**01 and 02: Introduction, Regression Analysis, and Gradient Descent**

[Next](http://www.holehouse.org/mlclass/03_Linear_algebra_review.html) [Index](http://www.holehouse.org/mlclass/index.html)

**Introduction to the course**

* We will learn about
  + State of the art
  + How to do the implementation
* Applications of machine learning include
  + Search
  + Photo tagging
  + Spam filters
* The AI dream of building machines as intelligent as humans
  + Many people believe best way to do that is mimic how humans learn
* What the course covers
  + Learn about state of the art algorithms
  + But the algorithms and math alone are no good
  + Need to know how to get these to work in problems
* Why is ML so prevalent?
  + Grew out of AI
  + Build intelligent machines
    - You can program a machine how to do some simple thing
      * For the most part hard-wiring AI is too difficult
    - Best way to do it is to have some way for machines to learn things themselves
      * A mechanism for learning - if a machine can learn from input then it does the hard work for you

***Examples***

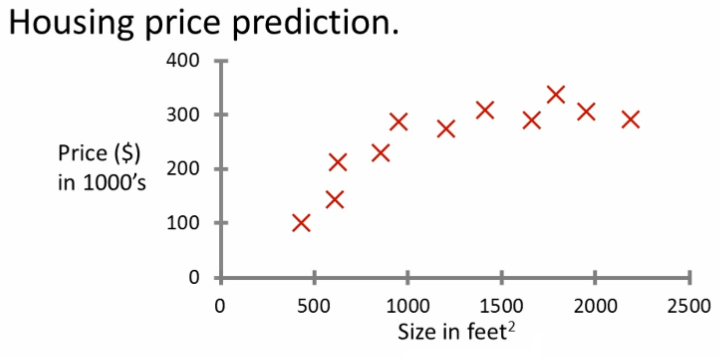
* Database mining
  + Machine learning has recently become so big party because of the huge amount of data being generated
  + Large datasets from growth of automation web
  + Sources of data include
    - Web data (click-stream or click through data)
      * Mine to understand users better
      * Huge segment of silicon valley
    - Medical records
      * Electronic records -> turn records in knowledges
    - Biological data
      * Gene sequences, ML algorithms give a better understanding of human genome
    - Engineering info
      * Data from sensors, log reports, photos etc
* Applications that we cannot program by hand
  + Autonomous helicopter
  + Handwriting recognition
    - This is very inexpensive because when you write an envelope, algorithms can automatically route envelopes through the post
  + Natural language processing (NLP)
    - AI pertaining to language
  + Computer vision
    - AI pertaining vision
* Self customizing programs
  + Netflix
  + Amazon
  + iTunes genius
  + Take users info
    - Learn based on your behavior
* Understand human learning and the brain
  + If we can build systems that mimic (or try to mimic) how the brain works, this may push our own understanding of the associated neurobiology

**What is machine learning?**

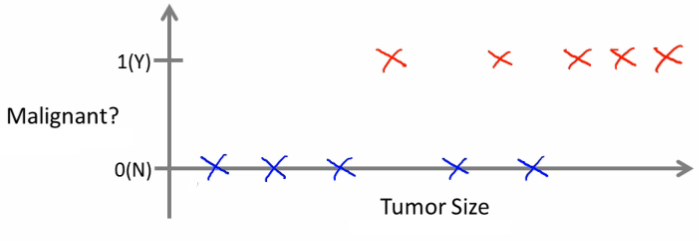
* Here we...
  + Define what it is
  + When to use it
* Not a well defined definition
  + Couple of examples of how people have tried to define it
* Arthur Samuel (1959)
  + ***Machine learning:* "Field of study that gives computers the ability to learn without being explicitly programmed"**
    - Samuels wrote a checkers playing program
      * Had the program play 10000 games against itself
      * Work out which board positions were good and bad depending on wins/losses
* Tom Michel (1999)
  + ***Well posed learning problem:****"***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."**
    - The checkers example,
      * E = 10000s games
      * T is playing checkers
      * P if you win or not
* Several types of learning algorithms
  + **Supervised learning**
    - Teach the computer how to do something, then let it use it;s new found knowledge to do it
  + **Unsupervised learning**
    - Let the computer learn how to do something, and use this to determine structure and patterns in data
  + Reinforcement learning
  + Recommender systems
* This course
  + Look at practical advice for applying learning algorithms
  + Learning a set of tools and **how** to apply them

