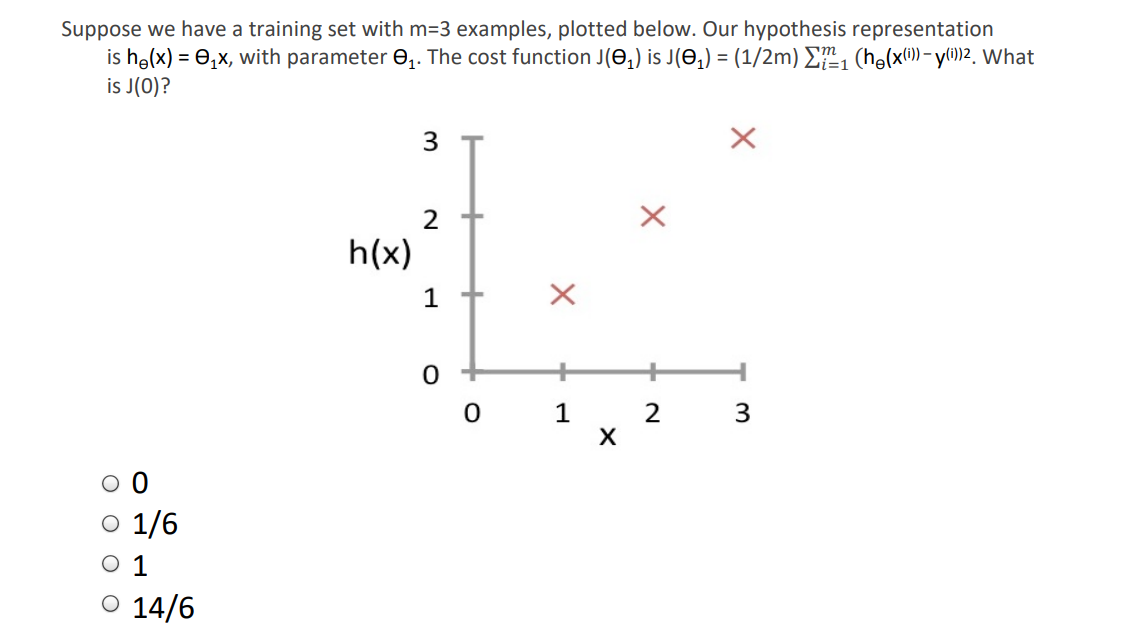
We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average difference (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's and the actual output y's.

J(\theta\_0, \theta\_1) = \dfrac {1}{2m} \displaystyle \sum \_{i=1}^m \left ( \hat{y}\_{i}- y\_{i} \right)^2 = \dfrac {1}{2m} \displaystyle \sum \_{i=1}^m \left (h\_\theta (x\_{i}) - y\_{i} \right)^2*J*(*θ*0​,*θ*1​)=2*m*1​*i*=1∑*m*​(*y*^​*i*​−*yi*​)2=2*m*1​*i*=1∑*m*​(*hθ*​(*xi*​)−*yi*​)2

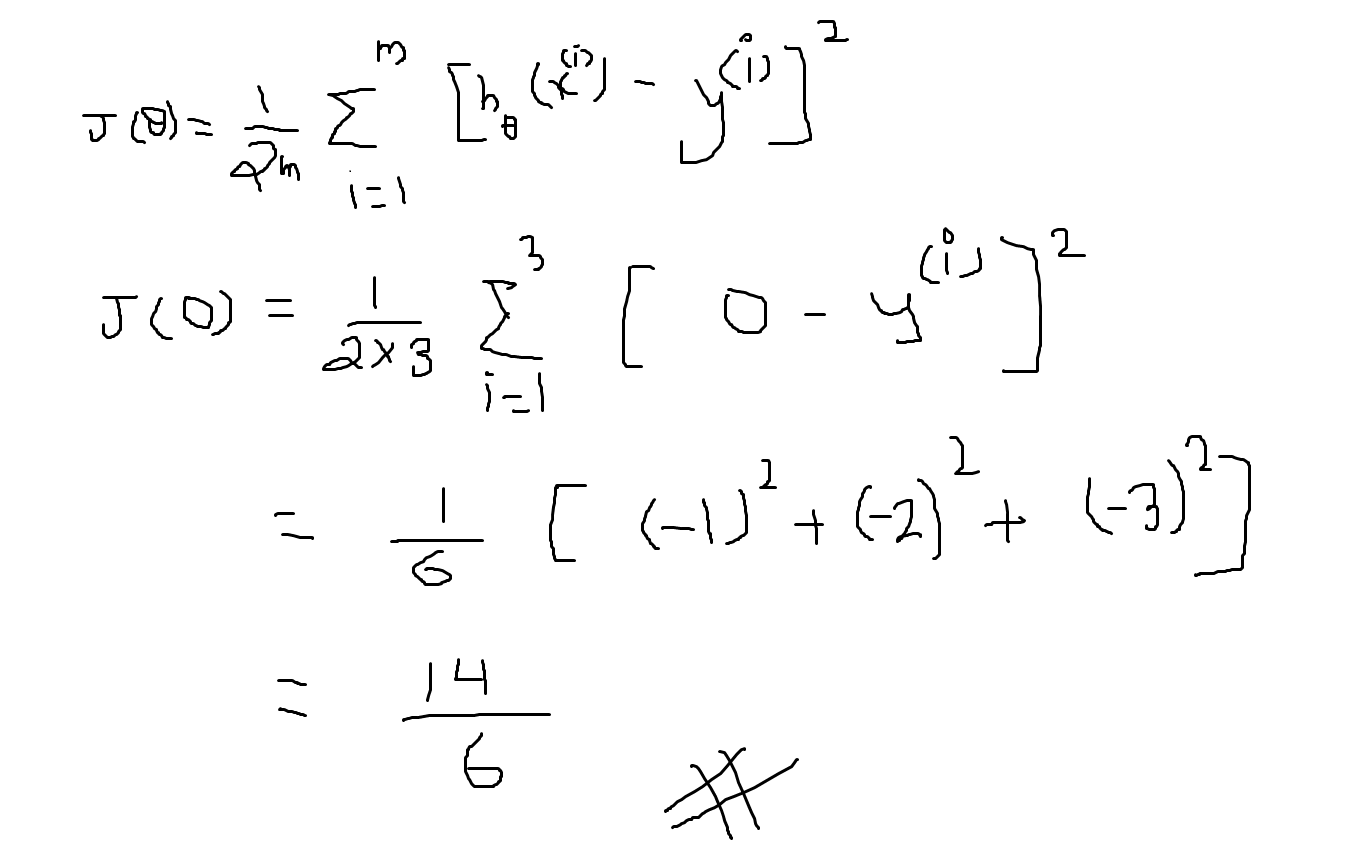
To break it apart, it is \frac{1}{2}21​ \bar{x}*x*ˉ where \bar{x}*x*ˉ is the mean of the squares of h\_\theta (x\_{i}) - y\_{i}*hθ*​(*xi*​)−*yi*​ , or the difference between the predicted value and the actual value.

This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved \left(\frac{1}{2}\right)(21​) as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the \frac{1}{2}21​ term. The following image summarizes what the cost function does:





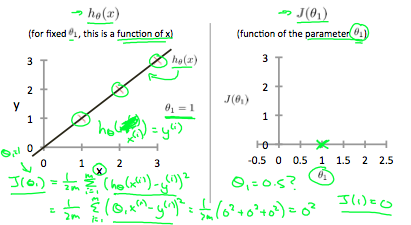
Ans: 14/6



# Cost Function - Intuition I

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make a straight line (defined by h\_\theta(x)*hθ*​(*x*)) which passes through these scattered data points.

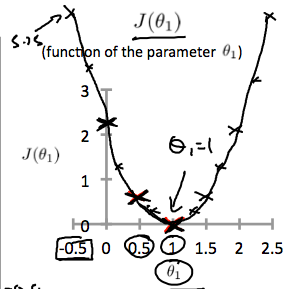
Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. Ideally, the line should pass through all the points of our training data set. In such a case, the value of J(\theta\_0, \theta\_1)*J*(*θ*0​,*θ*1​) will be 0. The following example shows the ideal situation where we have a cost function of 0.



When \theta\_1 = 1*θ*1​=1, we get a slope of 1 which goes through every single data point in our model. Conversely, when \theta\_1 = 0.5*θ*1​=0.5, we see the vertical distance from our fit to the data points increase.



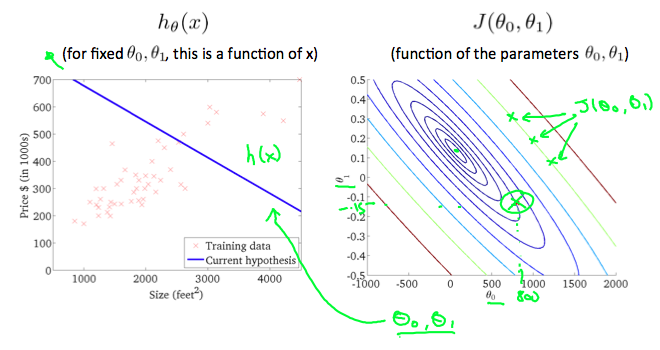
This increases our cost function to 0.58. Plotting several other points yields to the following graph:



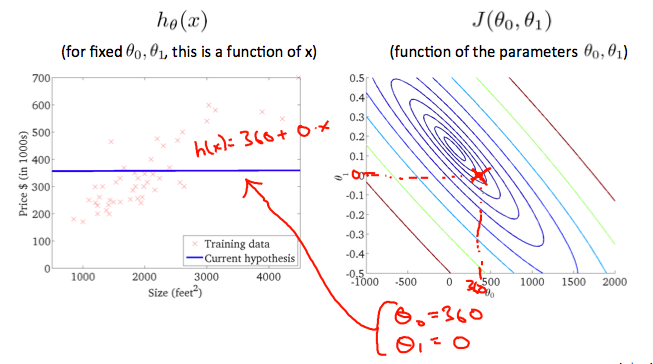
Thus, as a goal, we should try to minimize the cost function. In this case, \theta\_1 = 1*θ*1​=1 is our global minimum.

# Cost Function - Intuition II

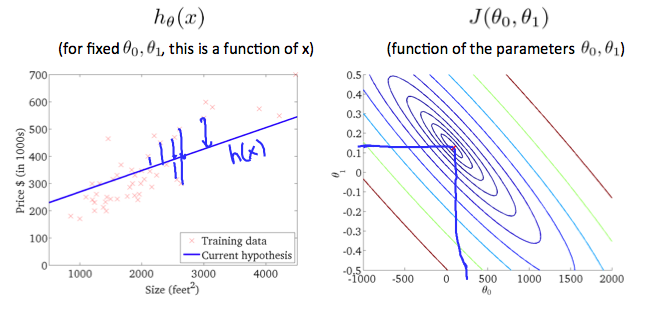
A contour plot is a graph that contains many contour lines. A contour line of a two variable function has a constant value at all points of the same line. An example of such a graph is the one to the right below.



Taking any color and going along the 'circle', one would expect to get the same value of the cost function. For example, the three green points found on the green line above have the same value for J(\theta\_0,\theta\_1)*J*(*θ*0​,*θ*1​) and as a result, they are found along the same line. The circled x displays the value of the cost function for the graph on the left when \theta\_0*θ*0​ = 800 and \theta\_1*θ*1​= -0.15. Taking another h(x) and plotting its contour plot, one gets the following graphs:



When \theta\_0*θ*0​ = 360 and \theta\_1*θ*1​ = 0, the value of J(\theta\_0,\theta\_1)*J*(*θ*0​,*θ*1​) in the contour plot gets closer to the center thus reducing the cost function error. Now giving our hypothesis function a slightly positive slope results in a better fit of the data.



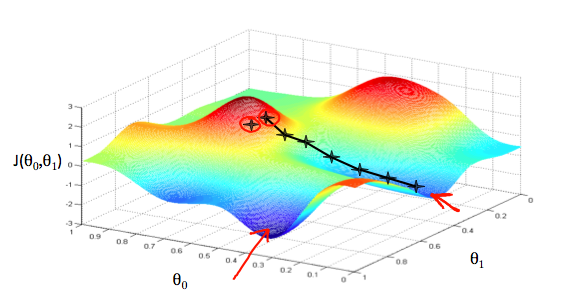
The graph above minimizes the cost function as much as possible and consequently, the result of \theta\_1*θ*1​ and \theta\_0*θ*0​ tend to be around 0.12 and 250 respectively. Plotting those values on our graph to the right seems to put our point in the center of the inner most 'circle'.

# Gradient Descent

So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in the hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields \theta\_0*θ*0​ and \theta\_1*θ*1​ (actually we are graphing the cost function as a function of the parameter estimates). We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters.

We put \theta\_0*θ*0​ on the x axis and \theta\_1*θ*1​ on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters. The graph below depicts such a setup.



We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum. The red arrows show the minimum points in the graph.

The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter α, which is called the learning rate.

For example, the distance between each 'star' in the graph above represents a step determined by our parameter α. A smaller α would result in a smaller step and a larger α results in a larger step. The direction in which the step is taken is determined by the partial derivative of J(\theta\_0,\theta\_1)*J*(*θ*0​,*θ*1​). Depending on where one starts on the graph, one could end up at different points. The image above shows us two different starting points that end up in two different places.

The gradient descent algorithm is:

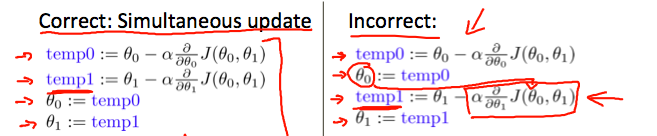
repeat until convergence:

\theta\_j := \theta\_j - \alpha \frac{\partial}{\partial \theta\_j} J(\theta\_0, \theta\_1)*θj*​:=*θj*​−*α*∂*θj*​∂​*J*(*θ*0​,*θ*1​)

where

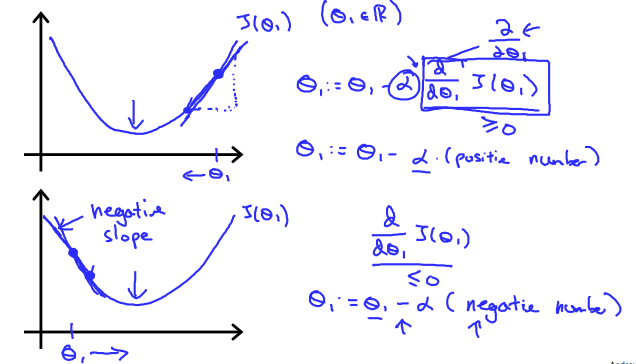
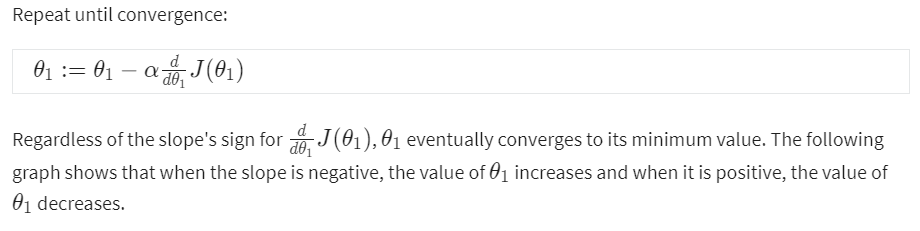
j=0,1 represents the feature index number.

At each iteration j, one should simultaneously update the parameters \theta\_1, \theta\_2,...,\theta\_n*θ*1​,*θ*2​,...,*θn*​. Updating a specific parameter prior to calculating another one on the j^{(th)}*j*(*th*) iteration would yield to a wrong implementation.

