**Machine Learning – Lecture Notes – Part I**

**Week 1**

**Introduction**

Welcome

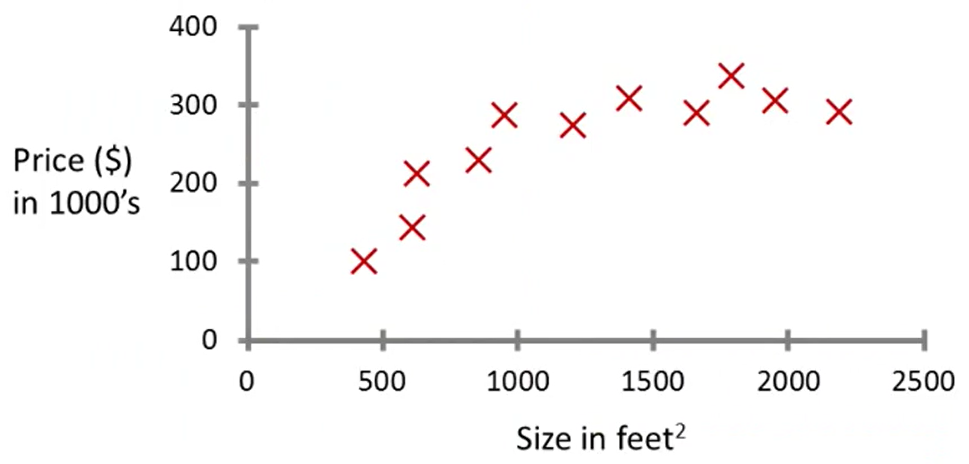
* Machine learning is all about building intelligent machines.
  + Grew out of the realm of AI
  + Gives new capabilities for computers
* Here are some examples of uses of machine learning:
  + Database mining
    - Large datasets from growth of automation/web
    - E.g. web click data (click-stream data), medical records, biology, engineering
  + Applications that can’t be programmed by hand
    - E.g. autonomous helicopter, handwriting recognition, most of Natural Language Processing (NLP), Computer Vision
  + Self-customizing programs
    - Amazon, Netflix product recommendations
  + Understanding human learning (brain, real AI)

What is Machine Learning?

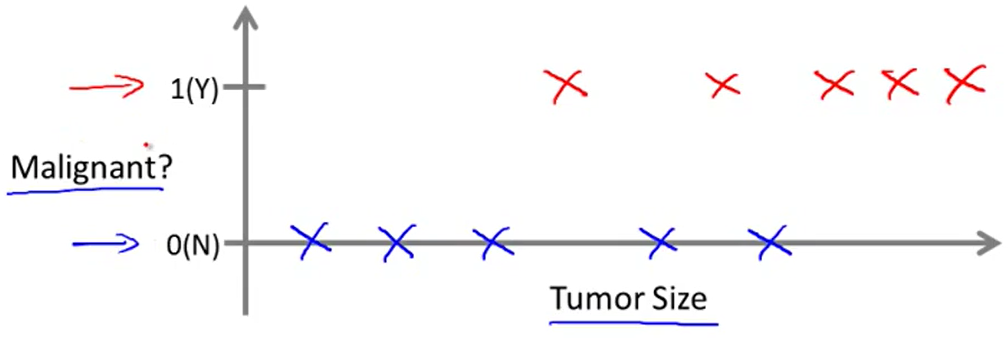
* Machine Learning definitions:
  + Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
  + Tom Mitchell (1998). Well-posed Learning Problem: A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with E.
* Machine learning algorithms generally fall under two categories.
  + Supervised learning
  + Unsupervised learning
* Other algorithms include: reinforcement learning, recommender systems.

Supervised Learning

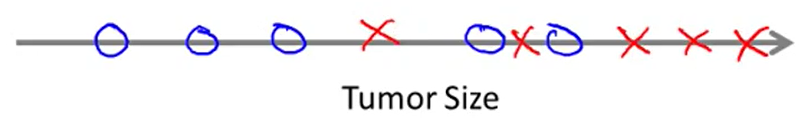
* Let’s start with an example. Let’s say you want to model housing prices in a city. You collect a dataset as shown below.



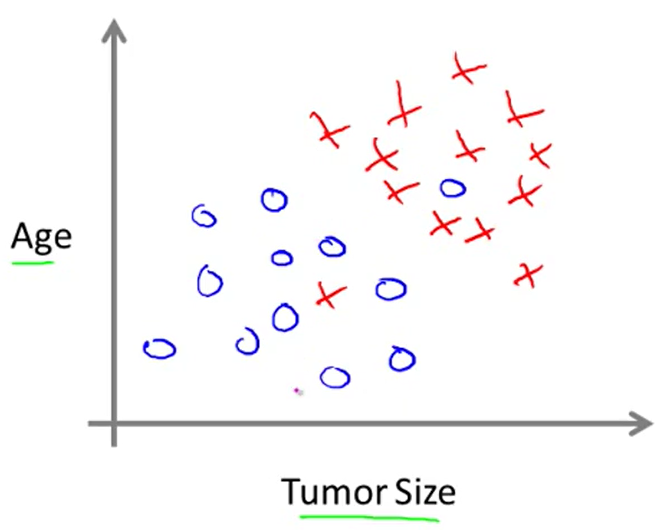
* A simple algorithm might just put a line of best fit through the data set. A better algorithm might use a quadratic to model the data.
* But this is an example of a supervised learning algorithm.
  + “Right answers” are given
* This is also called a **regression** problem. Regression is used to predict a **continuous** valued output (price).
* Another example is finding weather a breast tumour is malignant or benign, based on size. Let’s say we get the graph below.



* Let’s say a friend has tumour of a certain size. Is it possible to find the probability that the person has a malignant tumour?
* This is known as a **classification** problem. Classification is used to predict a **discrete** valued output (0 or 1).
* In classification problems, there is also a slightly different way to plot this data. This data is all plotted on a 1-dimensional scale.



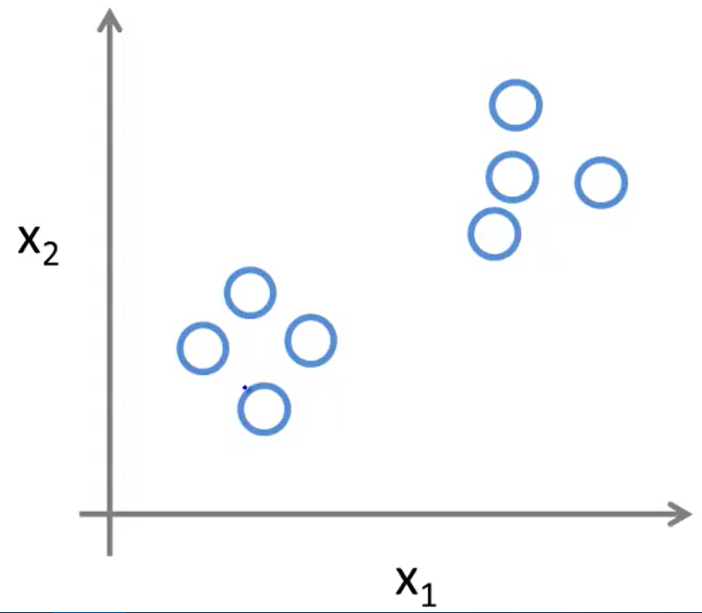
* This is important because when you have multiple attributes, you get higher dimensional data, which is harder to plot.
* Let’s say we know each patient’s age and tumour size (and whether it was malignant/benign). The graph may look like the following.