**Supervised learning - introduction**

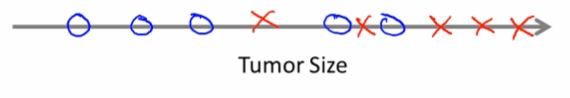
* Probably the most common problem type in machine learning
* Starting with an example
  + How do we predict housing prices
    - Collect data regarding housing prices and how they relate to size in feet



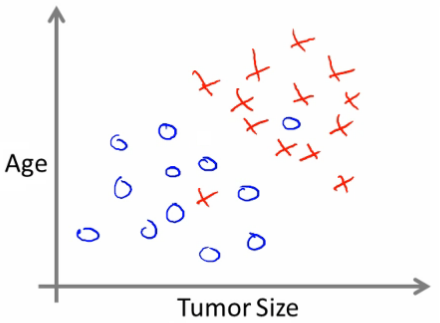
* **Example problem:** "Given this data, a friend has a house 750 square feet - how much can they be expected to get?"
* What approaches can we use to solve this?
  + Straight line through data
    - Maybe $150 000
  + Second order polynomial
    - Maybe $200 000
  + One thing we discuss later - how to chose straight or curved line?
  + Each of these approaches represent a way of doing supervised learning
* *What does this mean?*
  + We gave the algorithm a data set where a "right answer" was provided
  + So we know actual prices for houses
    - The idea is we can learn what makes the price a certain value from the **training data**
    - The algorithm should then produce more right answers based on new training data where we don't know the price already
      * i.e. predict the price
* We also call this a **regression problem**
  + Predict continuous valued output (price)
  + No real discrete delineation
* Another example
  + Can we definer breast cancer as malignant or benign based on tumour size



* Looking at data
  + Five of each
  + Can you estimate prognosis based on tumor size?
  + This is an example of a **classification problem**
    - Classify data into one of two discrete classes - no in between, either malignant or not
    - In classification problems, can have a discrete number of possible values for the output
      * e.g. maybe have four values
        + 0 - benign
        + 1 - type 1
        + 2 - type 2
        + 3 - type 4
* In classification problems we can plot data in a different way



* Use only one attribute (size)
  + In other problems may have multiple attributes
  + We may also, for example, know age and tumor size

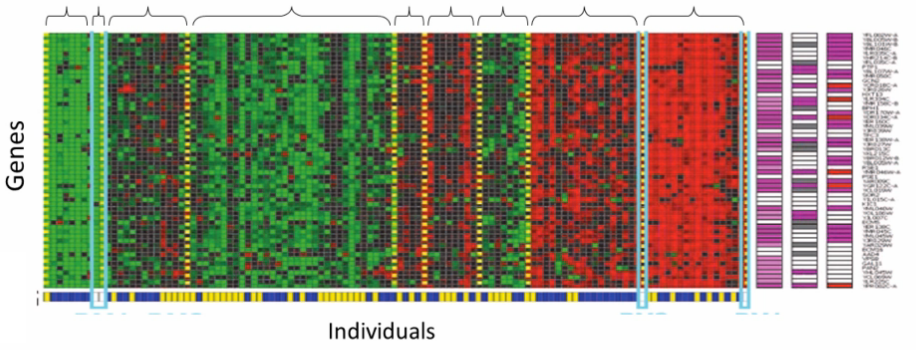
****

* Based on that data, you can try and define separate classes by
  + Drawing a straight line between the two groups
  + Using a more complex function to define the two groups (which we'll discuss later)
  + Then, when you have an individual with a specific tumor size and who is a specific age, you can hopefully use that information to place them into one of your classes
* You might have many features to consider
  + Clump thickness
  + Uniformity of cell size
  + Uniformity of cell shape
* The most exciting algorithms can deal with an infinite number of features
  + How do you deal with an infinite number of features?
  + Neat mathematical trick in support vector machine (which we discuss later)
    - If you have an infinitely long list - we can develop and algorithm to deal with that
* ***Summary***
  + Supervised learning lets you get the "right" data a
  + Regression problem
  + Classification problem