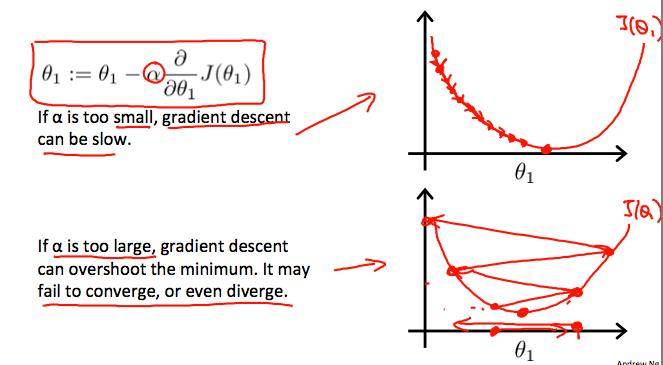


# Gradient Descent Intuition

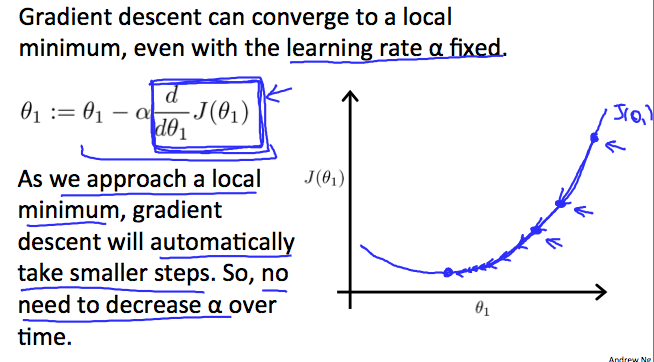
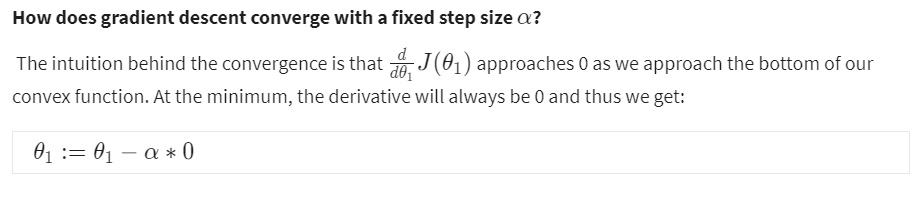
In this video we explored the scenario where we used one parameter *θ*1​ and plotted its cost function to implement a gradient descent. Our formula for a single parameter was :



On a side note, we should adjust our parameter \alpha*α* to ensure that the gradient descent algorithm converges in a reasonable time. Failure to converge or too much time to obtain the minimum value imply that our step size is wrong.

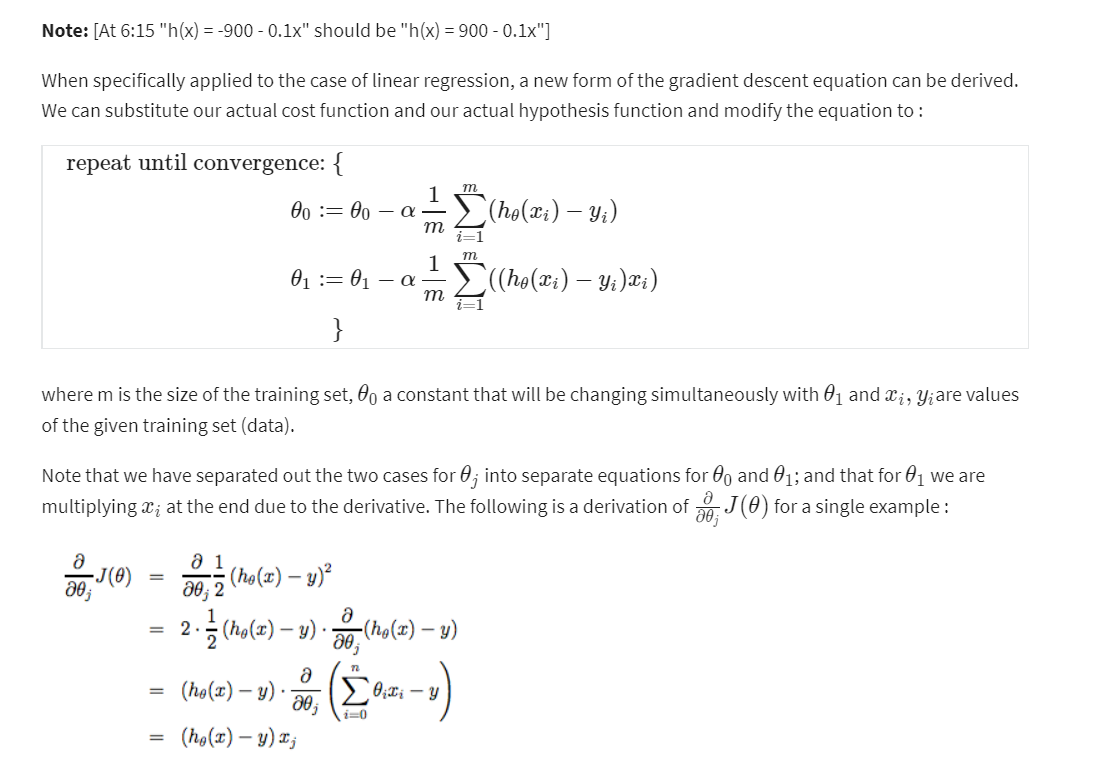


### How does gradient descent converge with a fixed step size *α*?



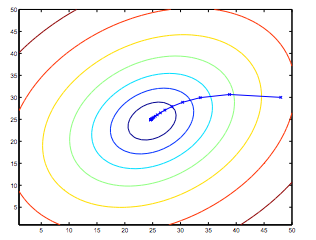
# **Gradient Descent For Linear Regression**

# Gradient Descent For Linear Regression



The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

So, this is simply gradient descent on the original cost function J. This method looks at every example in the entire training set on every step, and is called **batch gradient descent**. Note that, while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum. Indeed, J is a convex quadratic function. Here is an example of gradient descent as it is run to minimize a quadratic function.

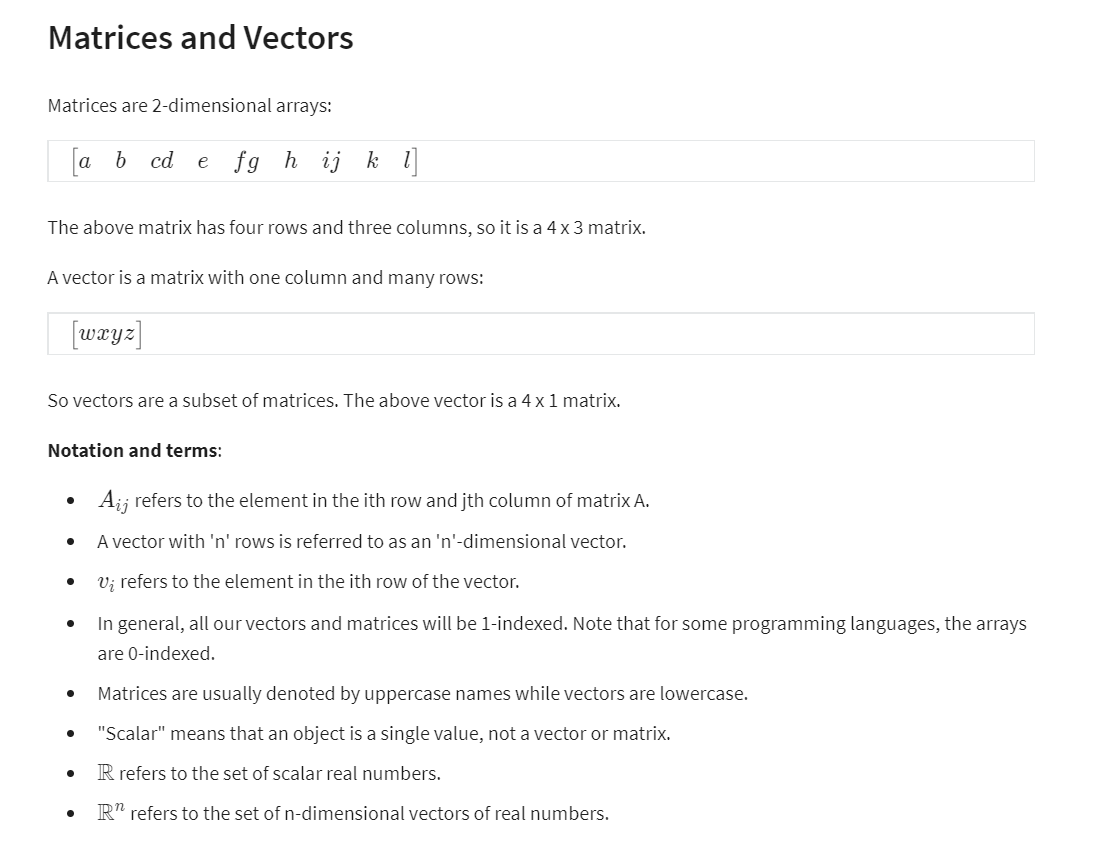


The ellipses shown above are the contours of a quadratic function. Also shown is the trajectory taken by gradient descent, which was initialized at (48,30). The x’s in the figure (joined by straight lines) mark the successive values of θ that gradient descent went through as it converged to its minimum.

## Linear Regression with One Variable

Quiz—30 min

<https://gist.github.com/mGalarnyk/cc964bea99b09e3c733b339ad3b7b019>



Sample Source Code in Octave/Matlab: Feel free to create matrices and vectors and try out different things:

% 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)

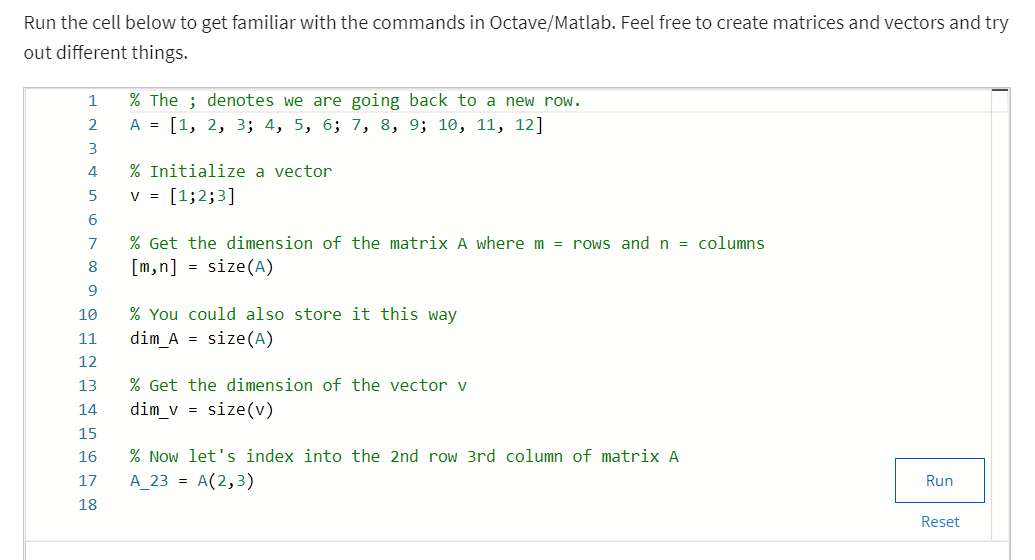


Figure : Sample Code

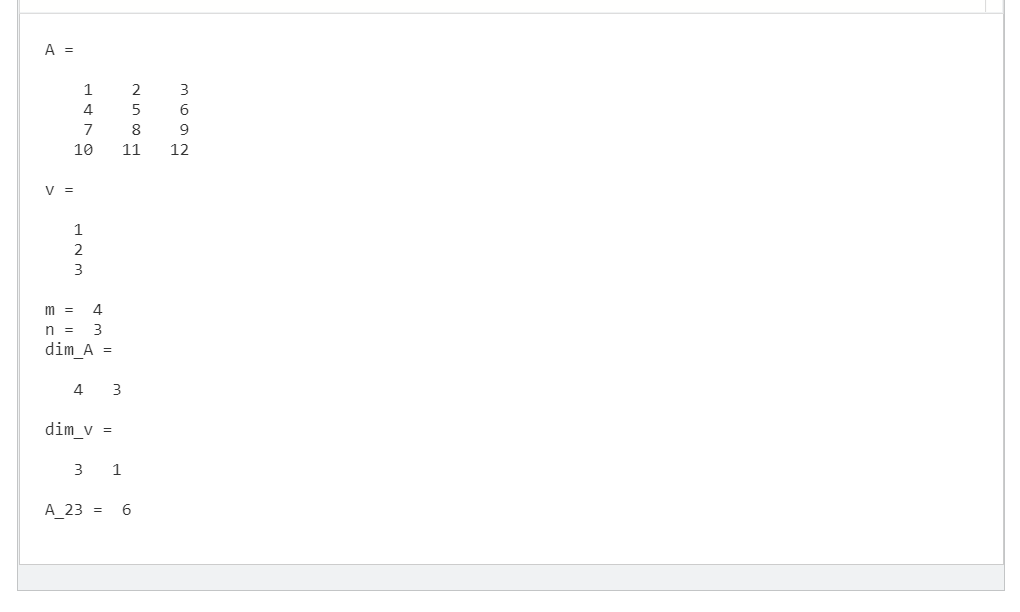
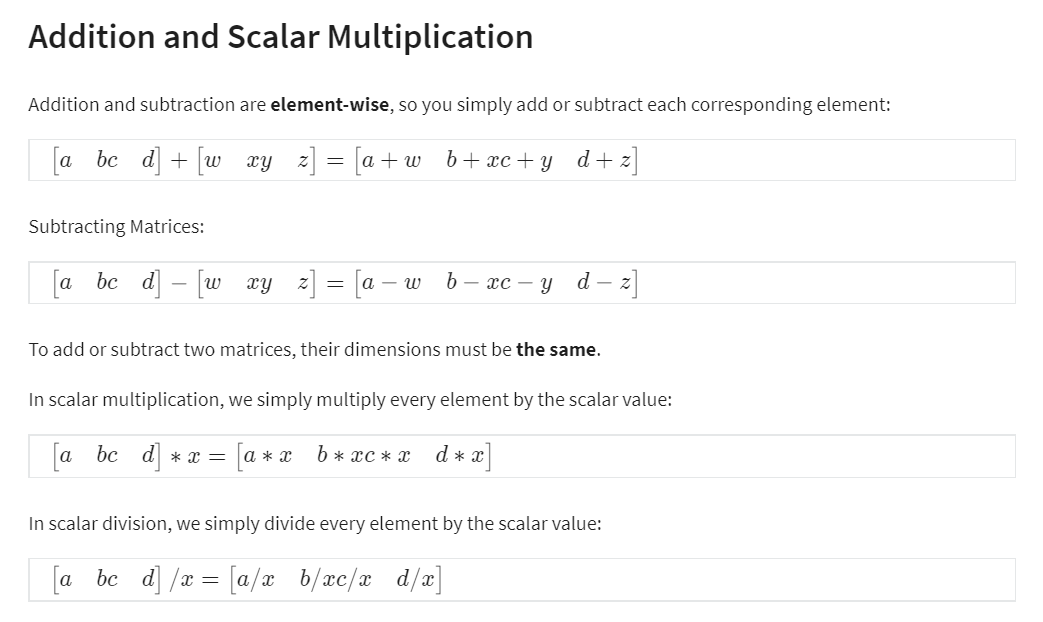