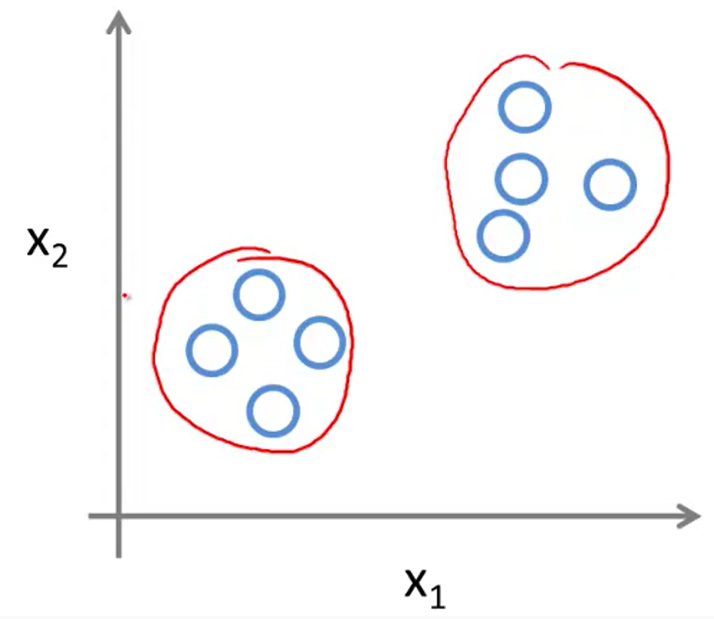
* Again, if you have a friend has a tumour, you could use a model to find the probability that she has a malignant tumour.
* You could fit a straight line through the data, for instance.

Unsupervised Learning

* In an unsupervised learning algorithm, a computer is given a data set, but it is not told what data points are correct and what data points are incorrect. Instead, all the data points seem uniform.
* A computer might be given the data set below.



* The algorithm analyzing this data could decide that it lives in two clusters.

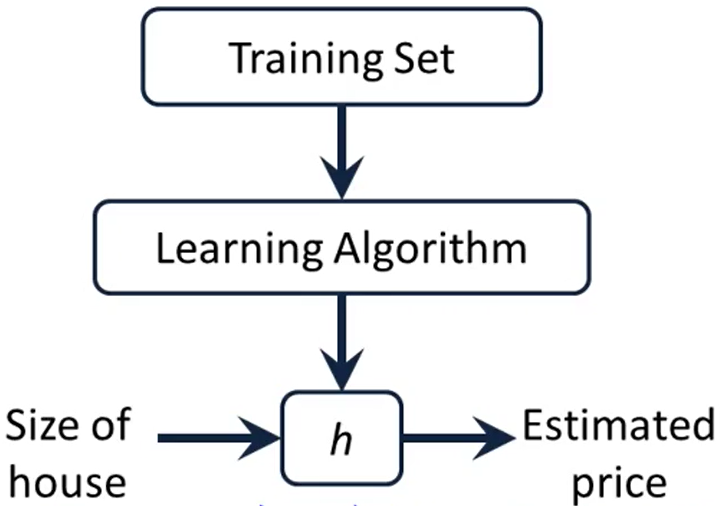


* This is called a **clustering algorithm**. This algorithm is used by Google News. When you look at an article on Google News, it suggests links that are related to that story.
* Clustering algorithms are also used in understanding genomics. For example, take a collection of 1,000,000 genes, and find 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.
* It is also used in organizing computer clusters, analyzing social networks, market segmentation, and astronomical data analysis.
* Non-clustering algorithms are used to find structure in a chaotic environment (e.g. identifying individual sound tracks or voices from a mesh of sounds at a cocktail party).

**Model and Cost Function**

Model Representation

* Let’s refer to the last example of selling houses in Portland, OR. The data that you feed into the algorithm is called the **data set** or **training set**.
* Notation of the training sets will be used as follows.
  + **m** = number of training examples
  + **x**’s = “input” variables / features
  + **y**’s = “output” variables / “target” features
  + **(x, y)** = one training example
  + **(x(i), y(i))** = the ith training example
  + **X** = the space of input values **x**
  + **Y** = the space of output values **y**
* To describe the supervised learning problem more formally, our goal is, given a training set, to learn a function so that is a “good” predictor for the corresponding value of *y*.
* The way this training set is used is that the set is fed into the learning algorithm, which improves the hypothesis function *h*. The function *h* maps some set of *x*’s to some set of *y*’s.



* In this case the *x* is the size of house (ft2) and the *y* is the estimated price ($).
* The next critical question is, how to we pick a hypothesis *h*? For now, we will say that our is a linear function.
* This model is called linear regression with one variable or **univariate linear regression**.

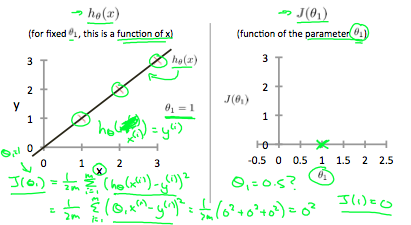
Cost Function

* In a linear regression, we have both a training set and a hypothesis function.
  + Hypothesis:
  + : Parameters
* In this video, we learn how to choose the parameters and .
* **Idea:** Choose and so that is close to for our training examples .
* We can do this by minimizing over and . Thus, we must minimize the function below, which is the sum of squares of differences between the predicted and actual values in a training set.

* This is also known as the squared error function. The mean is halved as a convenience for computing the gradient descent, as the derivative term of the function will cancel the .

Cost Function – Intuition I

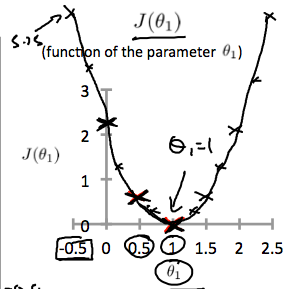
* To better visualize the function, we will use a simplified cost function . The function . Then we get the cost function below.
* Now the goal is to minimize over .
* Let’s better understand both functions.
  + for a fixed , this is a function of
  + function of a parameter
* To minimize this function, you can pick different values of and see what values are returned from . Then, you get a cost function. You pick the minimum value of to get the best linear model for the data.



* This is one example where the training set consists of the points . If we use a , we get a line that passes through all points and .