**Unsupervised learning - introduction**

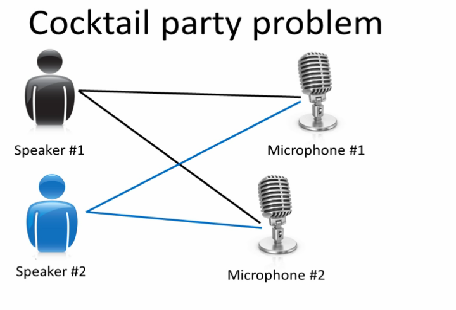
* Second major problem type
* In unsupervised learning, we get unlabeled data
  + Just told - here is a data set, can you structure it
* One way of doing this would be to cluster data into to groups
  + This is a **clustering algorithm**

**Clustering algorithm**

* Example of clustering algorithm
  + Google news
    - Groups news stories into cohesive groups
  + Used in any other problems as well
    - Genomics
    - Microarray data
      * Have a group of individuals
      * On each measure expression of a gene
      * Run algorithm to cluster individuals into types of people  
        
    - Organize computer clusters
      * Identify potential weak spots or distribute workload effectively
    - Social network analysis
      * Customer data
    - Astronomical data analysis
      * Algorithms give amazing results
* Basically
  + Can you automatically generate structure
  + Because we don't give it the answer, it's unsupervised learning

**Cocktail party algorithm**

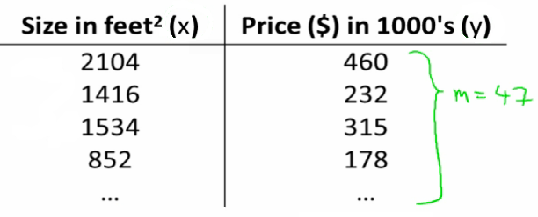
* Cocktail party problem
  + Lots of overlapping voices - hard to hear what everyone is saying
    - Two people talking
    - Microphones at different distances from speakers



* Record sightly different versions of the conversation depending on where your microphone is
  + But overlapping none the less
* Have recordings of the conversation from each microphone
  + Give them to a cocktail party algorithm
  + Algorithm processes audio recordings
    - Determines there are two audio sources
    - Separates out the two sources
* Is this a very complicated problem
  + Algorithm can be done with one line of code!
  + **[W,s,v] = svd((repmat(sum(x.\*x,1), size(x,1),1).\*x)\*x');**
    - Not easy to identify
    - But, programs can be short!
    - Using octave (or MATLAB) for examples
      * Often prototype algorithms in octave/MATLAB to test as it's very fast
      * Only when you show it works migrate it to C++
      * Gives a much faster agile development
* Understanding this algorithm
  + **svd** - linear algebra routine which is built into octave
    - In C++ this would be very complicated!
  + Shown that using MATLAB to prototype is a really good way to do this

**Linear Regression**

* Housing price data example used earlier
  + Supervised learning regression problem
* What do we start with?
  + Training set (this is your data set)
  + Notation (*used throughout the course*)
    - m = number of **training examples**
    - x's = input variables / features
    - y's = output variable "target" variables
      * (x,y) - single training example
      * (xi, yj) - specific example (ith training example)
        + i is an index to training set



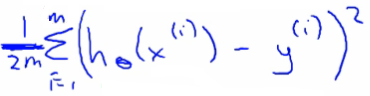
* With our training set defined - how do we used it?
  + Take training set
  + Pass into a learning algorithm
  + Algorithm outputs a function (denoted *h*) (h = **hypothesis**)
    - This function takes an input (e.g. size of new house)
    - Tries to output the estimated value of Y
* How do we represent hypothesis *h*?
  + Going to present h as;
    - hθ(x) = θ0 + θ1x
      * h(x) (shorthand)

http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr_files/Image%20%5b7%5d.png

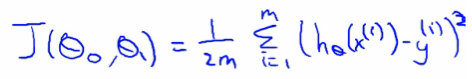
* What does this mean?
  + Means Y is a linear function of x!
  + θi are **parameters**
    - θ0 is zero condition
    - θ1 is gradient
* This kind of function is a linear regression with one variable
  + Also called **univariate linear regression**
* So in summary
  + A hypothesis takes in some variable
  + Uses parameters determined by a learning system
  + Outputs a prediction based on that input