Figure : Output of Sample Code



Sample Code 2

% 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

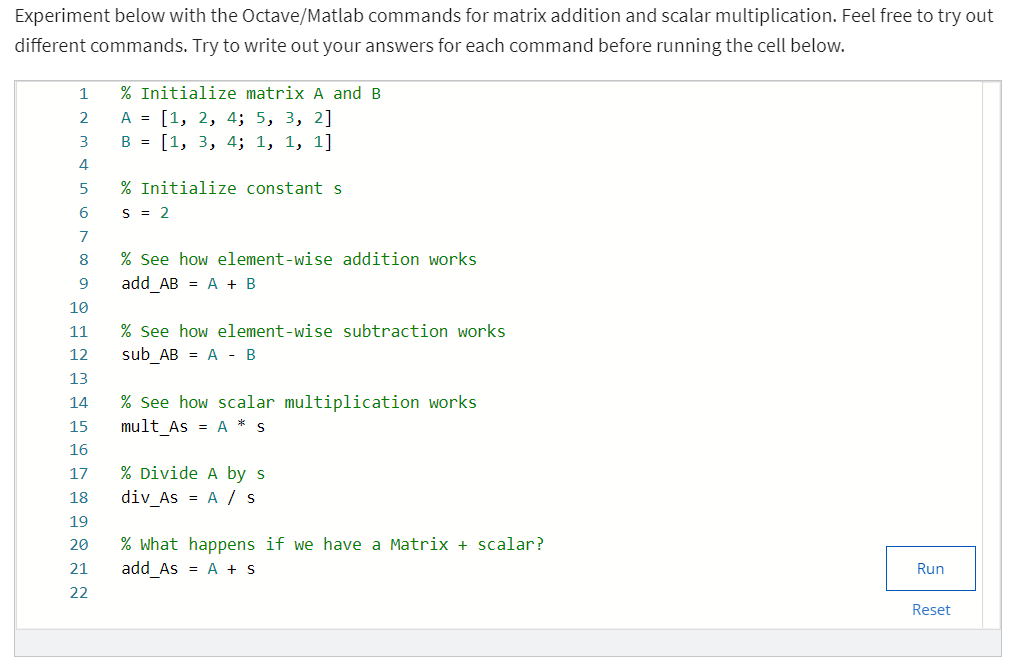


Figure : Sample Code2



Figure : Output of Sample Code2



Figure : Matrix Vector Multiplication code

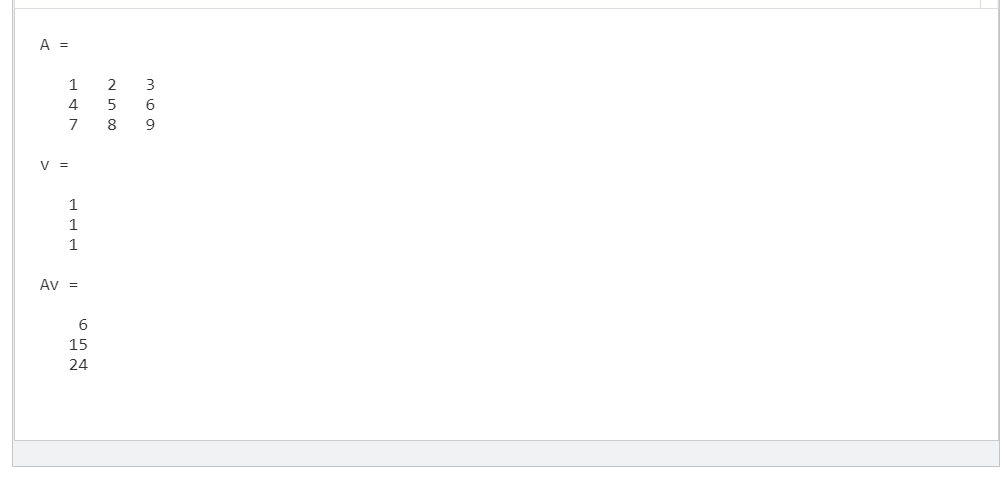


Figure : Output of Matrix Vector Multiplication Code

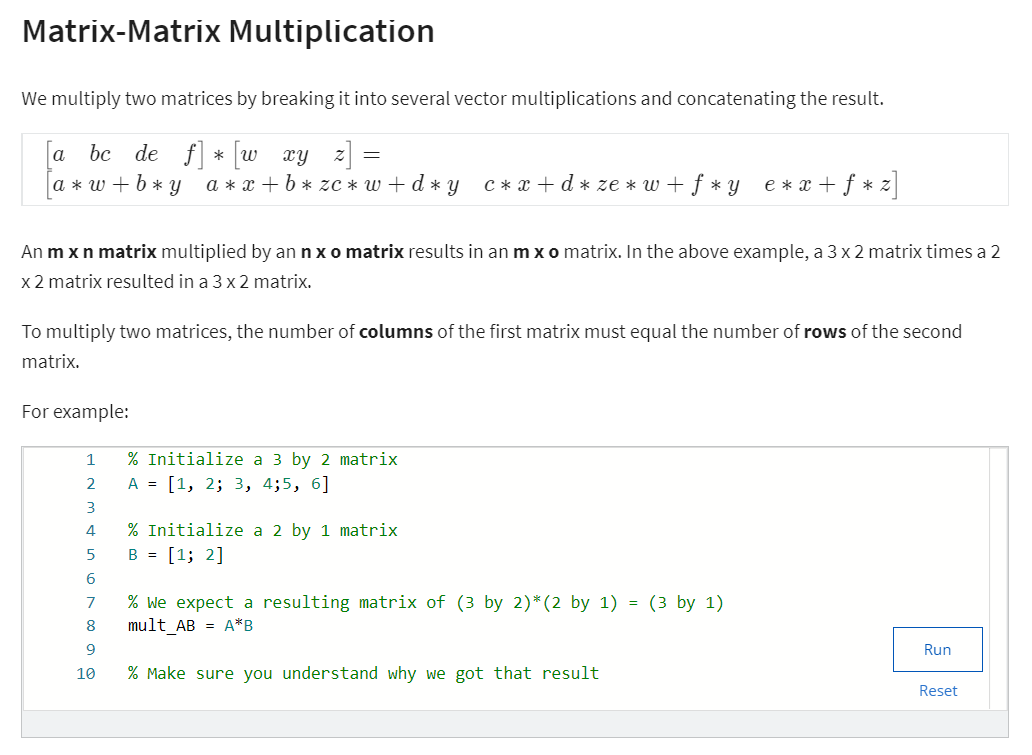


Figure : Matrix Matrix Multiplication with code

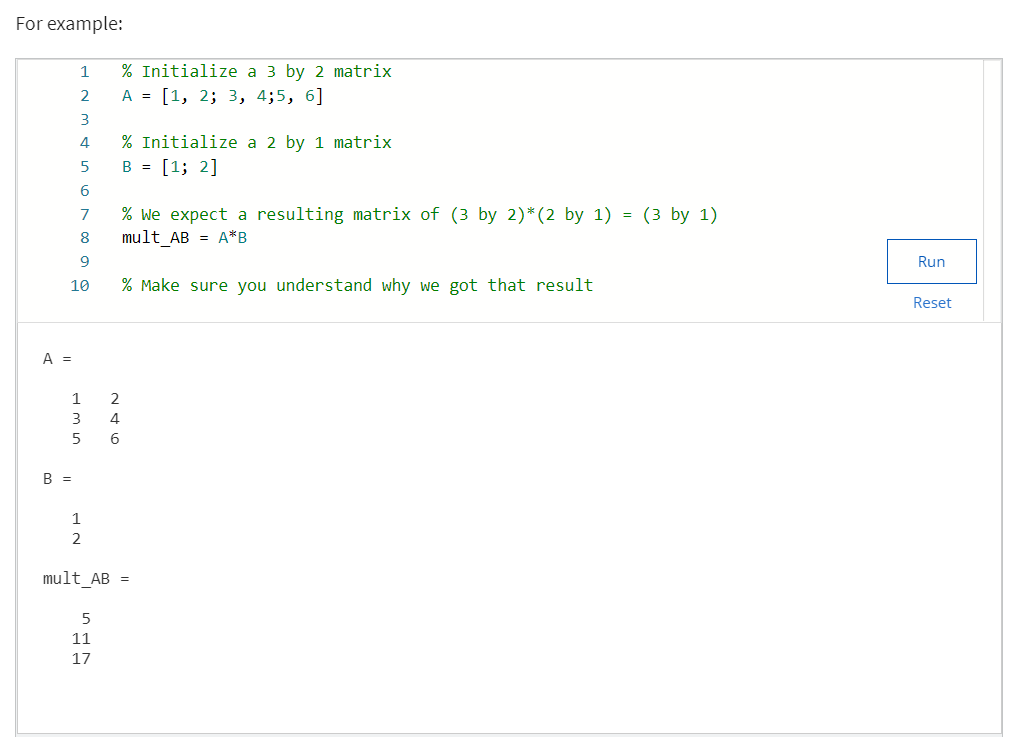
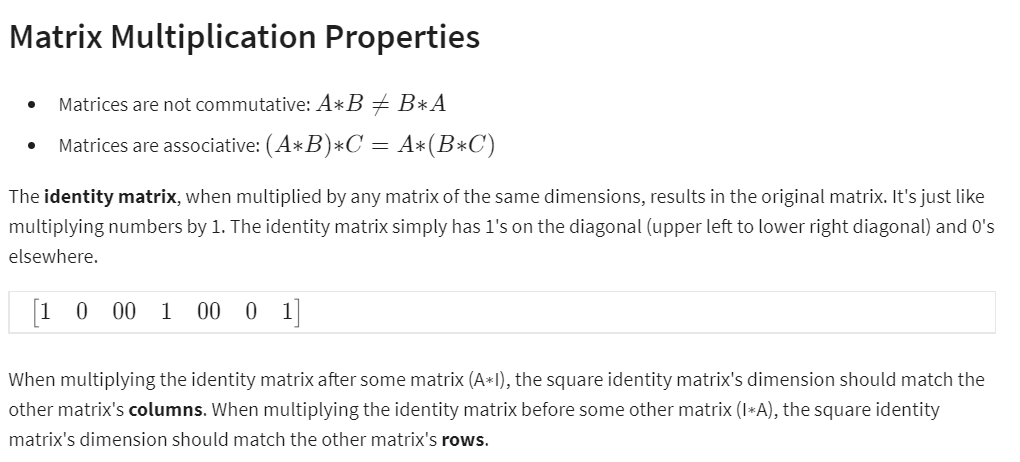


Figure : Matrix Matrix Multiplication Code and Output



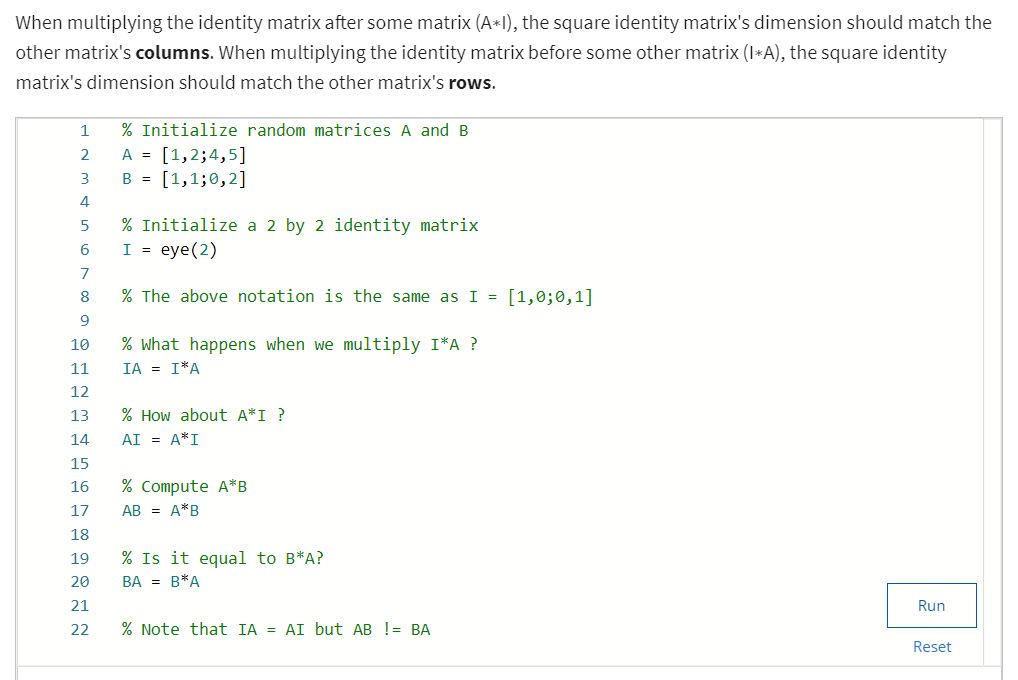
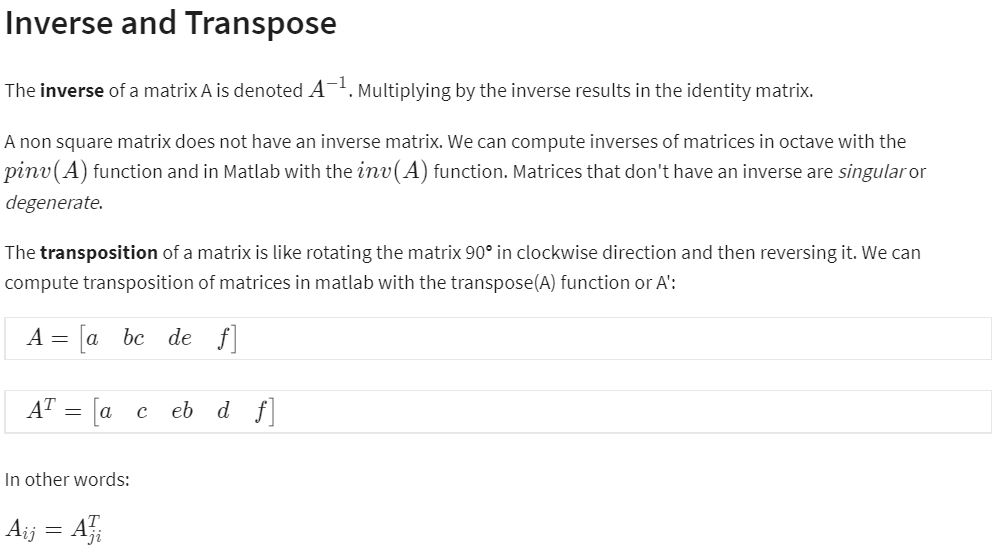