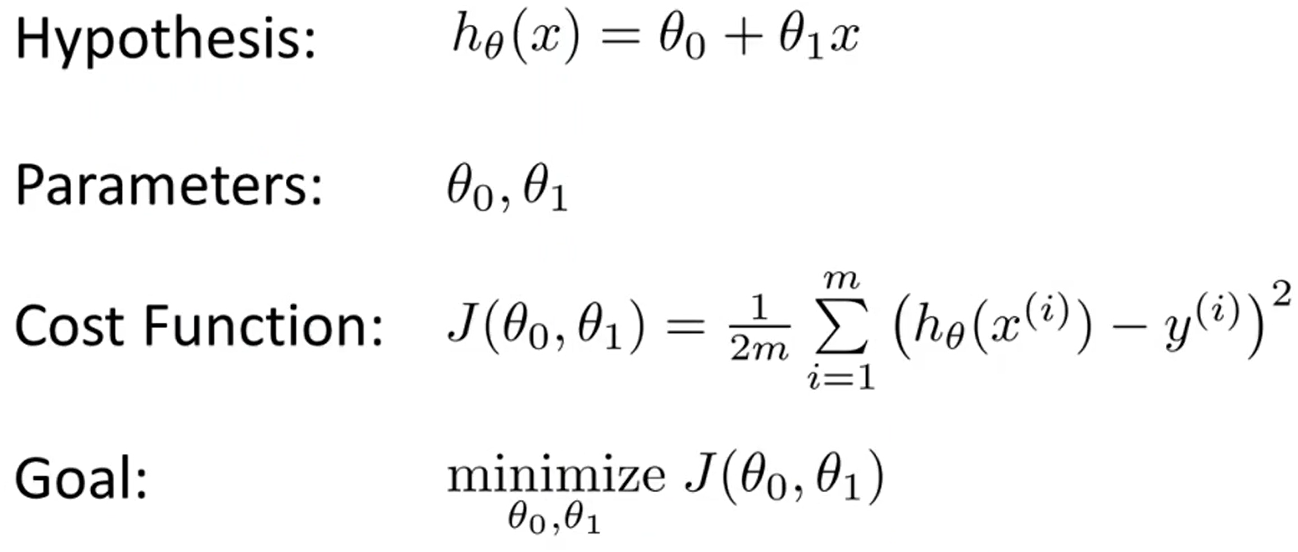
* If we use a , then , which is a worse model than the previous one. Eventually, we get the following graph.



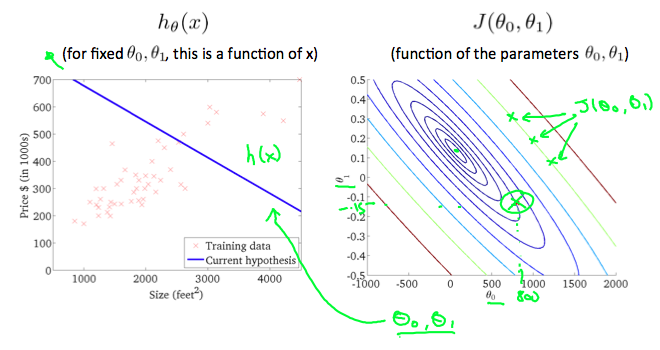
* This shows that the global minimum occurs at , which is the optimal value of the hypothesis function .

Cost Function – Intuition II

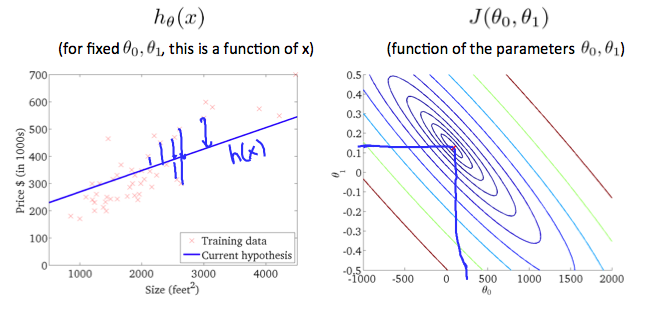
* Now we will look at the entire cost function, including both parameters and .



* A contour plot is a graph that contains many contour lines. A contour line of a two variable function has constant value for all points on an individual line.
* When we plot , we get a contour plot on a 2D graph since we cannot plot all 3 dimensions. The function looks like below for a specific example.



* This graph is quite far from the center of the ellipses in the contour graph. Thus, it means that the value of is quite high.

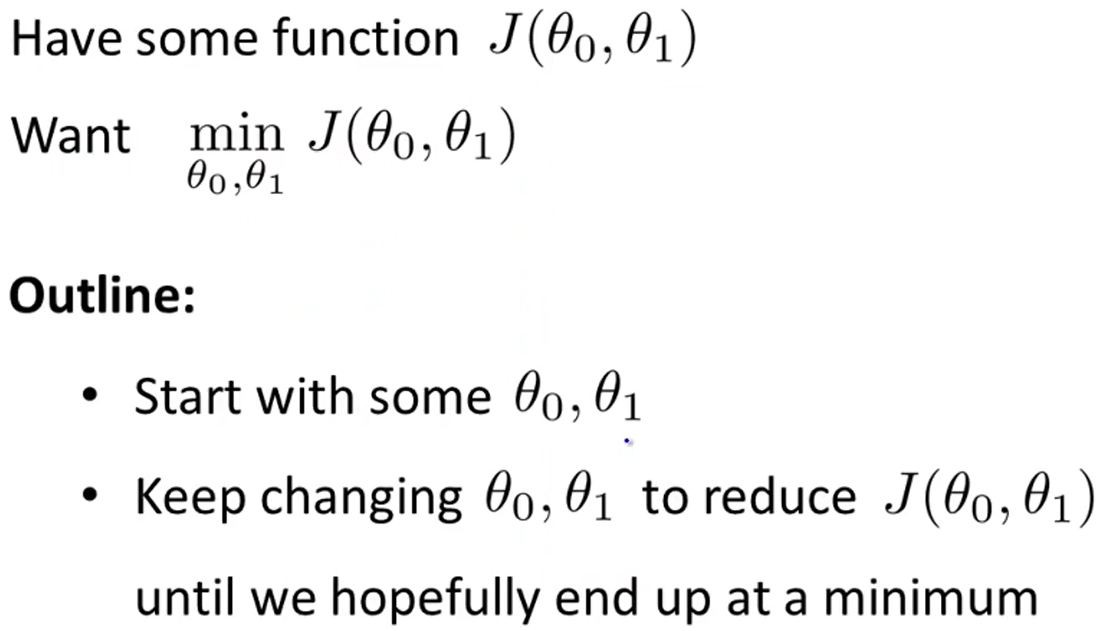


* When a new line of best fit is taken, it shows that the value of is very low because it lies near the center of the smallest circle on the contour graph. Thus, it is the line of best fit for the data.