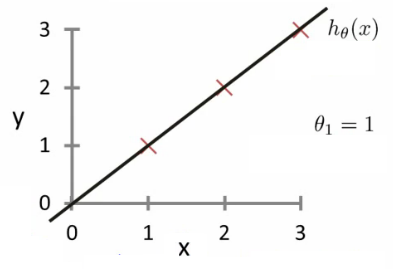
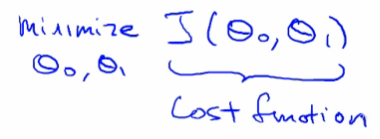
**Linear regression - implementation (cost function)**

* A cost function lets us figure out how to fit the best straight line to our data
* Choosing values for θi (parameters)
  + Different values give you different functions
  + If θ0 is 1.5 and θ1 is 0 then we get straight line parallel with X along 1.5 @ y
  + If θ1 is > 0 then we get a positive slope
* Based on our training set we want to generate parameters which make the straight line
  + Chosen these parameters so hθ(x) is close to y for our training examples
    - Basically, uses xs in training set with hθ(x) to give output which is as close to the actual y value as possible
    - Think of hθ(x) as a "y imitator" - it tries to convert the x into y, and considering we already have y we can evaluate how well hθ(x) does this
* To formalize this;
  + We want to want to solve a **minimization problem**
  + Minimize (hθ(x) - y)2
    - i.e. minimize the difference between h(x) and y for each/any/every example
  + Sum this over the training set



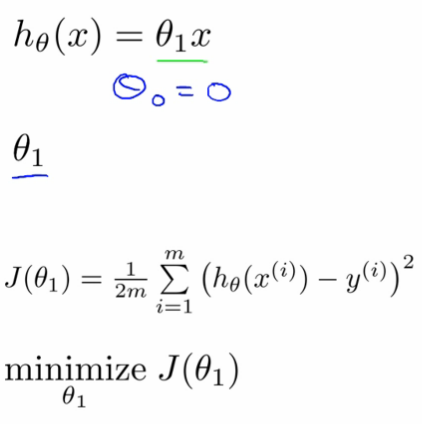
* Minimize squared different between predicted house price and actual house price
  + 1/2m
    - 1/m - means we determine the average
    - 1/2m the 2 makes the math a bit easier, and doesn't change the constants we determine at all (i.e. half the smallest value is still the smallest value!)
  + Minimizing θ0/θ1 means we get the values of θ0 and θ1 which find on average the minimal deviation of x from y when we use those parameters in our hypothesis function
* More cleanly, this is a cost function

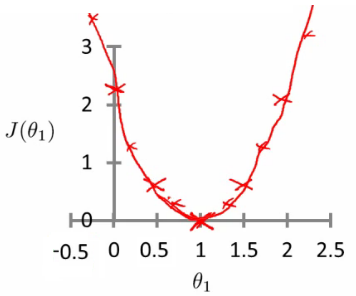


* And we want to minimize this cost function
  + Our cost function is (because of the summartion term) inherently looking at ALL the data in the training set at any time
* **So to recap**
  + **Hypothesis** - is like your prediction machine, throw in an *x* value, get a putative *y* value  
    
  + **Cost** - is a way to, using your training data, determine values for your θ values which make the hypothesis as accurate as possible  
    
    - This cost function is also called the squared error cost function
      * This cost function is reasonable choice for most regression functions
      * Probably most commonly used function
  + In case J(θ0,θ1) is a bit abstract, going into what it does, why it works and how we use it in the coming sections

**Cost function - a deeper look**

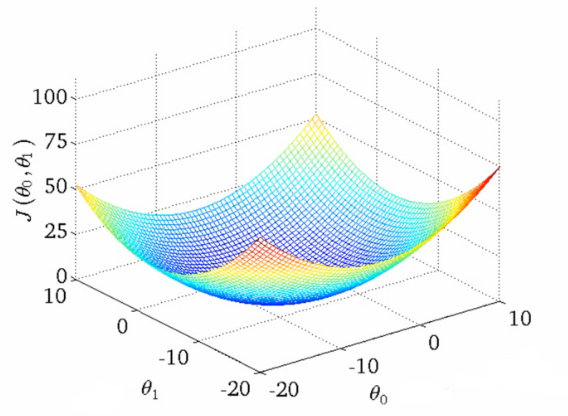
* Lets consider some intuition about the cost function and why we want to use it
  + The cost function determines parameters
  + The value associated with the parameters determines how your hypothesis behaves, with different values generate different
* Simplified hypothesis
  + Assumes θ0 = 0