Figure : Matrix Multiplication Properties Code



Figure : Matrix Multiplication Properties Output



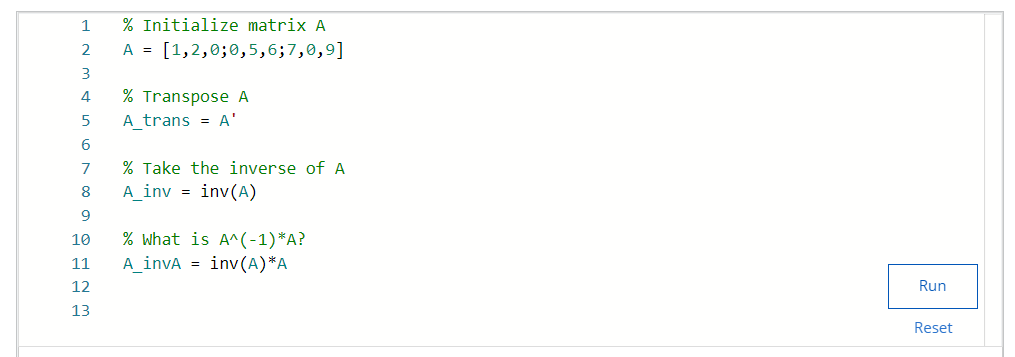


Figure : Inverse and Transpose Coding Implementation

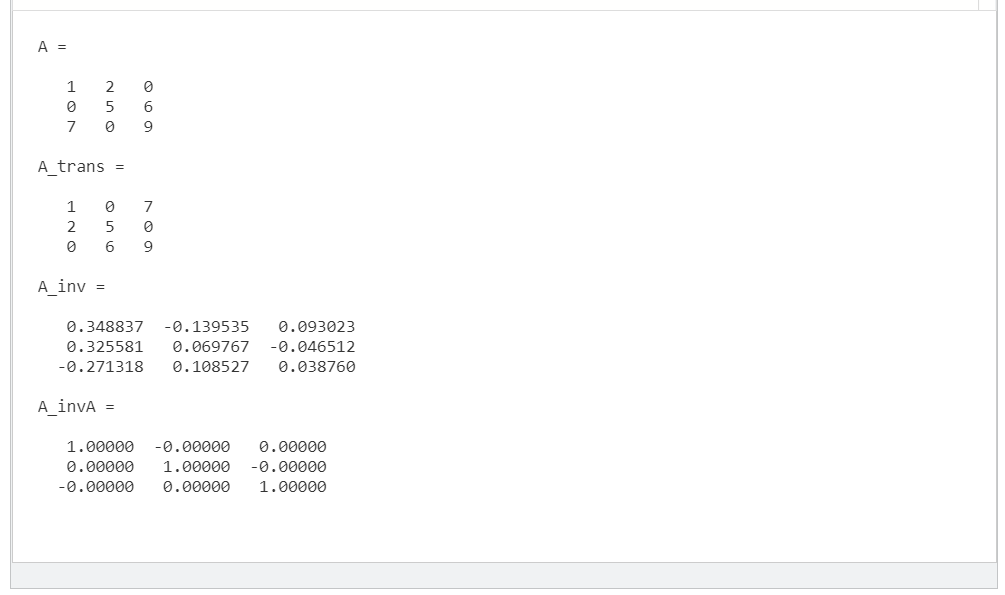
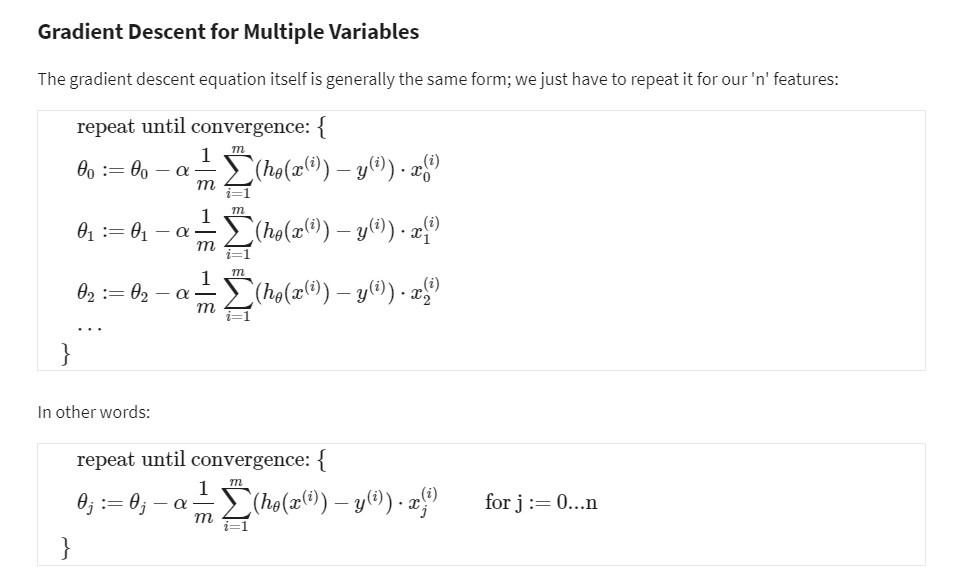


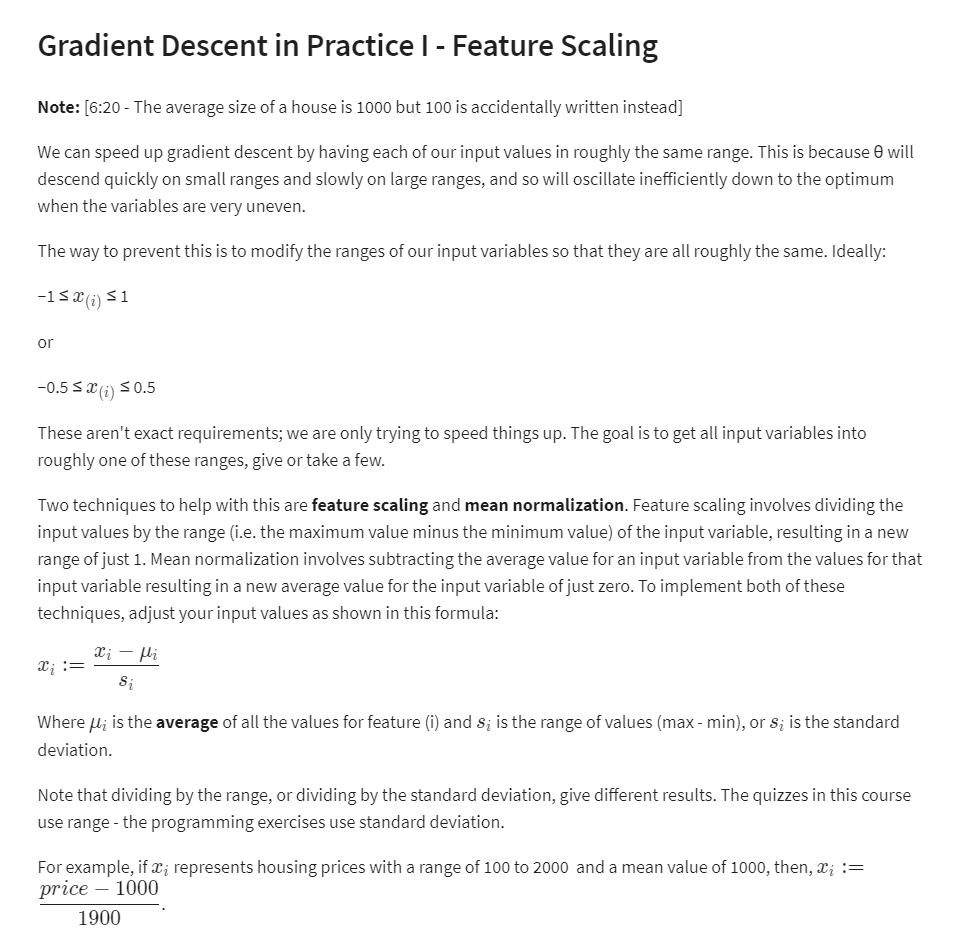
Figure : Inverse and Transpose Output



Figure : Multiple Features

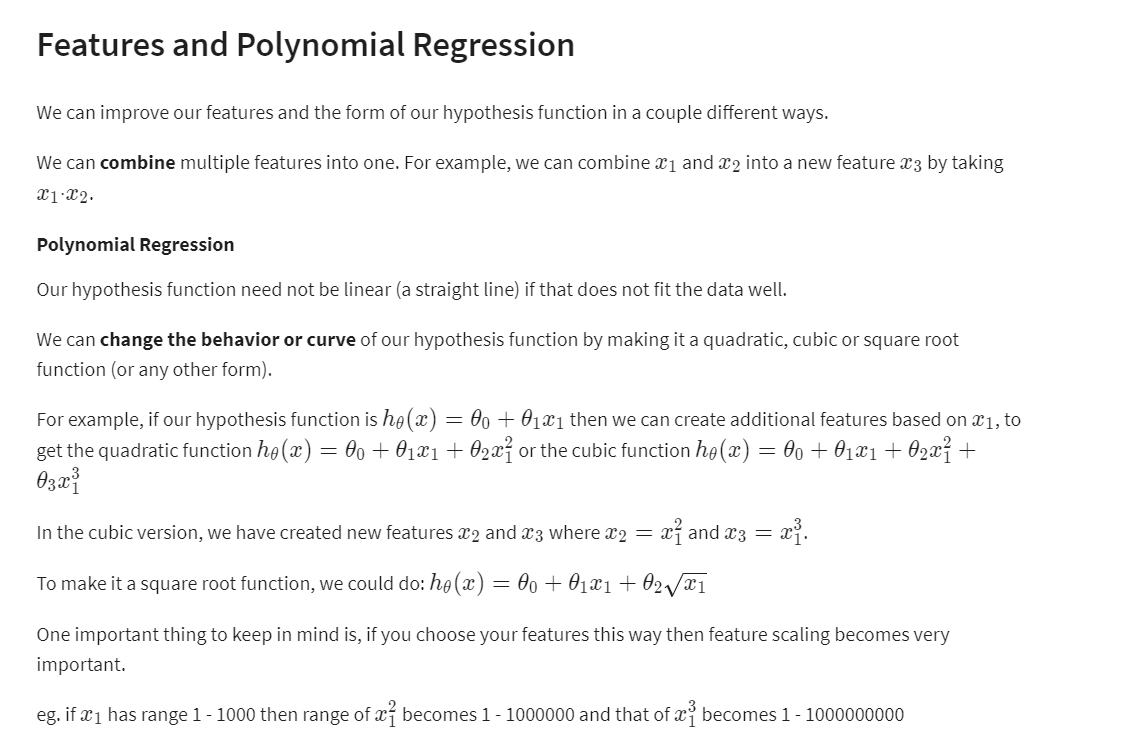


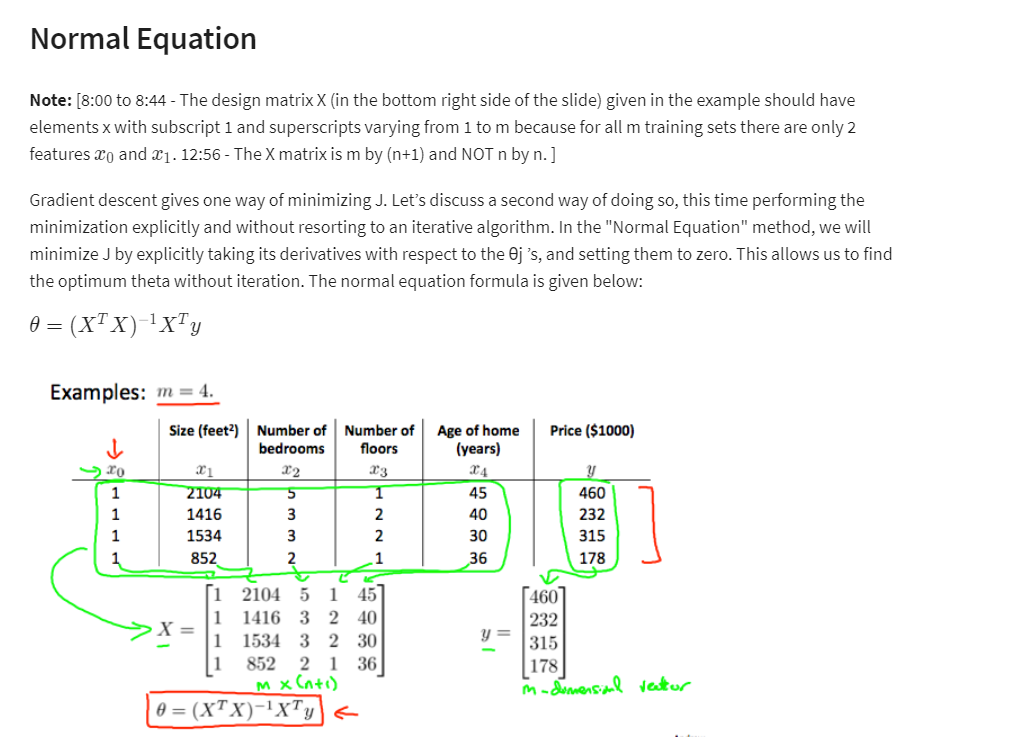


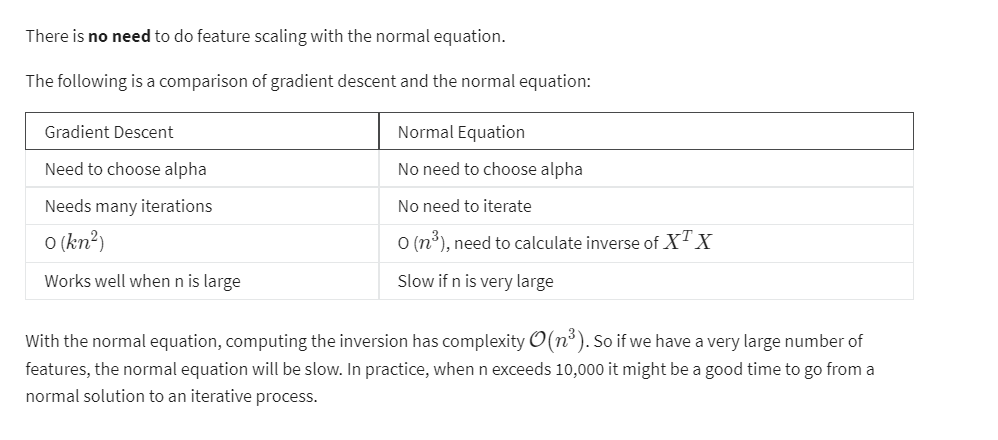


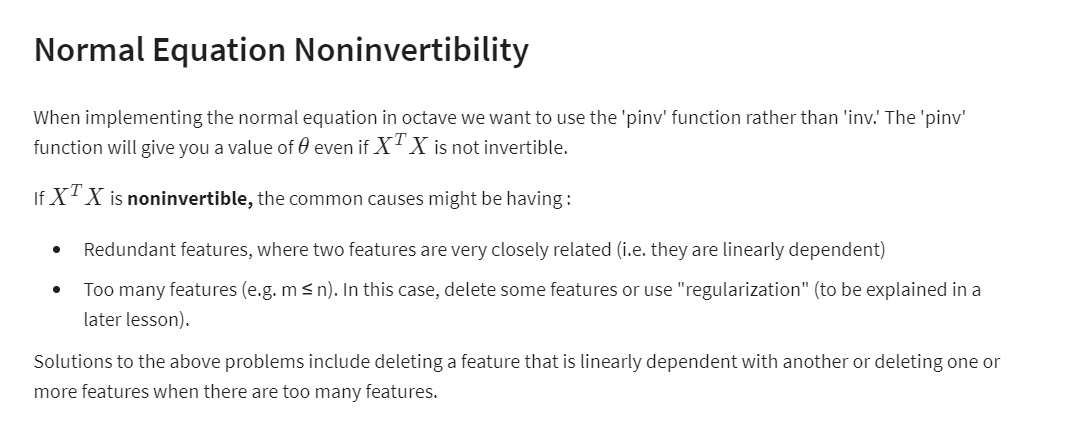
# **Gradient Descent in Practice II - Learning Rate**











# **Programming tips from Mentors**

Thank you to Machine Learning Mentor, Tom Mosher, for compiling this list

## Subject: Confused about "h(x) = theta' \* x" vs. "h(x) = X \* theta?"

### **Text:**

The lectures and exercise PDF files are based on Prof. Ng's feeling that novice programmers will adapt to for-loop techniques more readily than vectorized methods. So the videos (and PDF files) are organized toward processing one training example at a time. The course uses column vectors (in most cases), so h (a scalar for one training example) is theta' \* x.

Lower-case x typically indicates a single training example.

The more efficient vectorized techniques always use X as a matrix of all training examples, with each example as a row, and the features as columns. That makes X have dimensions of (m x n). where m is the number of training examples. This leaves us with h (a vector of all the hypothesis values for the entire training set) as X \* theta, with dimensions of (m x 1).

X (as a matrix of all training examples) is denoted as upper-case X.

Throughout this course, dimensional analysis is your friend.

## Subject: Tips from the Mentors: submit problems and fixing program errors

### **Text:**

This post contains some frequently-used tips about the course, and to help get your programs working correctly.

### **The Most Important Tip:**

Search the forum before posting a new question. If you've got a question, the chances are that someone else has already posted it, and received an answer. Save time for yourself and the Forum users by searching for topics before posting a new one.

### **Running your scripts:**

At the Octave/Matlab command line, you do not need to include the ".m" portion of the script file name. If you include the ".m", you'll get an error message about an invalid indexing operation. So, run the Exercise 1 script by typing just "ex1" at the command line.