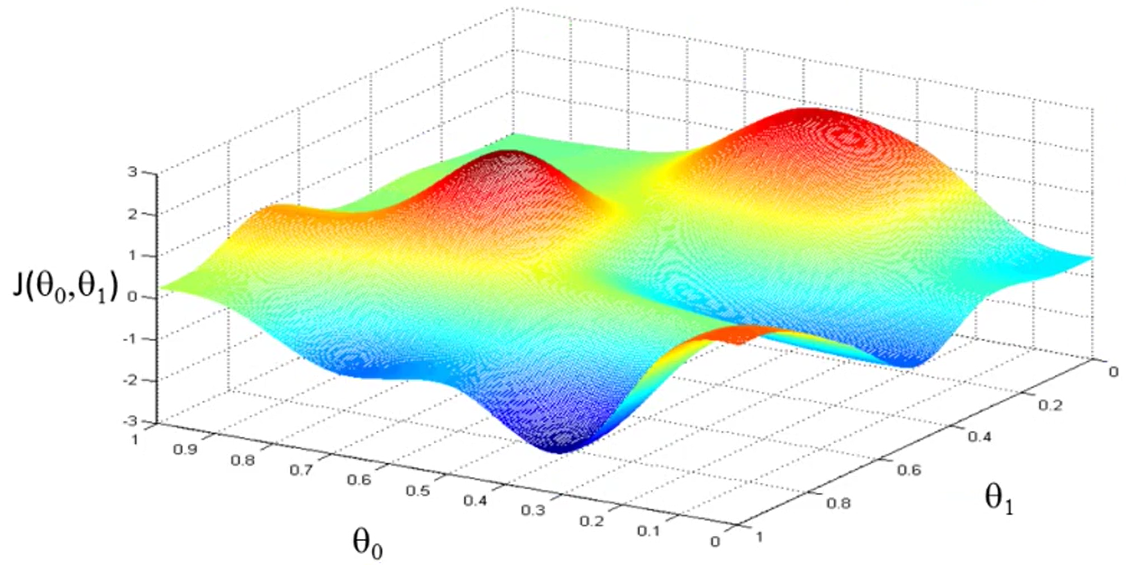
**Parameter Learning**

Gradient Descent

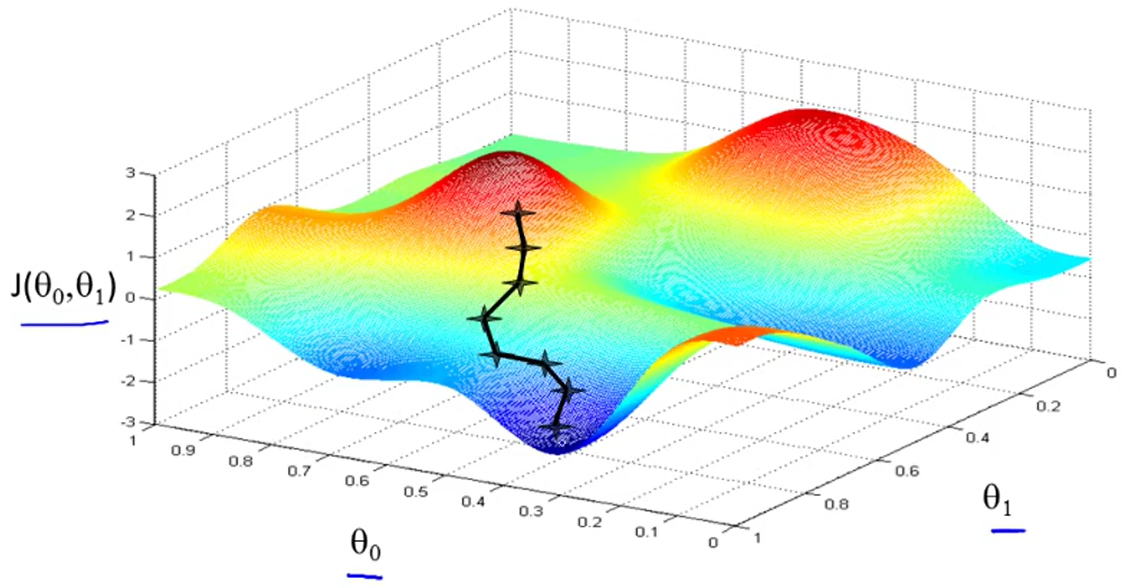
* Here is the outline of the problem we hope to solve with a method called **gradient descent**.



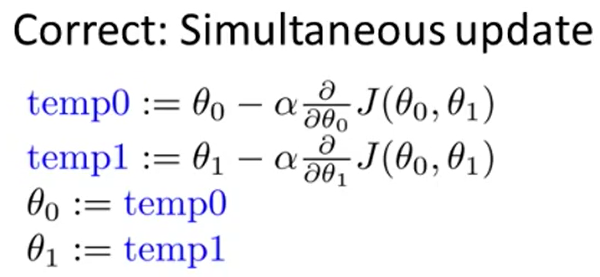
* Commonly, we initialize the variables to .
* The intuition behind the gradient descent algorithm is as follows. Let’s say you’re standing on a hilly terrain as shown in the graph below.



* The gradient descent algorithm will start a specific point. It will then look 360° and find the direction that will take it downhill most quickly. It might form a path like the one below.

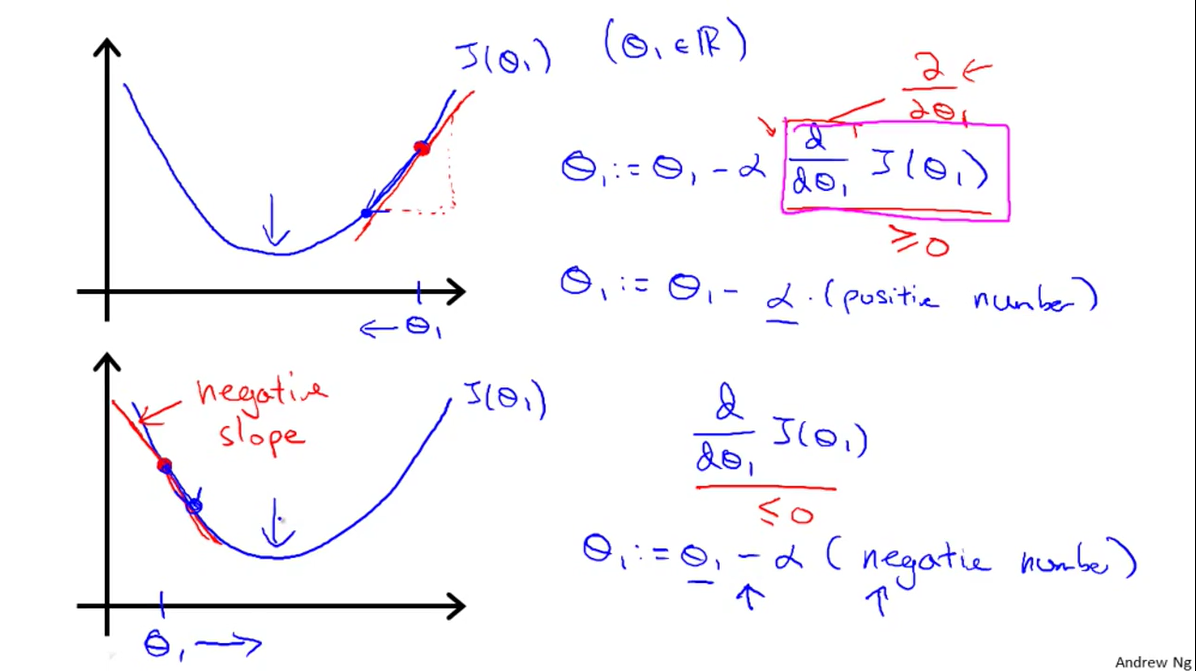


* However, this algorithm doesn’t always find the **global minimum** – it finds the **local minimum** that is easiest to reach.
* Now we’ll look at the gradient descent algorithm.
* The symbol is the assignment operator. It means override the value of the left side with the value of the right side (in computer science).
* The represents the magnitude of the step that the algorithm takes in any direction. It is also called the **learning rate**.
* The last term is the partial derivative of the cost function with respect to .
* We can do this simultaneously updating both variables using the following method.