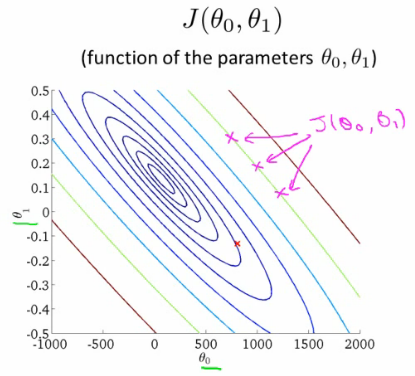
* Cost function and goal here are very similar to when we have θ0, but with a simpler parameter
  + Simplified hypothesis makes visualizing cost function J() a bit easier
* So hypothesis pass through 0,0
* Two key functins we want to understand
  + hθ(x)
    - Hypothesis is a function of x - function of what the size of the house is
  + J(θ1)
    - Is a function of the parameter of θ1
  + So for example
    - θ1 = 1
    - J(θ1) = 0
  + Plot
    - θ1 vs J(θ1)
    - Data
      * 1)
        + θ1 = 1
        + J(θ1) = 0
      * 2)
        + θ1 = 0.5
        + J(θ1) = ~0.58
      * 3)
        + θ1 = 0
        + J(θ1) = ~2.3
  + If we compute a range of values plot
    - J(θ1) vs θ1 we get a polynomial (looks like a quadratic)  
      
* The optimization objective for the learning algorithm is find the value of θ1 which minimizes J(θ1)
  + So, here θ1 = 1 is the best value for θ1

**A deeper insight into the cost function - simplified cost function**

* Assume you're familiar with contour plots or contour figures
  + Using same cost function, hypothesis and goal as previously
  + It's OK to skip parts of this section if you don't understand cotour plots
* Using our original complex hyothesis with two pariables,
  + So cost function is
    - J(θ0, θ1)
* Example,
  + Say
    - θ0 = 50
    - θ1 = 0.06
  + Previously we plotted our cost function by plotting
    - θ1 vs J(θ1)
  + Now we have two parameters
    - Plot becomes a bit more complicated
    - Generates a 3D surface plot where axis are
      * X = θ1
      * Z = θ0
      * Y = J(θ0,θ1)



* We can see that the height (y) indicates the value of the cost function, so find where y is at a minimum
* Instead of a surface plot we can use a **contour figures/plots**
  + Set of ellipses in different colors
  + Each colour is the same value of J(θ0, θ1), but obviously plot to different locations because θ1 and θ0 will vary
  + Imagine a bowl shape function coming out of the screen so the middle is the concentric circles

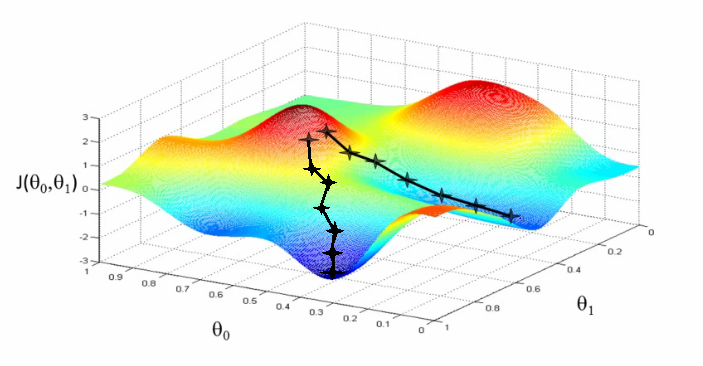


* Each point (like the red one above) represents a pair of parameter values for Ɵ0 and Ɵ1
  + Our example here put the values at
    - θ0 = ~800
    - θ1 = ~-0.15
  + Not a good fit
    - i.e. these parameters give a value on our contour plot far from the center
  + If we have
    - θ0 = ~360
    - θ1 = 0
    - This gives a better hypothesis, but still not great - not in the center of the countour plot
  + Finally we find the minimum, which gives the best hypothesis
* Doing this by eye/hand is a pain in the ass
  + What we really want is an efficient algorithm fro finding the minimum for θ0 and θ1