You also do not need to include parenthesis () when using the submit script. Just type "submit".

You cannot execute your functions by simply typing the name. All of the functions you will work on require a set of parameter values, enter between a set of parenthesis. Your three methods of testing your code are:

1 - use an exercise script, such as "ex1"

2 - use a Unit Test (see below) where you type-in the entire command line including the parameters.

3 - use the submit script.

### **Making the grader happy:**

The submit grader uses a different test case than what is in the PDF file. These test cases use a different size of data set and are more sensitive to small errors than the ex test cases. Your code must work correctly with any size of data set.

Your functions must handle the general case. This means:

- You should avoid using hard-coded array indexes.

- You should avoid having fixed-length arrays and matrices.

It is very common for students to think that getting the same answer as listed in the PDF file means they should get full credit from the grader. This is a false hope. The PDF file is just one test case. The grader uses a different test case.

Also, the grader does not like your code to send any additional outputs to the workspace. So, every line of code should end with a semicolon.

### **Getting Help:**

When you want help from the Forum community, please use this two-step procedure:

1 - Search the Forum for keywords that relate to your problem. Searching by the function name is a good start.

2 - If you don't find a suitable thread, then do this:

2a - Find the unit tests for that exercise (see below), and run the appropriate test. Attempt to debug your code.

2b - Take a screen capture of your whole console workspace (including the command line), and post it to the forum, along with any other useful information (computer type, Octave/Matlab version, other tests you've tried, etc).

### Debugging:

If your code runs but gives the wrong answers, you can insert a "keyboard" command in your script, just before the function ends. This will cause the program to exit to the debugger, so you can inspect all your variables from the command line. This often is very helpful in analysing math errors, or trying out what commands to use to implement your function.

There are additional test cases and tutorials listed in pinned threads under "All Course Discussions". The test cases are especially helpful in debugging in situations where you get the expected output in ex but get no points or an error when submitting.

### Unit Tests:

Each programming assignment has a "Discussions" area in the Forum. In this section you can often find "unit tests". These are additional test cases, which give you a command to type, and provides the expected results. It is always a good idea to test your functions using the unit tests before submitting to the grader.

If you run a unit test and do not get the correct results, you can most easily get help on the forums by posting a screen capture of your workspace - including the command line you entered, and the results.

### Having trouble submitting your work to the grader?:

- This section will need to be supplemented with info appropriate to the new submission system. If you run the submit script and get a message that your identity can't be verified, be sure that you have logged-in using your Coursera account email and your Programming Assignment submission password.

- If you get the message "submit undefined", first check that you are in the working directory where you extracted the files from the ZIP archive. Use "cd" to get there if necessary.

- If the "submit undefined" error persists, or any other "function undefined" messages appear, try using the "addpath(pwd)" command to add your present working directory (pwd) to the Octave execution path.

-If the submit script crashes with an error message, please see the thread "Mentor tips for submitting your work" under "All Course Discussions".

-The submit script does not ask for what part of the exercise you want to submit. It automatically grades any function you have modified.

### Found some errata in the course materials?

This course material has been used for many previous sessions. Most likely all of the errata has been discovered, and it's all documented in the 'Errata' section under 'Supplementary Materials'. Please check there before posting errata to the Forum.

Error messages with fmincg()

The "short-circuit" warnings are due to use a change in the syntax for conditional expressions (| and & vs || and &&) in the newer versions of Matlab. You can edit the fmincg.m file and the warnings may be resolved.

### Warning messages about "automatic broadcasting"?

See [this](https://www.gnu.org/software/octave/doc/interpreter/Broadcasting.html) link for info.

### Warnings about "divide by zero"

These are normal in some of the exercises, and do not represent a problem in your function. You can ignore them - Octave senses the issue and substitutes a +Inf or -Inf value so your program continues to execute.

Week 3:

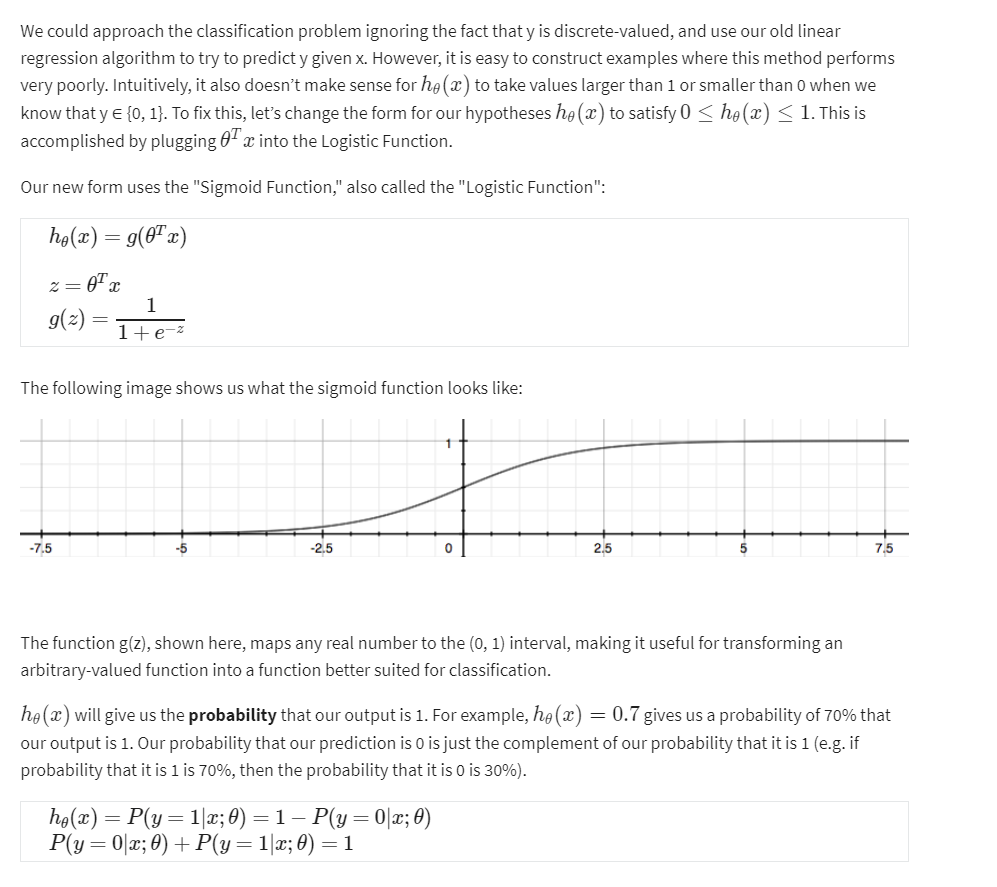
# **Classification**

# Classification

To attempt classification, one method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. However, this method doesn't work well because classification is not actually a linear function.

The classification problem is just like the regression problem, except that the values we now want to predict take on only a small number of discrete values. For now, we will focus on the **binary classification** **problem** in which y can take on only two values, 0 and 1. (Most of what we say here will also generalize to the multiple-class case.) For instance, if we are trying to build a spam classifier for email, then x^{(i)}*x*(*i*) may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. Hence, y∈{0,1}. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols “-” and “+.” Given x^{(i)}*x*(*i*), the corresponding y^{(i)}*y*(*i*) is also called the label for the training example.

# **Hypothesis Representation**



# **Decision Boundary**

# Decision Boundary

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

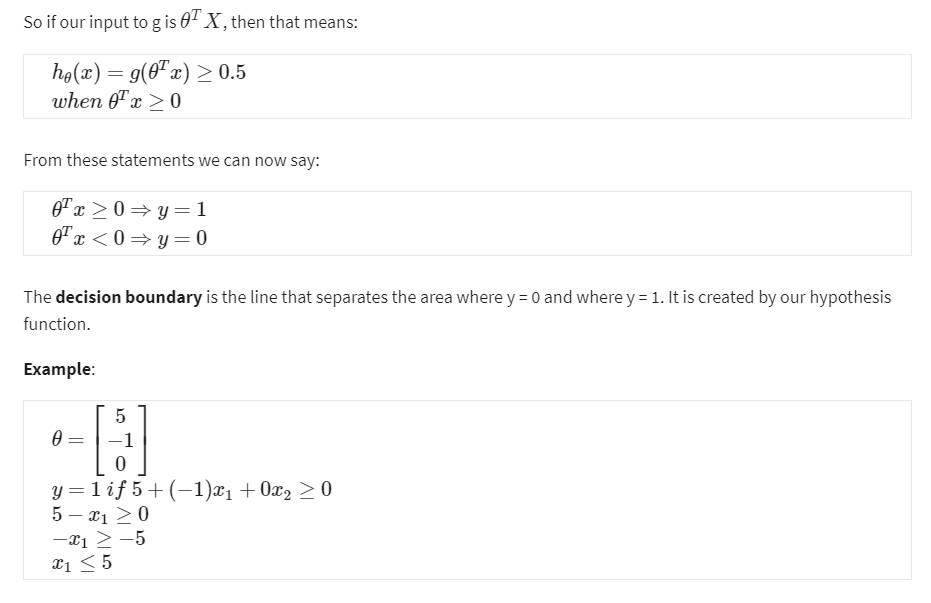


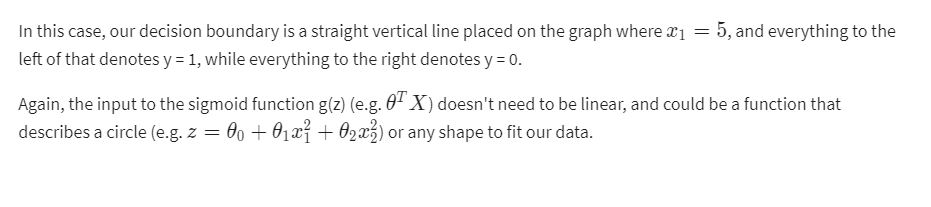
The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5:



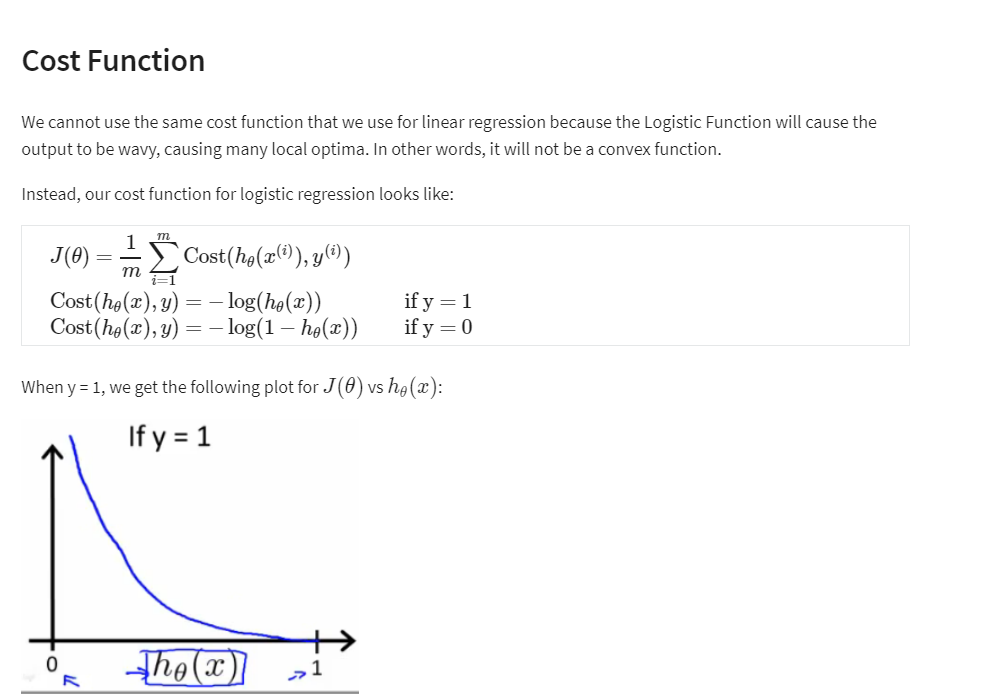
Remember.