Gradient Descent – Intuition

* We saw the method of gradient descent last video. Here, we will show what the learning rate () and derivative term () mean.
* We will consider problems of the form , where .
* Then let’s see a function that needs to be minimized. The following shows such a graph. As shown, if , then the is updated to the left, and vice versa on the other side of the graph.



* If is too small, the gradient descent algorithm might work very slowly.
* If is too large, the gradient descent algorithm can overshoot the minimum. It may fail to converge and thus diverge – yielding no solution.
* If the gradient descent algorithm finds a local minimum, then it updates as follows.
* Gradient descent can converge to a local minimum, even with a fixed learning rate .
* As the algorithm approaches the minimum, gradient descent will automatically take smaller steps. Thus, no need to decrease over time.

Gradient Descent for Linear Regression

* To find out how to apply this gradient descent method to linear regression, the key term to calculate is .
* Let’s find the partial derivative of the cost function with respect to each of the variables and .
* But we must find this derivative for both .
* Now here’s the gradient descent algorithm for linear regression.
* Gradient descent has the problem that it finds local minima, not global minima. However, for all regression problems, the cost function is always a **convex function** or a “bowl-shaped function”.
* The function won’t have any local optima except for the global optimum.
* This algorithm is also known as **“batch” gradient descent**.
  + “Batch”: Each step of gradient descent uses all the training examples.

**Linear Algebra Review**

Matrices and Vectors

* **Matrix:** A rectangular array of numbers.
* **Dimension of Matrix:** number of rows **x** number of columns (the one above is a 3x2 matrix or a matrix in ).
* **Matrix Elements** (entries of matrix)

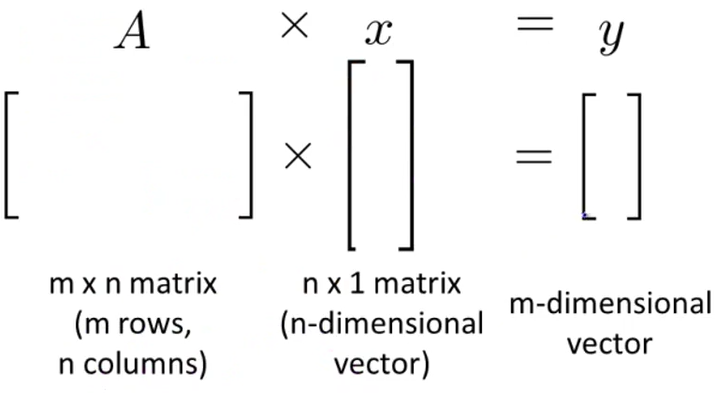
* **Vector:** An matrix.
* **Dimension of Vector:** number of rows (the one above is 4-dimensional vector, or a vector in )
* **Vector Elements** (entries of vector)
* There are two ways to index into a vector, 1-indexed or 0-indexed.
* In general, all our vectors and matrices will be 1-indexed. Note that for some programming languages, the arrays are 0-indexed.

Addition and Scalar Multiplication

* In **matrix addition**, you just add the respective components of the matrices to each other.
* You can only add matrices of the same dimension. In other words, matrix can only be added to another matrix.
* In **scalar multiplication**, you just multiply the scalar by each of the components of the matrix.
* You can also “divide” a matrix by a scalar by multiplying by the reciprocal.

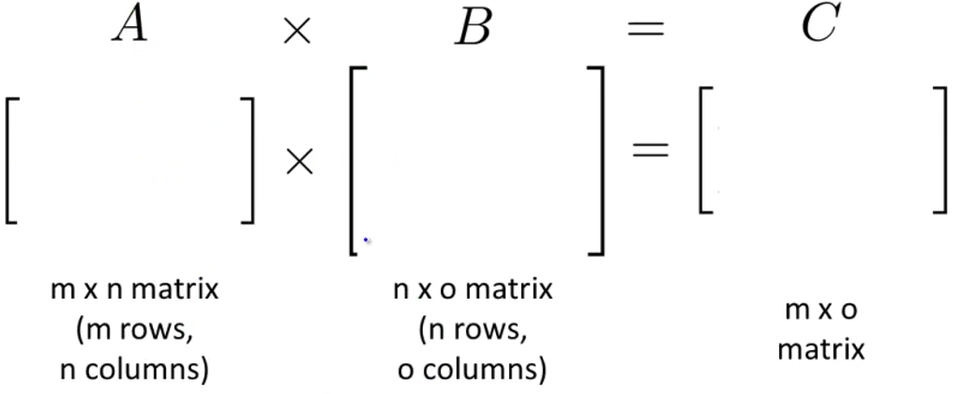
Matrix-Vector Multiplication

* Now we’ll start by talking about multiplying matrices and vectors. Let’s start with an example.
* Since we multiplied a matrix by a matrix, we get a matrix.
* In general, we have the following rule.



Matrix-Matrix Multiplication

* Let’s start with an example of matrix-matrix multiplication.
* Here is the rule about multiplying matrices together.



* The column of matrix is obtained by multiplying with the column of matrix (for ).

Matrix Multiplication Properties

* Let and be matrices. Then, in general, (**not commutative**).
* Let , and be matrices. Then, in general, (**associative**).
* The **identity matrix** is the matrix that, if multiplied with any other (valid) matrix, will output the original matrix.
* It is denoted by or . Examples of these matrices include the following.
* For any matrix , . **For** , if is an matrix, then must be the identity matrix. **For** , if is an matrix, then must be the identity matrix.
* Thus, this really says .

Inverse and Transpose:

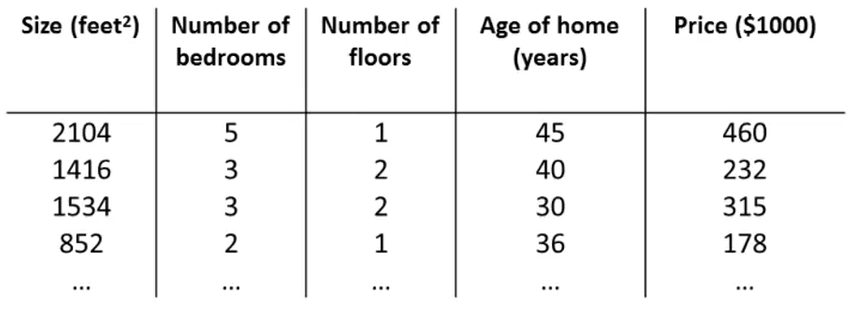
* Now we will look at the inverse element – the element such that when multiplied by the original, yields the identity element.
* In the space of , not all numbers have an inverse (). What is the inverse of a matrix?
* **Matrix Inverse:** If is an matrix (square matrix), and if it has an inverse, then .
* Let’s look at an example of this.
* The actual way to find the inverse of a matrix is shown below.
* This is only the case for a matrix. Also, if , then the inverse of matrix does not exist. Matrices that don’t have an inverse are called “singular” or “degenerate”.
* **Matrix Transpose:** Let be an matrix and let . Then is an matrix and,
* Let’s see the example below.

**Week 2**

**Multivariate Linear Regression**

Multiple Features

* In our previous gradient descent models, we just had one feature upon which we predicted the output. E.g. We used house size (ft2) to predict house price ($1000s).
* Now imagine we had multiple attributes to use to predict the price of the house as shown below.