**Gradient descent algorithm**

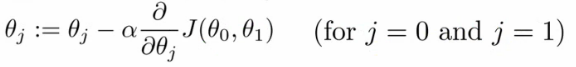
* Minimize cost function J
* Gradient descent
  + Used all over machine learning for minimization
* Start by looking at a general J() function
* Problem
  + We have J(θ0, θ1)
  + We want to get **min J(θ0, θ1)**
* Gradient descent applies to more general functions
  + J(θ0, θ1, θ2 .... θn)
  + min J(θ0, θ1, θ2 .... θn)

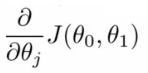
**How does it work?**

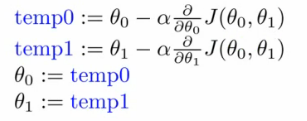
* Start with initial guesses
  + Start at 0,0 (or any other value)
  + Keeping changing θ0 and θ1 a little bit to try and reduce J(θ0,θ1)
* Each time you change the parameters, you select the gradient which reduces J(θ0,θ1) the most possible
* Repeat
* Do so until you converge to a local minimum
* Has an interesting property
  + Where you start can determine which minimum you end up  
    
  + Here we can see one initialization point led to one local minimum
  + The other led to a different one

**A more formal definition**

* Do the following until covergence

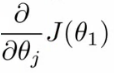


* What does this all mean?
  + Update θj by setting it to (θj - α) times the partial derivative of the cost function with respect to θj
* Notation
  + :=
    - Denotes assignment
    - NB a = b is a *truth assertion*
  + α (alpha)
    - Is a number called the **learning rate**
    - Controls how big a step you take
      * If α is big have an aggressive gradient descent
      * If α is small take tiny steps
* Derivative term  
  
  + Not going to talk about it now, derive it later
* There is a subtly about how this gradient descent algorithm is implemented
  + Do this for θ0 and θ1
  + For j = 0 and j = 1 means we **simultaneously**update both
  + How do we do this?
    - Compute the right hand side for both θ0and θ1
      * So we need a temp value
    - Then, update θ0and θ1 at the same time
    - We show this graphically below



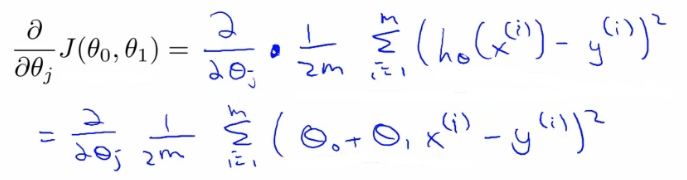
* If you implement the non-simultaneous update it's not gradient descent, and will behave weirdly
  + But it might look sort of right - so it's important to remember this!

**Understanding the algorithm**

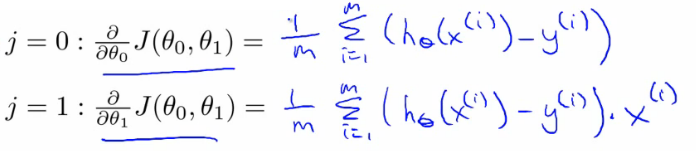
* To understand gradient descent, we'll return to a simpler function where we minimize one parameter to help explain the algorithm in more detail
  + min θ1 J(θ1) where θ1 is a real number
* Two key terms in the algorithm
  + Alpha
  + Derivative term
* Notation nuances
  + Partial derivative vs. derivative
    - Use partial derivative when we have multiple variables but only derive with respect to one
    - Use derivative when we are deriving with respect to all the variables
* Derivative term  
          
  + Derivative says
    - Lets take the tangent at the point and look at the slope of the line
    - So moving towards the mimum (down) will greate a negative derivative, alpha is always positive, so will update j(θ1) to a smaller value
    - Similarly, if we're moving up a slope we make j(θ1) a bigger numbers
* Alpha term (α)
  + What happens if alpha is too small or too large
  + Too small
    - Take baby steps
    - Takes too long
  + Too large
    - Can overshoot the minimum and fail to converge
* When you get to a local minimum
  + Gradient of tangent/derivative is 0
  + So derivative term = 0
  + alpha \* 0 = 0
  + So θ1 = θ1- 0
  + So θ1 remains the same
* As you approach the global minimum the derivative term gets smaller, so your update gets smaller, even with alpha is fixed
  + Means as the algorithm runs you take smaller steps as you approach the minimum
  + So no need to change alpha over time