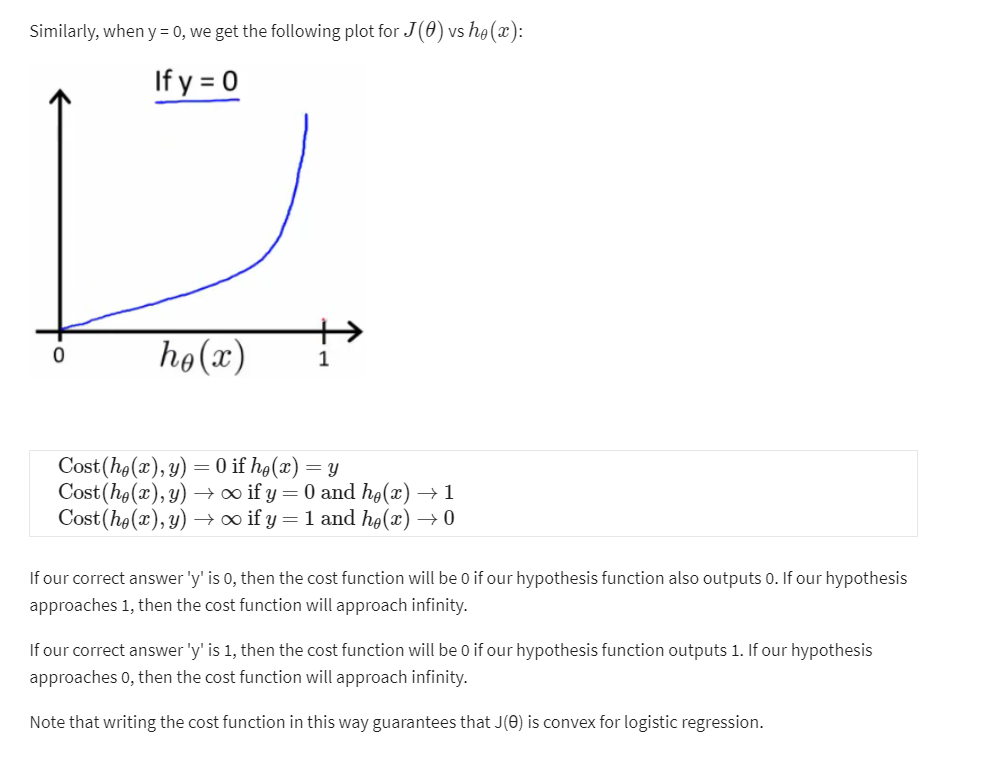


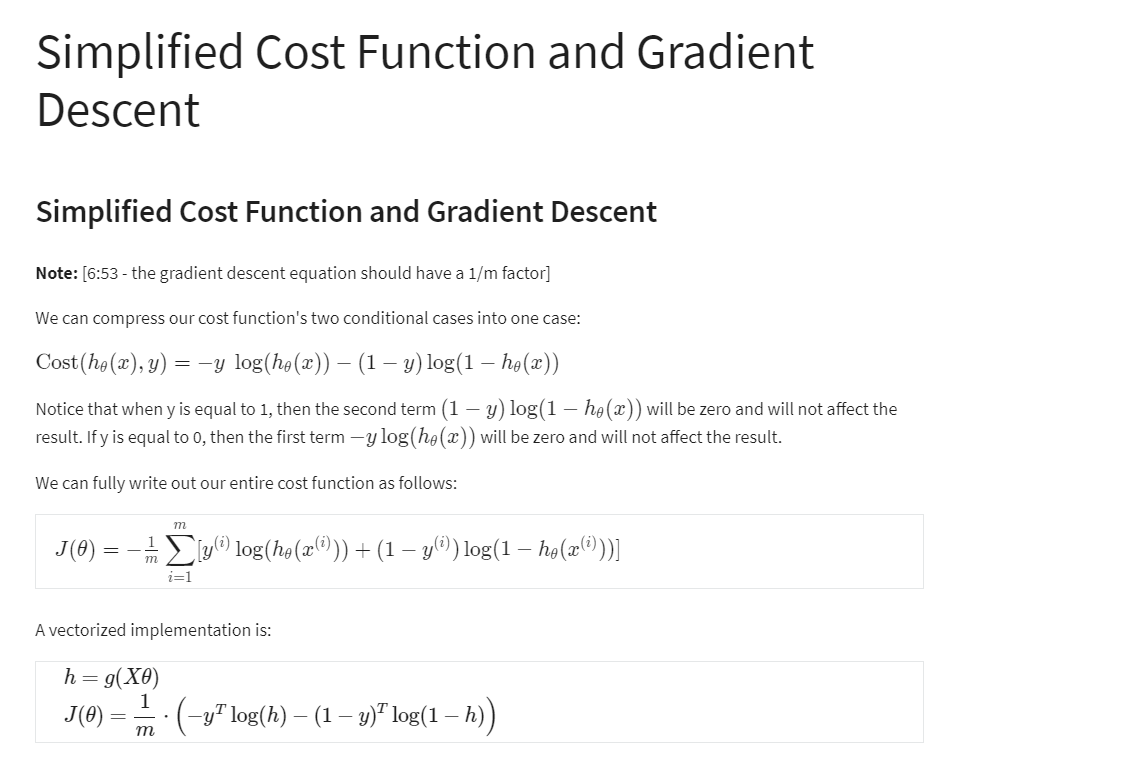




# **Cost Function**

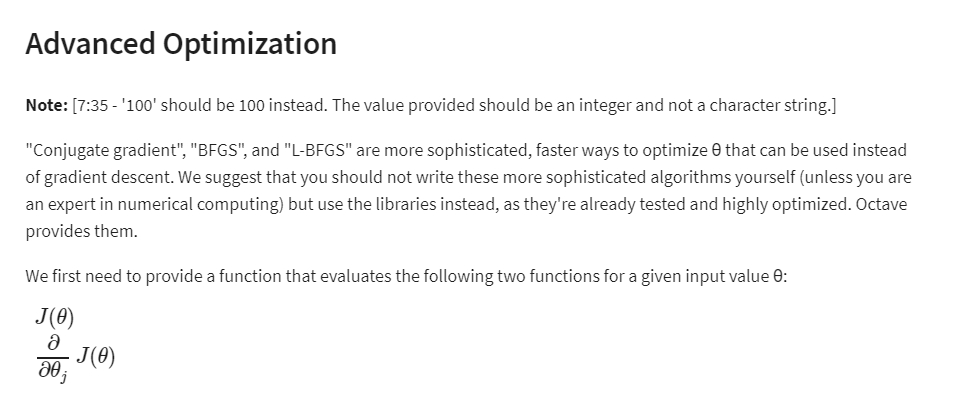


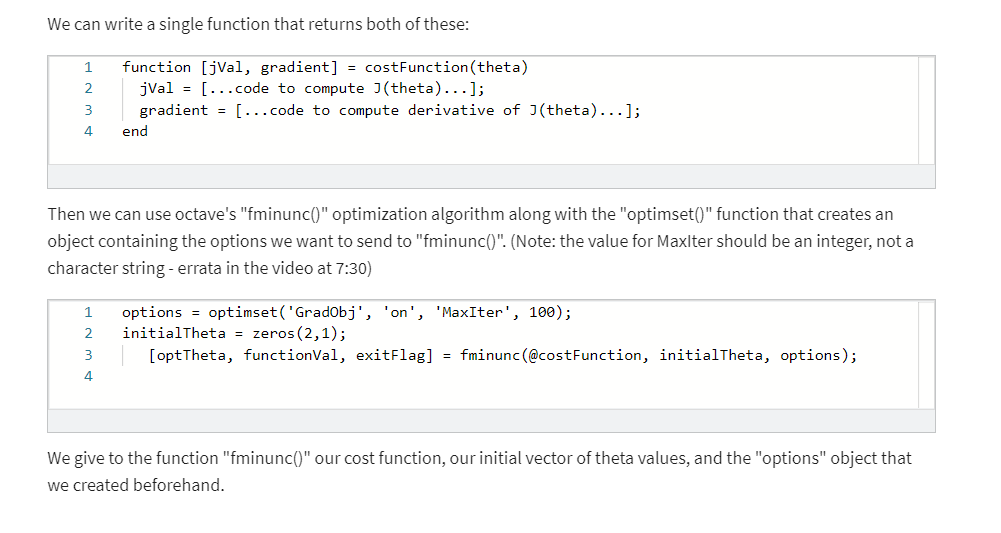






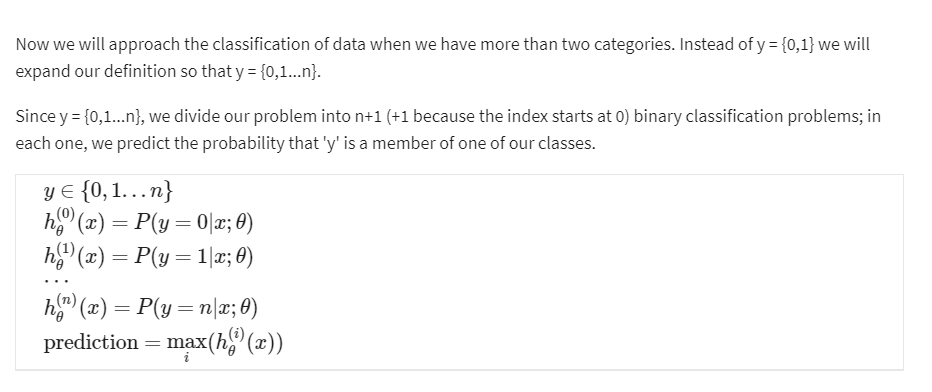
# **Advanced Optimization**

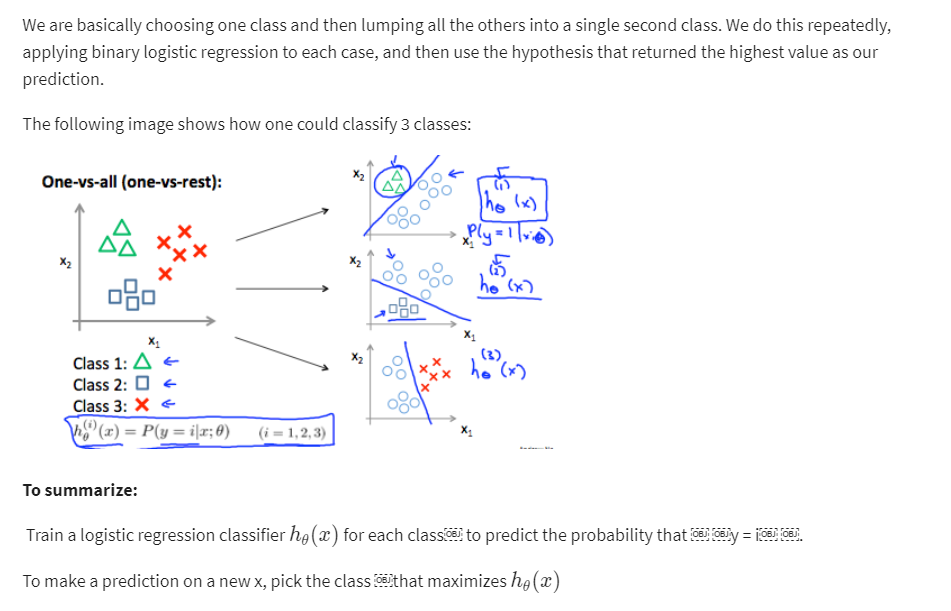




# **Multiclass Classification**

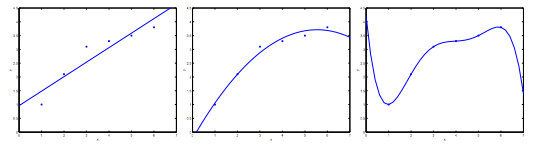
# **Multiclass Classification: One-vs-all**





# **The Problem of Overfitting**

Consider the problem of predicting y from x ∈ R. The leftmost figure below shows the result of fitting a y = *θ*0​+*θ*1​*x* to a dataset. We see that the data doesn’t really lie on straight line, and so the fit is not very good.



Instead, if we had added an extra feature *x*2 , and fit *y*=*θ*0​+*θ*1​*x*+*θ*2​*x*2 , then we obtain a slightly better fit to the data (See middle figure). Naively, it might seem that the more features we add, the better. However, there is also a danger in adding too many features: The rightmost figure is the result of fitting a 5*th*order polynomial *y*=∑5*j*=0 ​*θj* ​*xj*. We see that even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor of, say, housing prices (y) for different living areas (x). Without formally defining what these terms mean, we’ll say the figure on the left shows an instance of **underfitting**—in which the data clearly shows structure not captured by the model—and the figure on the right is an example of **overfitting**.

Underfitting, or high bias, is when the form of our hypothesis function h maps poorly to the trend of the data. It is usually caused by a function that is too simple or uses too few features. At the other extreme, overfitting, or high variance, is caused by a hypothesis function that fits the available data but does not generalize well to predict new data. It is usually caused by a complicated function that creates a lot of unnecessary curves and angles unrelated to the data.

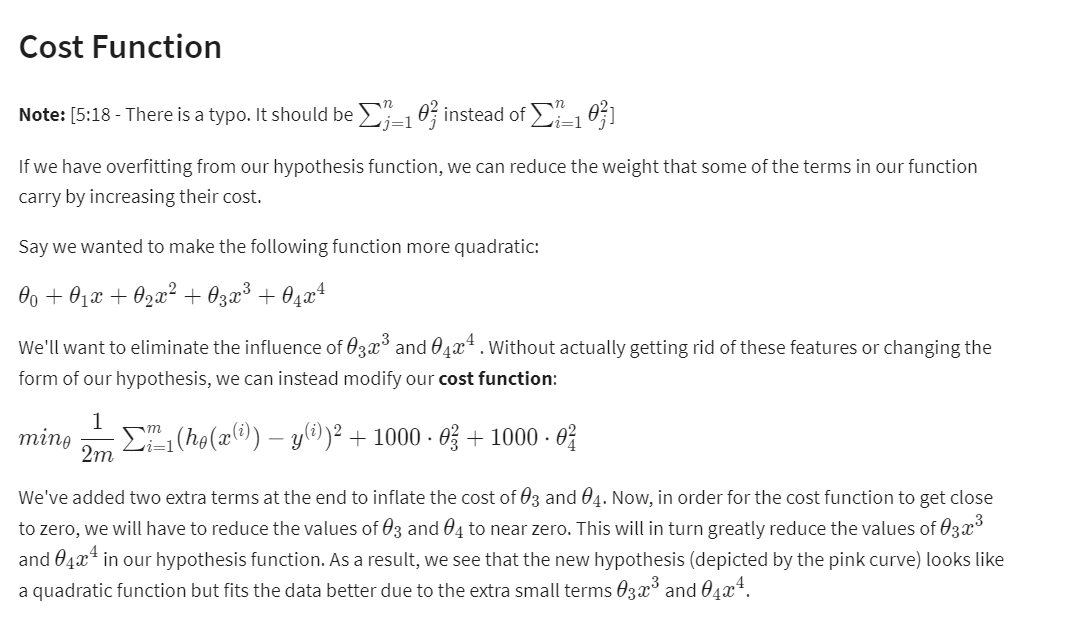
This terminology is applied to both linear and logistic regression. There are two main options to address the issue of overfitting:

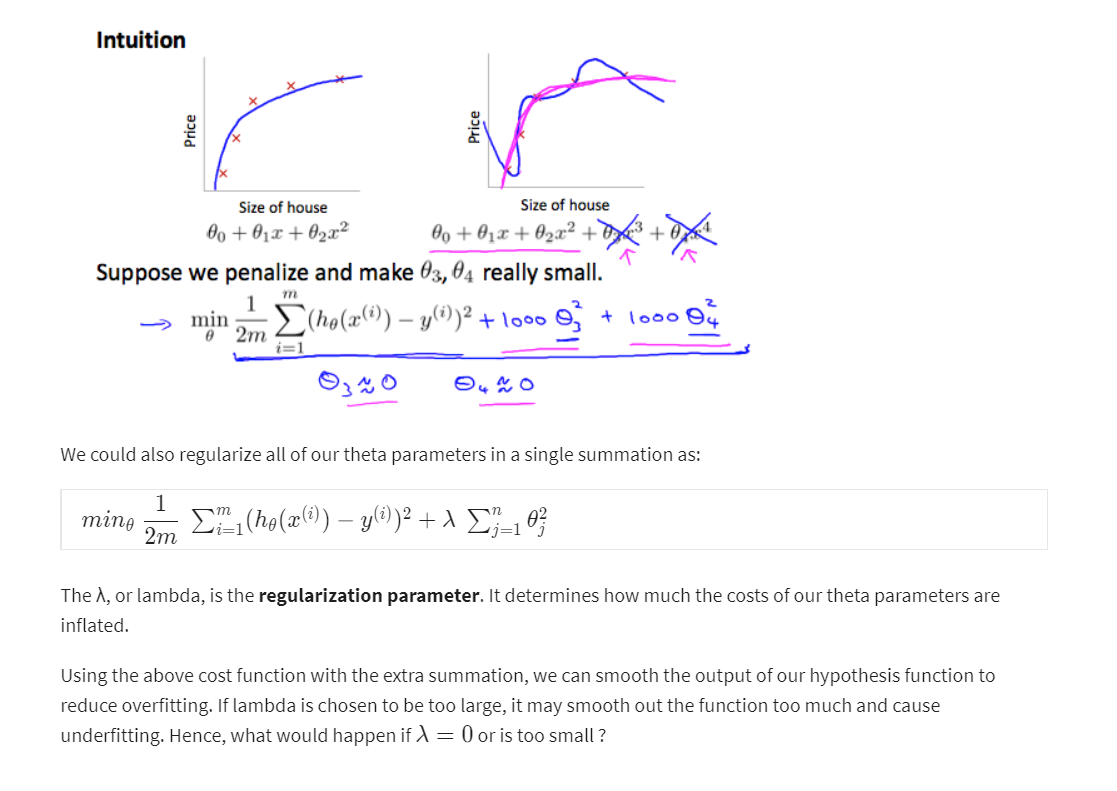
1) Reduce the number of features:

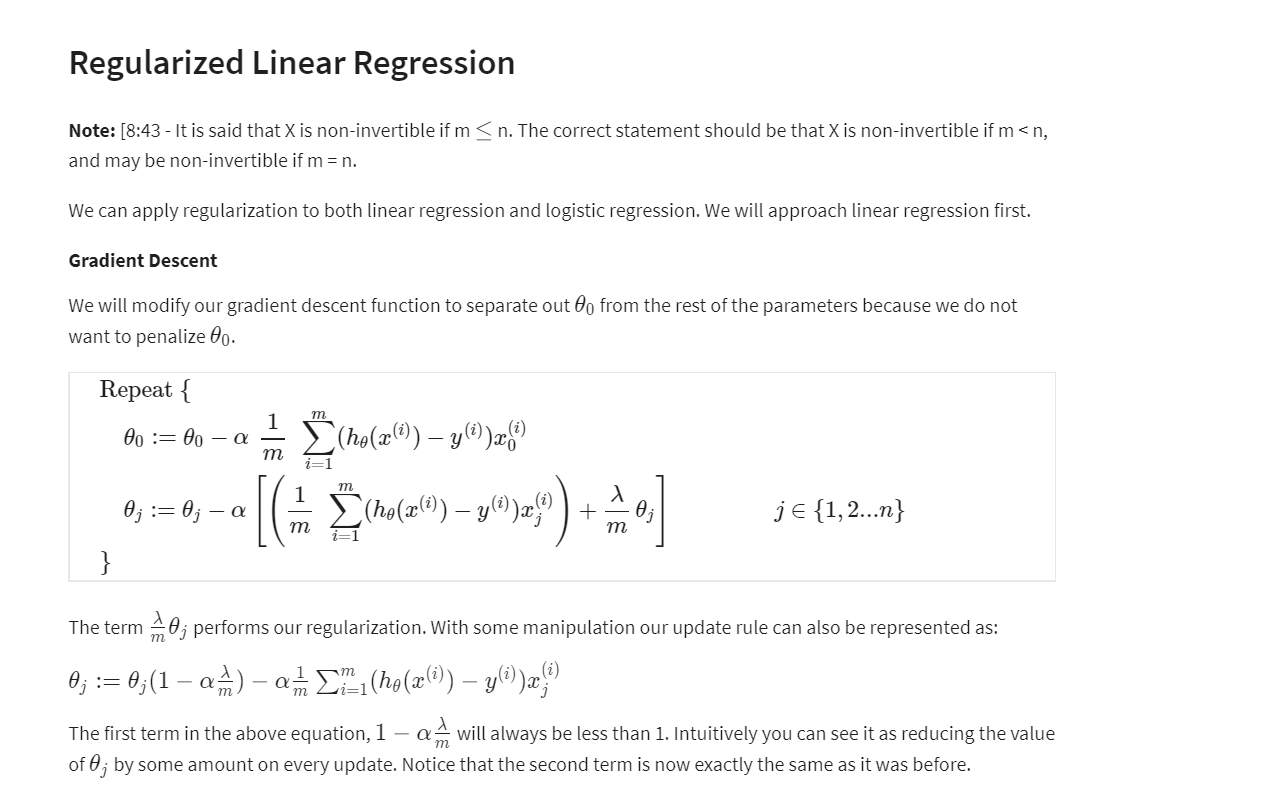
* Manually select which features to keep.
* Use a model selection algorithm (studied later in the course).