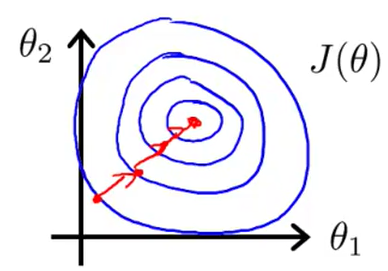
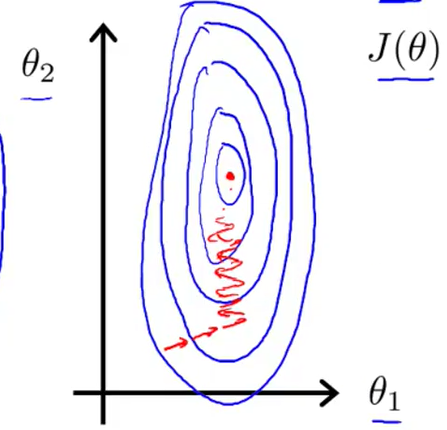
* We use to denote each of the features and to denote the output variable. In the case above .
* Here is some additional required notation:
  + = number of features
  + = input (features) of training example
  + = value of feature in training example
* In the example table above, we can use the notation to represent specific rows, columns, or cells.
* Now that we have multiple features, our hypothesis becomes the following.
* For convenience of notation and thus . Then, this vector that contains all the features becomes the following.
* We can also think of our parameters as a vector.
* Now we can rewrite our hypothesis as the following.

Gradient Descent for Multiple Variables

* Since now we are working with multivariate linear regression, our cost function becomes the following.
* In the equation above, is a dimensional vector.
* Here is the new gradient descent algorithm for features.
* We can expand the partial derivative term to get a new update rule for features.

Gradient Descent in Practice I – Feature Scaling

* Here is a practical idea to make **feature scaling** work.
  + **Idea:** make sure features are on a similar scale
  + E.g. = size (0–2000 ft2) and = number of bedrooms (1–5)
* When you have such different scales, when you plot the contour graph of and (ignoring ), you get a very narrow elliptical contour as shown below.



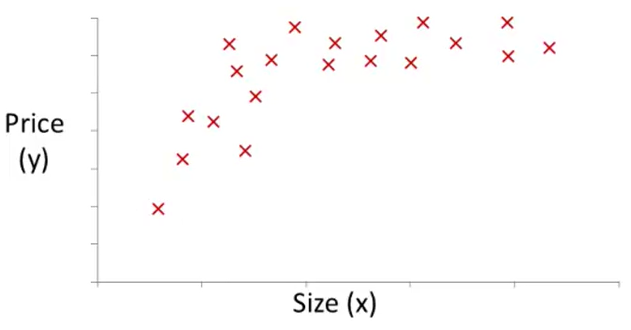
* Thus, the red arrows show that the gradient descent algorithm takes a long time to reach the global minimum.
* In turn, if you scale the features, then it is much easier for the gradient descent algorithm to move towards the global minimum in a smaller number of steps.
* Thus, you can apply a scaling technique to make all the features on a similar scale.
* In general, you want to get every feature into an approximate range of .
* In addition to feature scaling, it can be useful to apply **mean normalization**.
  + Replace with to make features have approximately zero mean. (Do not apply to .)
  + E.g. Let’s do this with the same example.
* In general, to make a feature scaled appropriately, we apply the following transformation to each element.
* In the equation, is the feature vector itself, is the average value of in the training set, and is the standard deviation of in the training set or the range ().

Gradient Descent in Practice II – Learning Rate

* Now, we will show how to choose an effective learning rate .
* Firstly, to make sure gradient descent is working properly, we can graph the number of iterations versus the .
* Overall, the graph should decrease over the number of iterations. An example of an automatic convergence test:
  + Declare convergence if decreases by less than in one iteration.
* If your graph is increasing, then you need to use a smaller .
  + For sufficiently small , should decrease on every iteration.
  + But if is too small, gradient descent may take very long to converge.

Features and Polynomial Regression

* Now we’ll look at how to select appropriate features for a model to develop a powerful algorithm to estimate the output.
* Let’s look at housing prices. Suppose you pick two features to model your hypothesis function.
* You might feel that it’s better to model house price based on the total area rather than these frontage and depth dimensions. Thus, you can make a new variable for area.
* You can also pick non-linear models if you think the curve doesn’t follow a straight line, as shown below.

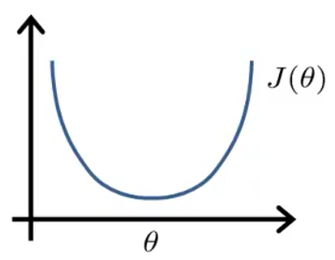


* You might feel that it follows a quadratic model better . However, a quadratic function does fall back down. So now we want to fit a cubic model .
* It turns out this requires a small modification to our original algorithm.
* Then we set the variables to the following.
* Then, we get the hypothesis function.
* If you do choose your features like this, then feature scaling is **very important**, since .
* You could also choose non-polynomial models to fit the data.
* Then we just do the same thing as before.

**Computing Parameters Analytically**

Normal Equation

* The method of the **normal equation** gives a way to solve for the parameters analytically.
* Let’s take a simplified cost function that only consists of one parameter. Thus , so it’s a one-dimensional vector or a scalar.



* To find the global minimum of this function, we must find where the derivative and solve for .
* However, when we work with many features, then . The cost function also becomes the following.
* Then, for every from to , you find where the partial derivative of the parameter is equal to .
* Let’s try to walk through an example using the normal equation method.



* Let’s pretend there are only training examples to work with. We have different features that we are looking at and an extra feature that just makes the math easier.
* Now let’s compile all the features into a single matrix . The output values will be stored in a vector .
* So is a matrix and is a vector (or a -dimensional vector). To analytically solve for the best parameters , you use the following formula.
* Let’s look at a very general case. Let’s say we have training examples ; and you have features. Each is a vector like the following.
* Then we have a matrix called the **design matrix**. We transpose each of the feature vectors and put it as a row vector into the matrix .
* For example, if I have an that only has one feature, we get the following vector.
* Then the design matrix is the following.
* The vector is still the same.
* Then we can calculate using .
  + This yields the optimal value of . When should each method be used?
  + Gradient Descent
    - Need to chose
    - Needs many iterations
    - Works well even when is large
  + Normal Equation
    - No need to chose
    - Don’t need to iterate
    - Need to compute
    - Slow if is very large

Normal Equation Non-Invertibility

* The normal equation is given by .
  + But what if is non-invertible (singular/degenerate)?
* can rarely be non-invertible. But it is non-invertible when you have one of two problems.
  + Redundant features (linearly dependent)
    - E.g. = size in ft2 and = size in m2
  + Too many features ()
    - Delete some features, or use regularization

**Octave/Matlab Tutorial**

Vectorization

* Let’s say we have the following hypothesis function.
* The above equations show that there is two ways to calculate the prediction output .
  + Un-vectorized implementation
    - **prediction = 0.0;**
    - **for j=1:n+1**
    - **prediction = prediction + theta(j) \* x(j);**
    - **end**
  + Vectorized implementation
    - **prediction = theta’ \* x**
* Let’s look at a more sophisticated and involved example. Say we want to calculate the gradient descent until convergence for all .
* We could do this using a for loop, but that would be extremely tedious and inefficient. Below is a vectorized solution.
  + Vectorized implementation
    - **theta = theta – alpha \* delta**
    - where **delta** =