**Linear regression with gradient descent**

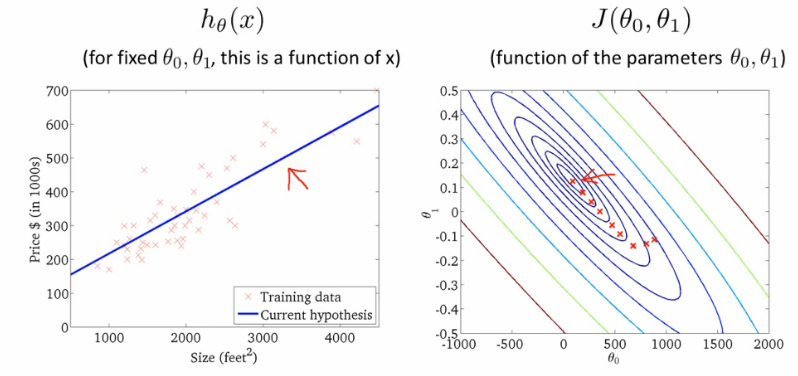
* Apply gradient descent to minimize the squared error cost function J(θ0, θ1)
* Now we have a partial derivative



* So here we're just expanding out the first expression
  + J(θ0, θ1) = 1/2m....
  + hθ(x) = θ0 + θ1\*x
* So we need to determine the derivative for each parameter - i.e.
  + When j = 0
  + When j = 1
* Figure out what this partial derivative is for the θ0 and θ1 case
  + When we derive this expression in terms of j = 0 and j = 1 we get the following



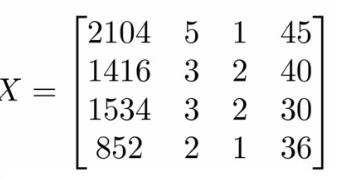
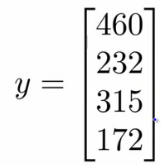
* To check this you need to know multivariate calculus
  + So we can plug these values back into the gradient descent algorithm
* How does it work
  + Risk of meeting different local optimum
  + The linear regression cost function is always a **convex function** - always has a single minimum
    - Bowl shaped
    - One global optima
      * So gradient descent will always converge to global optima
  + In action
    - Initialize values to
      * θ0 = 900
      * θ1 = -0.1



* End up at a global minimum
* This is actually **Batch Gradient Descent**
  + Refers to the fact that over each step you look at all the training data
    - Each step compute over m training examples
  + Sometimes non-batch versions exist, which look at small data subsets
    - We'll look at other forms of gradient descent (to use when m is too large) later in the course
* There exists a numerical solution for finding a solution for a minimum function
  + **Normal equations** method
  + Gradient descent scales better to large data sets though
  + Used in lots of contexts and machine learning

**What's next - important extensions**  
*Two extension to the algorithm*

* **1) Normal equation for numeric solution**
  + To solve the minimization problem we can solve it [ min J(θ0, θ1) ] exactly using a numeric method which avoids the iterative approach used by gradient descent
  + Normal equations method
  + Has advantages and disadvantages
    - Advantage
      * No longer an alpha term
      * Can be much faster for some problems
    - Disadvantage
      * Much more complicated
  + We discuss the normal equation in the **linear regression with multiple features** section
* **2) We can learn with a larger number of features**
  + So may have other parameters which contribute towards a prize
    - e.g. with houses
      * Size
      * Age
      * Number bedrooms
      * Number floors
    - x1, x2, x3, x4
  + With multiple features becomes hard to plot
    - Can't really plot in more than 3 dimensions
    - Notation becomes more complicated too
      * Best way to get around with this is the notation of linear algebra
      * Gives notation and set of things you can do with matrices and vectors
      * e.g. Matrix

* We see here this matrix shows us
  + Size
  + Number of bedrooms
  + Number floors
  + Age of home
* All in one variable
  + Block of numbers, take all data organized into one big block
* Vector
  + Shown as *y*
  + Shows us the prices
* Need linear algebra for more complex linear regression modles
* Linear algebra is good for making computationally efficient models (as seen later too)
  + Provide a good way to work with large sets of data sets
  + Typically vectorization of a problem is a common optimization technique