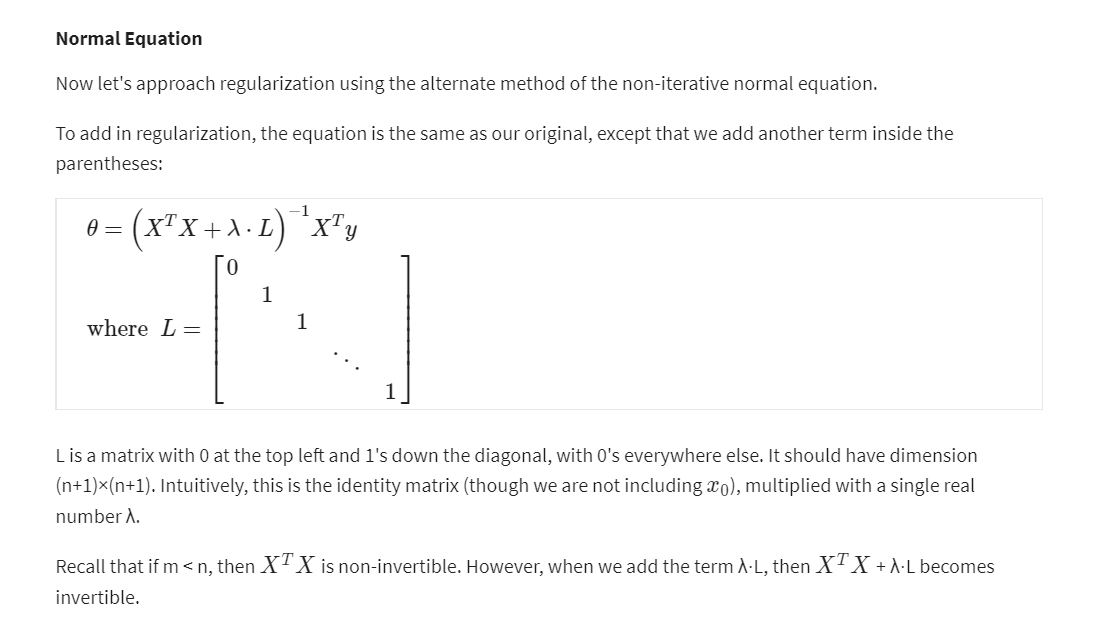
2) Regularization

* Keep all the features, but reduce the magnitude of parameters *θj*​.
* Regularization works well when we have a lot of slightly useful features.



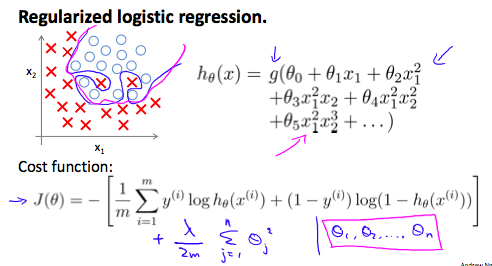


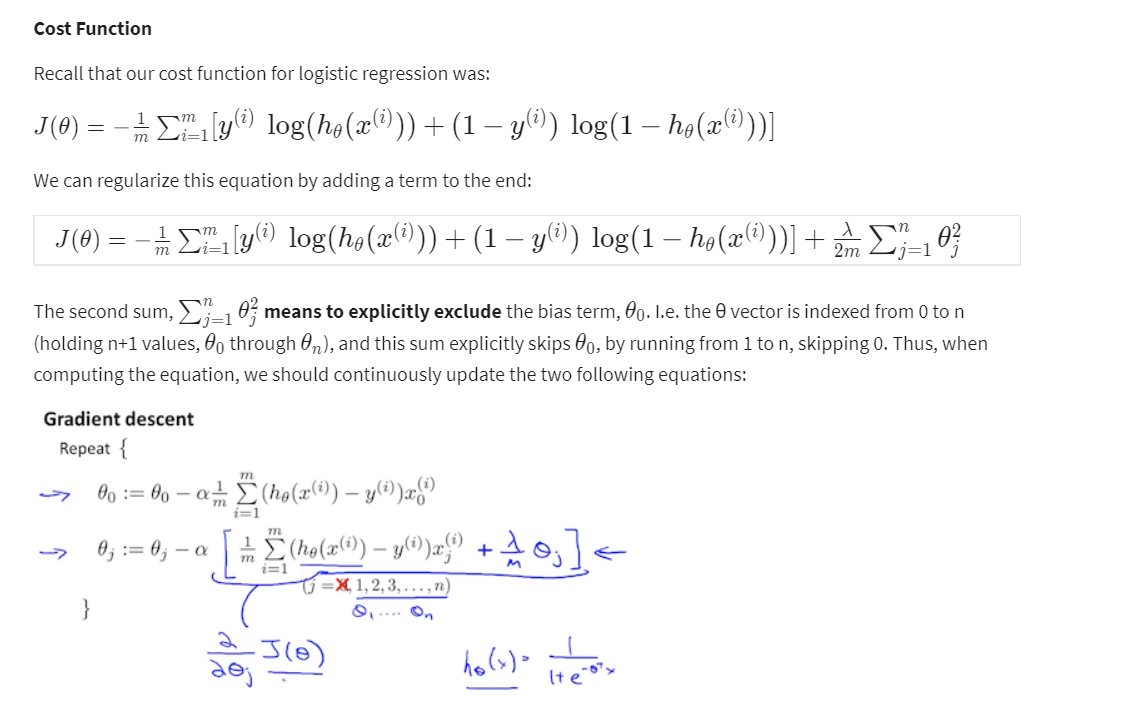




# Regularized Logistic Regression

We can regularize logistic regression in a similar way that we regularize linear regression. As a result, we can avoid overfitting. The following image shows how the regularized function, displayed by the pink line, is less likely to overfit than the non-regularized function represented by the blue line:





### Cost Function

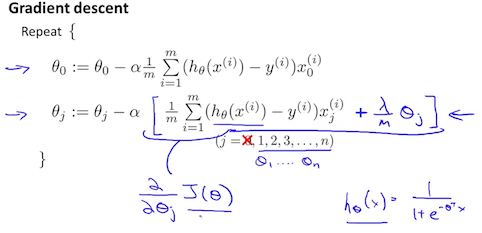
Recall that our cost function for logistic regression was:

J(\theta) = - \frac{1}{m} \sum\_{i=1}^m \large[ y^{(i)}\ \log (h\_\theta (x^{(i)})) + (1 - y^{(i)})\ \log (1 - h\_\theta(x^{(i)})) \large]*J*(*θ*)=−*m*1​∑*i*=1*m*​[*y*(*i*) log(*hθ*​(*x*(*i*)))+(1−*y*(*i*)) log(1−*hθ*​(*x*(*i*)))]

We can regularize this equation by adding a term to the end:

|  |
| --- |
| J(\theta) = - \frac{1}{m} \sum\_{i=1}^m \large[ y^{(i)}\ \log (h\_\theta (x^{(i)})) + (1 - y^{(i)})\ \log (1 - h\_\theta(x^{(i)}))\large] + \frac{\lambda}{2m}\sum\_{j=1}^n \theta\_j^2*J*(*θ*)=−*m*1​∑*i*=1*m*​[*y*(*i*) log(*hθ*​(*x*(*i*)))+(1−*y*(*i*)) log(1−*hθ*​(*x*(*i*)))]+2*mλ*​∑*j*=1*n*​*θj*2​ |

The second sum, \sum\_{j=1}^n \theta\_j^2∑*j*=1*n*​*θj*2​ **means to explicitly exclude** the bias term, \theta\_0*θ*0​. I.e. the θ vector is indexed from 0 to n (holding n+1 values, \theta\_0*θ*0​ through \theta\_n*θn*​), and this sum explicitly skips \theta\_0*θ*0​, by running from 1 to n, skipping 0. Thus, when computing the equation, we should continuously update the two following equations:



Week 4

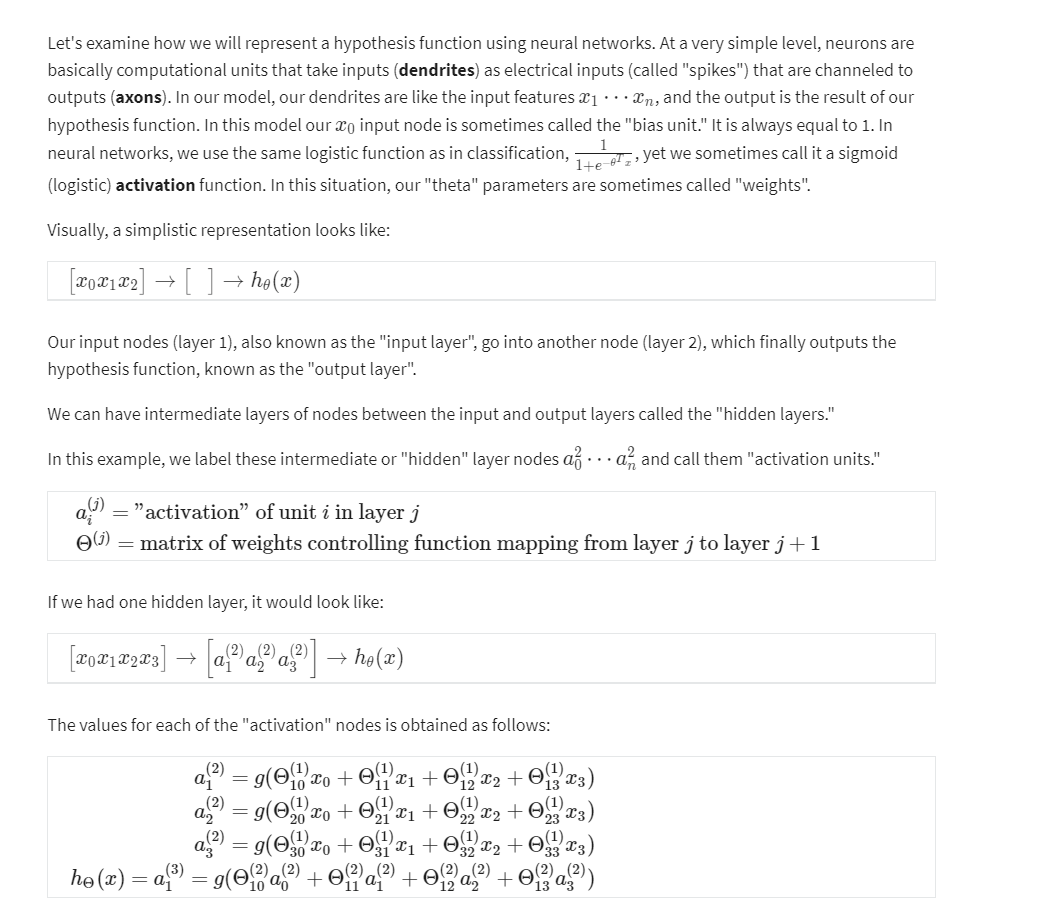
Motivations

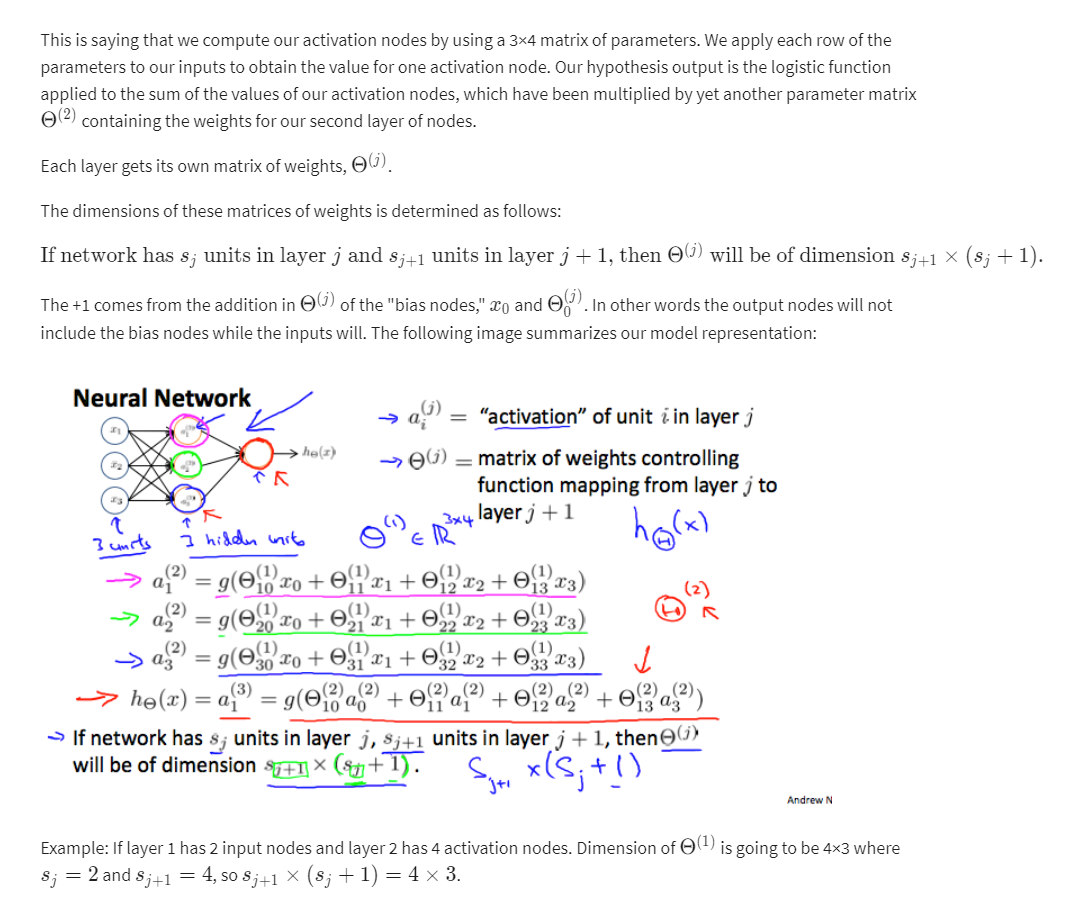
* Non-linear Hypotheses
* Neurons and the Brain