**Week 3**

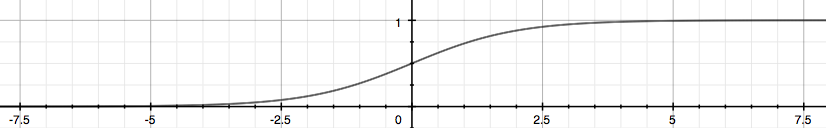
**Classification and Representation**

Classification

* In classification problems, the output variable is a discrete-valued output. The most popular tool to solve this problem is called **logistic regression**.
* Here are some examples of classification problems.
  + Email: spam / not spam?
  + Online Transactions: fraudulent (yes / no)?
  + Tumour: malignant / benign?
* In all cases, the output variable can take on two values since it is a **binary classification problem**.
* represents the “negative class” (e.g. benign tumour), while represents the “positive class” (e.g. malignant tumour).
* Later, we’ll talk about multiclass problems, where , where is the number of classes.
* Applying a linear regression to a classification problem isn’t a good idea – often times, the straight line gets skewed by some of the data points and you don’t have a good idea of how to classify the points anymore.
* A logistic regression, in turn, has the property that .

Hypothesis Representation

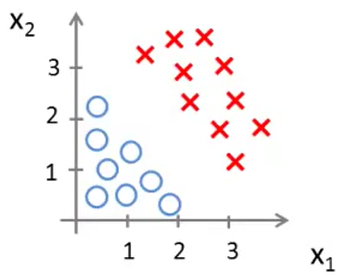
* Our logistic hypothesis must satisfy the property that . We’re going to make a small modification to our linear hypothesis shown below.
* The function , where , denotes the following function.
* The function above is called the **sigmoid function** or the **logistic function**. When we substitute the function into our hypothesis, we get the following.
* Below is a sketch of what this function looks like.



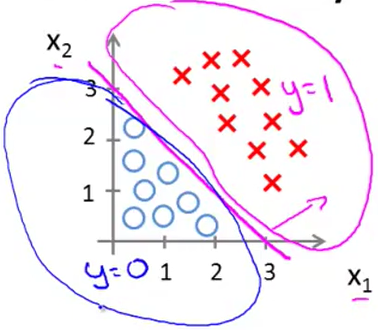
* Thus, this satisfies the property that .
* Here is the interpretation of the output of the hypothesis function .
  + = estimated probability that on input
  + “Probability that , given , parameterized by .”
* Also, we know that the probability that and will add up to .

Decision Boundary

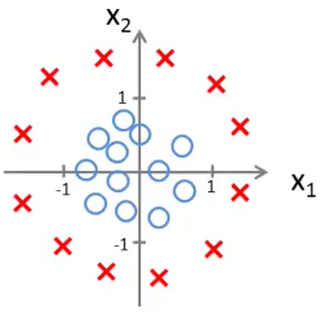
* One way to decide about whether an example falls into or is by the following.
  + Predict if
  + Predict if
* From the graph of the function we can tell that the sigmoid function when . Thus, the hypothesis function when .
* On the other side, by a similar argument when .
* Now let’s see how this can help us make decisions on an actual training set, given the graph below.



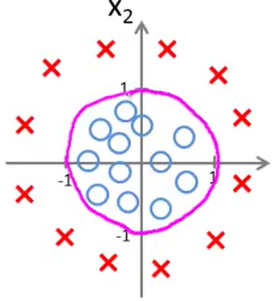
* Let’s say our hypothesis for this data set is the following function.
* Now let’s assume that our vector is the following. Although we don’t yet know how to calculate our parameters , we can assume it is given.
* Thus, we can predict that if . Values of that satisfy this inequality yield exactly.
* On the graph, this is shown as below. The line cutting between the two halves of the plane is called the **decision boundary**.



* But what if our boundary is non-linear?



* Then our hypothesis function might look like this.
* By some procedure, let’s say our parameter choices vector ends up like this.
* Then we predict that if .

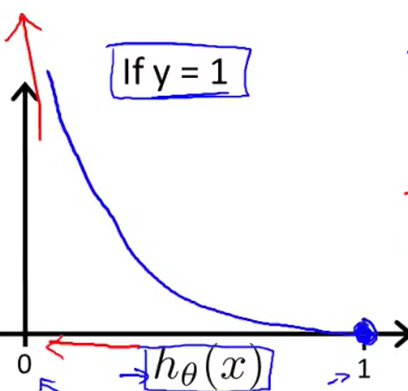


* Anything outside or on the pink line is and anything inside is .

**Logistic Regression Model**

Cost Function

* The optimization objective of this model is to choose the ideal set of parameters to get the best approximation to the decision boundary.
* Unfortunately, we can’t use the same cost function as we did with linear regression (the sum of the squared error terms).
* When we use the function , then the cost function yields a **non-convex** function – it has more than one local optimum. Thus, it cannot guarantee to be globally optimized using the gradient descent method.
* Here is the cost function we’ll use for the logistic regression model.
* The cost function looks something like this **if** . If , the curve would be flipped around the line .



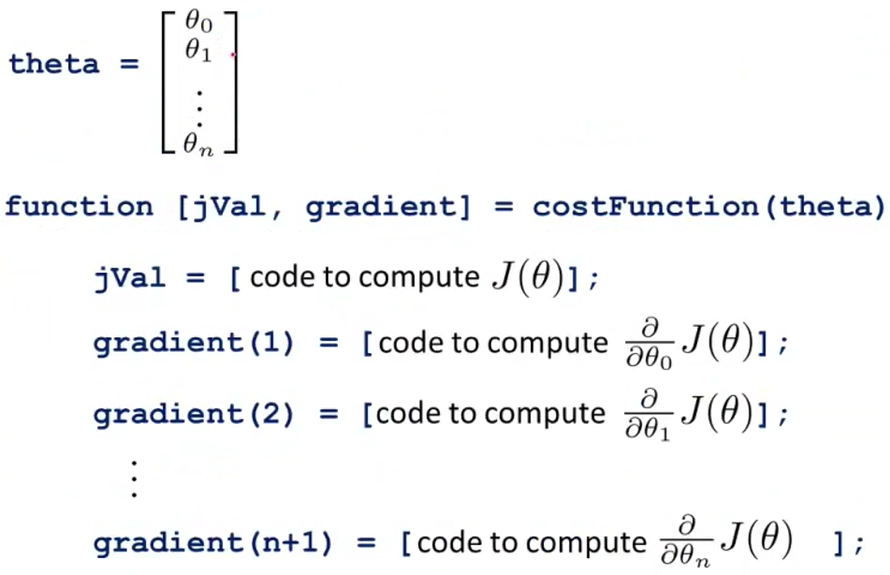
* This captures the intuition that if (predict that ), but actually equals , then penalize the algorithm by a very, very large cost. The vice versa happens when .

Simplified Cost Function and Gradient Descent

* There’s a simpler way to rewrite the cost function we encountered in the last video.
* If , then .
* If , then .
* Then, the overall cost function becomes the following.
* Now we must find .
* To do this, we will use gradient descent.
* A vectorized implementation for this is the following.