Neural Networks

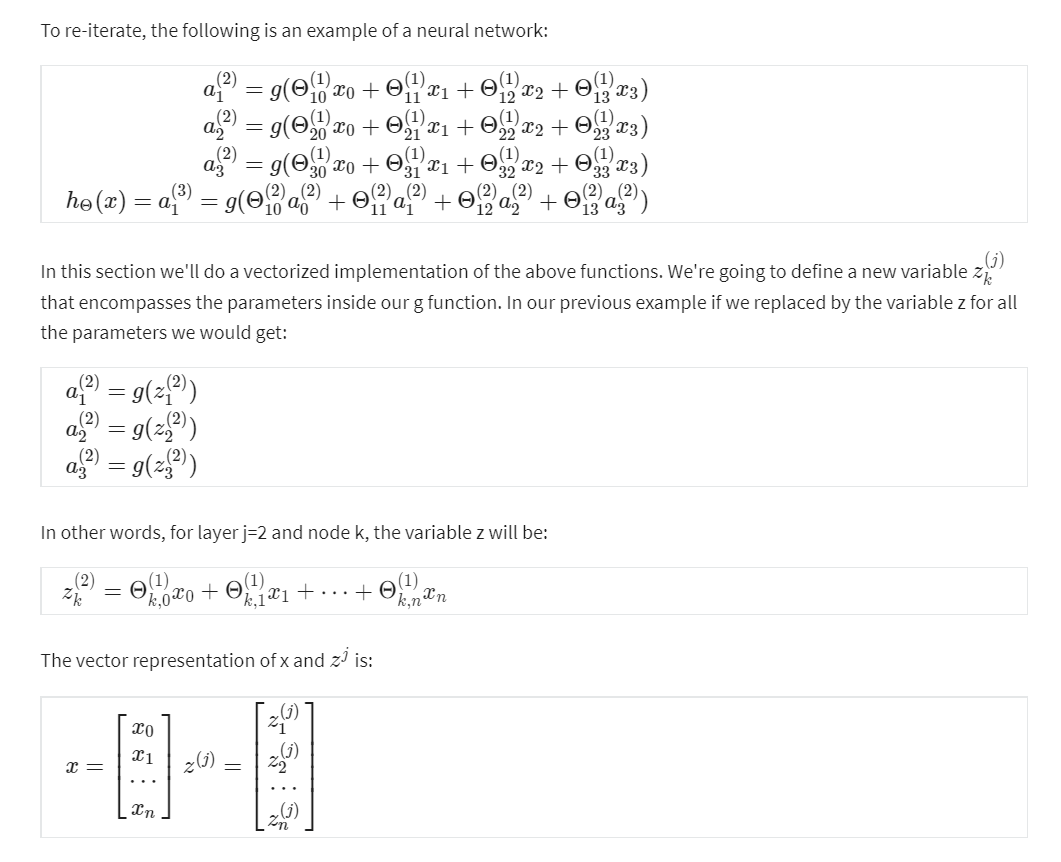
* Model Representation I
* Model Representation II

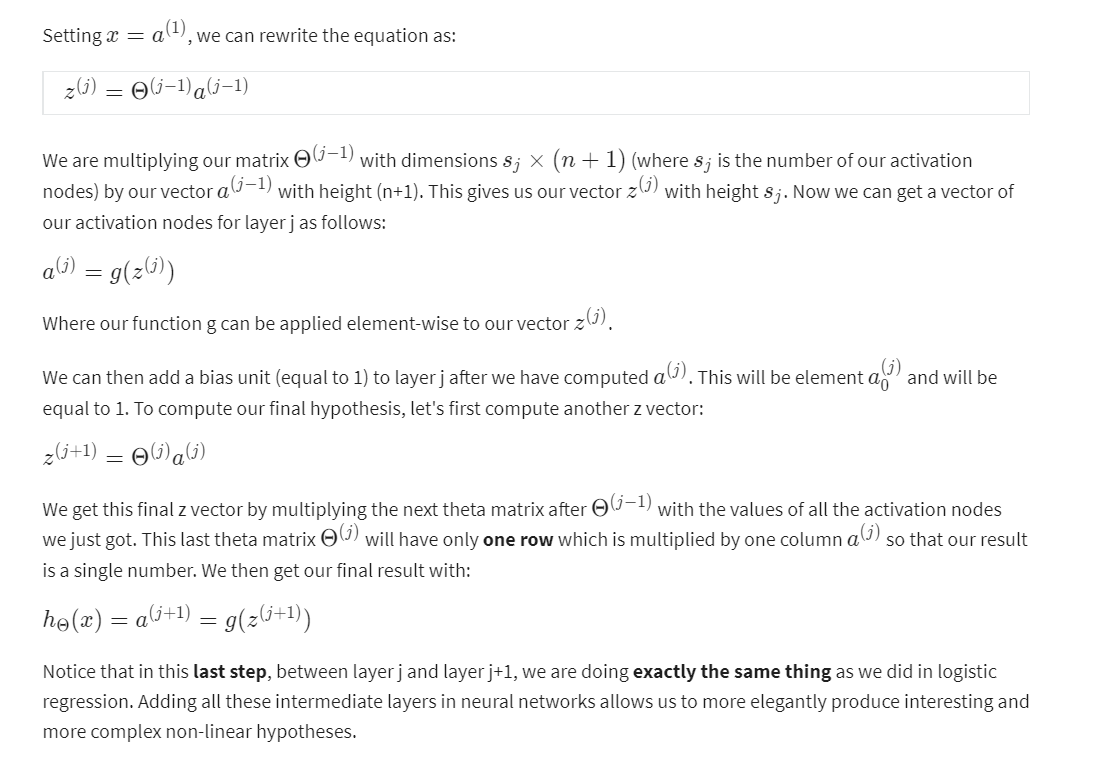
# Model Representation I

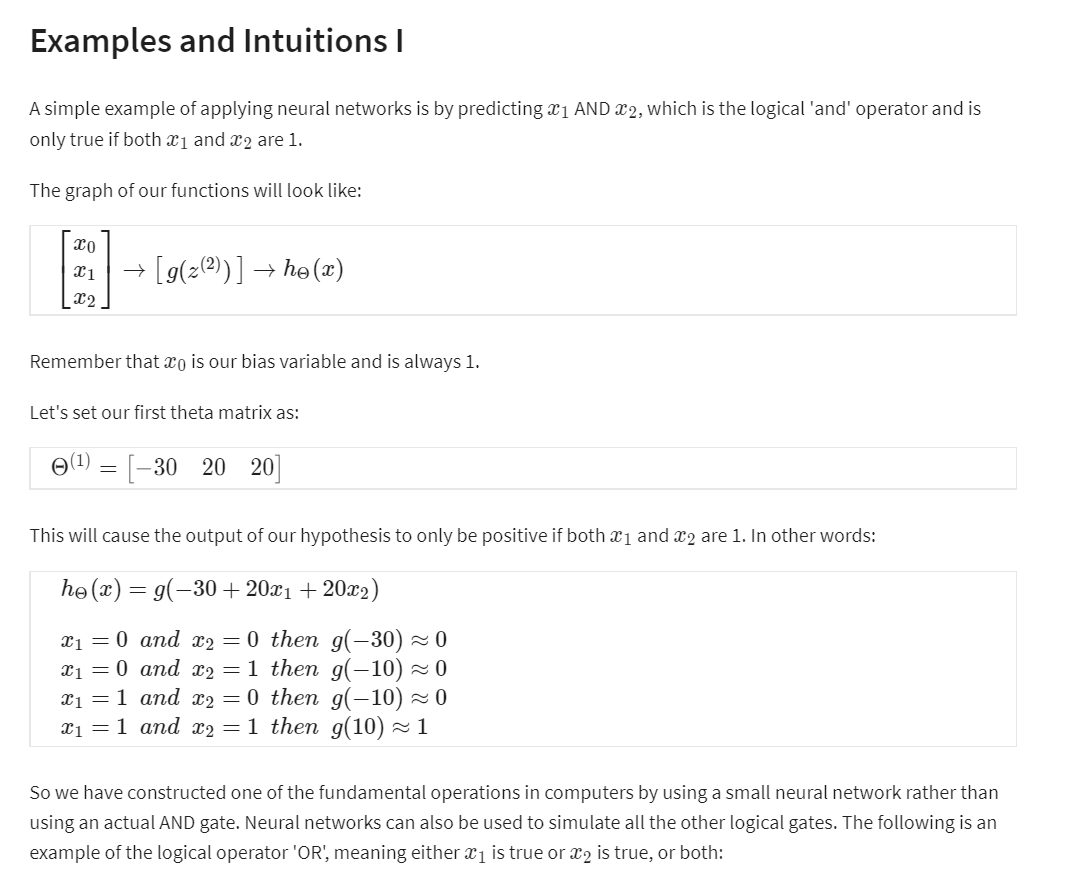


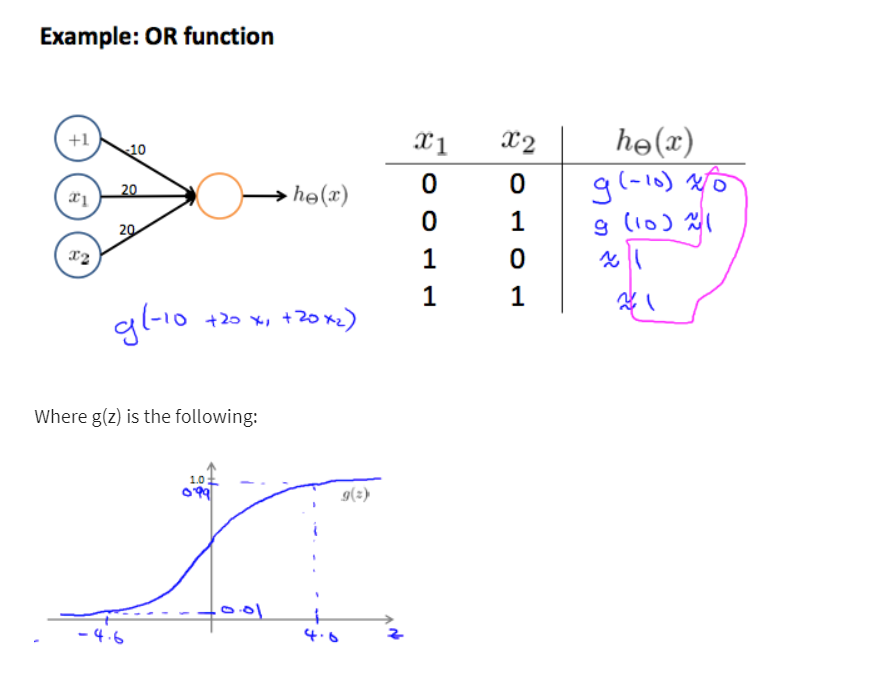


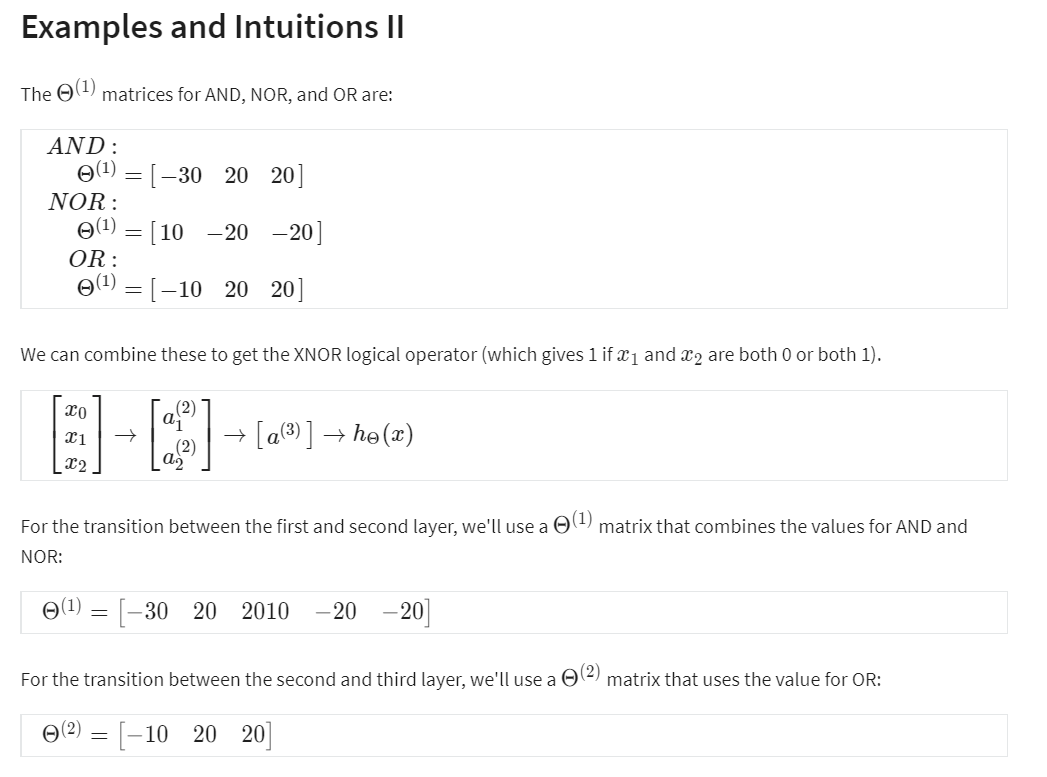
# Model Representation II

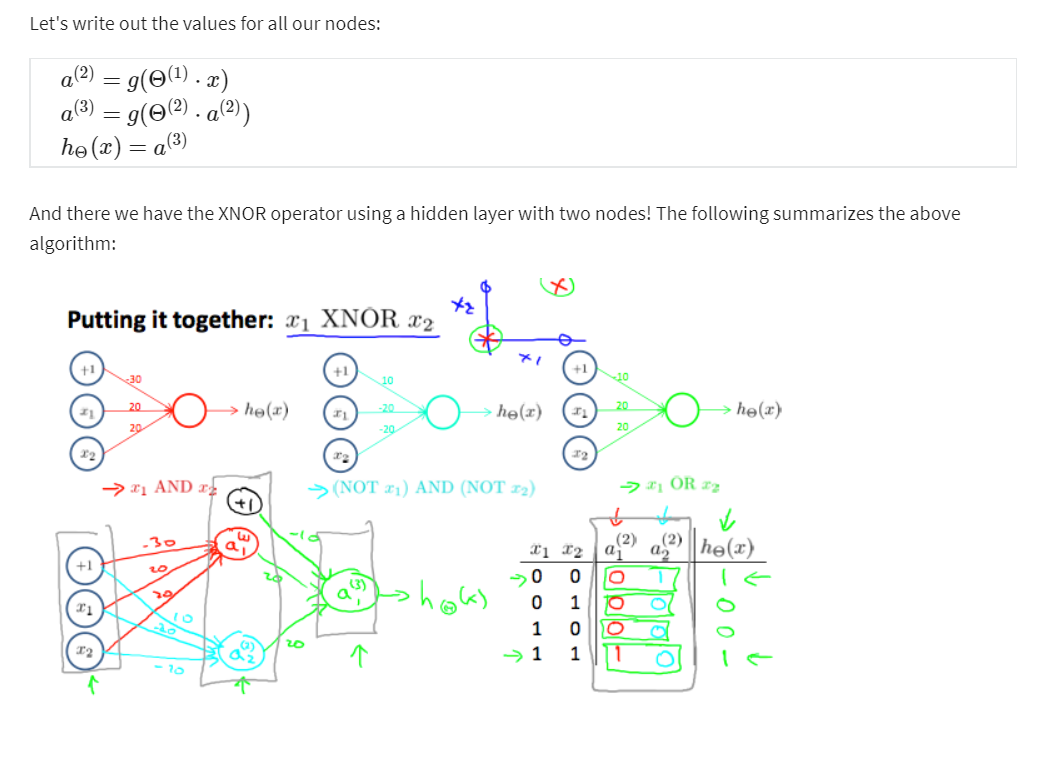












Khom