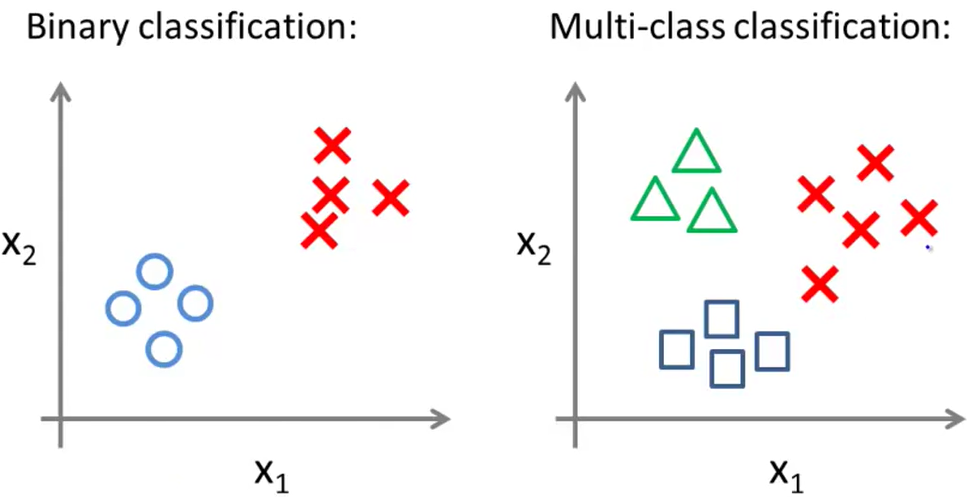
Advanced Optimization

* Other than gradient descent, there are a variety of optimization algorithms that solve the same problem.
  + Gradient Descent
  + Conjugate Gradient
  + BFGS (Broyden-Fletcher-Goldfarb-Shanno Algorithm)
  + L-BFGS (Limited Memory-BFGS Algorithm)
* Here are the advantages and disadvantages of these algorithms.
  + Advantages
    - No need to pick
    - Often faster than gradient descent
  + Disadvantages
    - More complex
* Let’s see how to implement one in Octave/Matlab.
  + E.g. .
* To implement this, you would create the following function.
  + **function [jVal, gradient] = costFunction(theta)**
  + **jVal = (theta(1) – 5)^2 + (theta(2) – 5)^2;**
  + **gradient = zeros(2, 1);**
  + **gradient(1) = 2\*(theta(1) – 5);**
  + **gradient(2) = 2\*(theta(2) – 5);**
  + **end**
* Then, you would minimize this function by using a Matlab library.
  + **options = optimset(‘GradObj’, ‘on’, ‘MaxIter’, ‘100’);**
  + **initialTheta = zeros(2, 1);**
  + **[optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options);**
* If we want to apply this to logistic regression more generally, we create the following function.

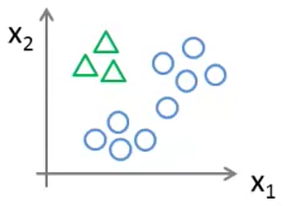


Multiclass Classification: One-vs-All

* Here are some examples of multiclass classification problems.
  + Email foldering / tagging: Work, Friends, Family, Hobby
  + Medical diagrams: Not ill, Cold, Flu
    - Not ill (), Cold (), Flu ()
  + Weather: Sunny, Cloudy, Rainy, Snowy



* To create decision boundaries around all 3 classes, we essentially separate out the data set. So the first data set may look like this.

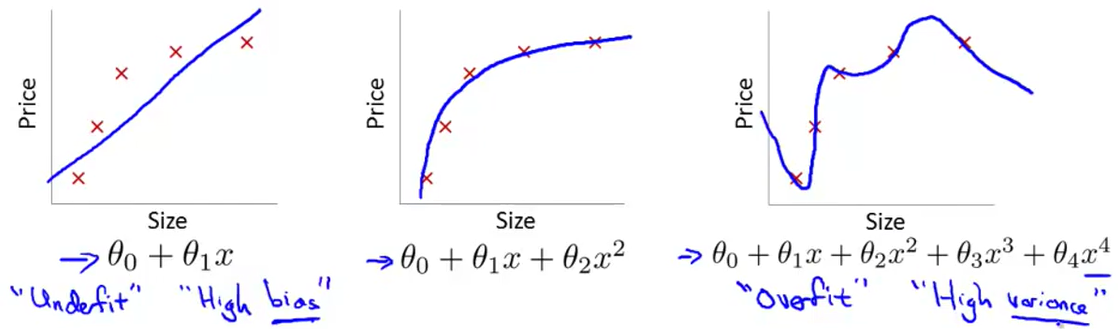


* The triangles are treated as 1 class and everything else is treated as another class. As a formula, we have different hypothesis functions based on which class we are looking at.

**Regularization**

The Problem of Overfitting

* When deciding what model to use to fit a curve, there are a couple of problems, specifically **underfitting** and **overfitting**.



* **Overfitting:** If we have too many features, the learned hypothesis may fit the training set very well (), but may fail to generalize to new examples.
* If a training set is being overfitted, there are a couple things to remedy the problem.

1. Reduce number of features.

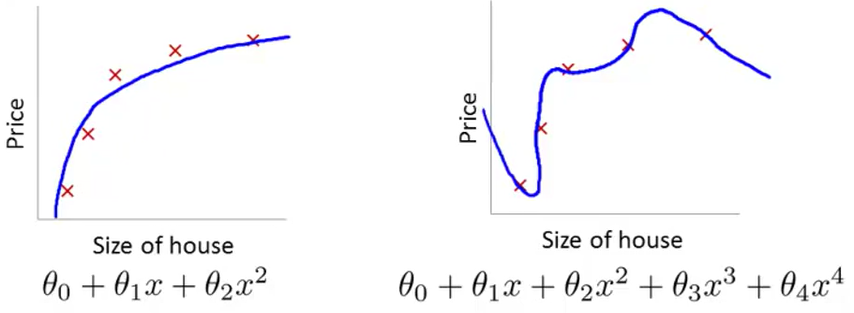
* Manually select features to keep.
* Model selection algorithm (later).

1. Regularization

* Keep all features, but reduce parameters/values of .
* Works well with lots of features, each which contributes a little bit to predicting .

Cost Function

* Suppose we have two different models to predict housing prices.



* Then, we want to make and smaller because they are contributing to the awkward behaviour of the function.
* Then, we can modify the cost function to the following.
* Thus, when we minimize the function, we get and . And we essentially end up with a quadratic function with small contributions from that make it more accurate.
* If we want to shrink all our parameters, then we can add an extra term at the end of the cost function.

Regularized Linear Regression

* When you apply the update rule on linear regression with a regularized term at the end, you get the following.
* Usually, the term is slightly less than , when is small and is quite large. Thus, the change is not big enough to completely change the function, but remove some of its irregularities.
* Now, we also used the normal equation method to solve for the parameters. How would that be fixed?
* Well the normal equation with the regularized term would be the following.
* The large matrix in the equation is an identity matrix of size , except that the top-left element of the matrix is .

Regularized Logistic Regression

* Here is the regularized logistic cost function.
* Then, the new gradient descent algorithm becomes the following.