Machine Learning

Two definitions of Machine Learning are offered.

Arthur Samuel in 1959 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 in 1998 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 two broad classifications:
Supervised learning and Unsupervised learning

The Machine Learning Pipeline

Phase 1: Data processing

This is where you format the data in such a way algorithms can ingest. It uses Linear Algebra. Here we do:

- Data collection
- Data formatting
- Labelling

Phase 2: Feature engineering and Selection

This is where you transform the data to make it easy for algorithm to understand. It uses Vectors and Matrices. These are fundamental data structures in ML and mathematics. You can use linear algebra to perform multiplication and addition on Vectors and Matrices.

Phase 3: Modeling

This is where you define the problem in such a way algorithm can optimize. It uses below for loss functions:

- Geometry - How close the things are to the target answers.
- Probability - How likely your data to be generated for the process defined. Probability of data being generated that gives you certains ways for evaluating the predictions.
- Norms - How close it is in producing true value observed.
- Statistics - All of this is formed by statistics.

GOAL - Learn what is driving observed events.

Phase 4: Optimization

This is where you iterate until certain conditions are met and then you choose the best model. It uses Vector Calculus and Numerical Methods.

Vector Calculus - Take derivative of your loss function and try to learn how to optimize on those steps.
Numerical Methods - Gradient Descent.

Take some family of models and pick that one which is most likely to predict your data. It has:

- Training Phase - Take large collection of data points and try to optimize over those data points.
- Data Evaluation - Data that has not been seen before and to see if it can predict over those data points.
- Prediction - Go out in the real world and apply in practice.

In basic terms, ML is the process of training a piece of software, called a model, to make useful predictions using a data set. This predictive model can then serve up predictions about previously unseen data. We use these predictions to take action in a product; for example, the system predicts that a user will like a certain video, so the system recommends that video to the user.

Often, people talk about ML as having two paradigms, supervised and unsupervised learning.
There are others as well: Reinforcement learning, recommender systems

Supervised Learning:

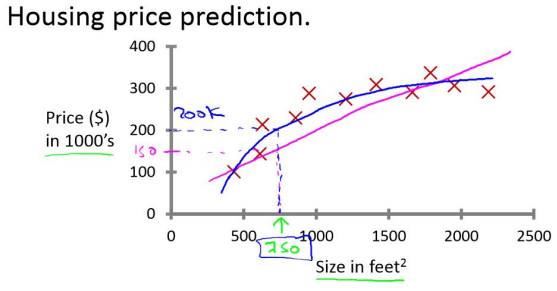
Supervised learning is a type of ML where the model is provided with **labeled** training data. Features are measurements or descriptions; the label is essentially the "answer."

In **supervised machine learning**, you feed the features and their corresponding labels into an algorithm in a process called **training**. During training, the algorithm gradually determines the relationship between features and their corresponding labels. This relationship is called the **model**.

To tie it all together, supervised machine learning finds patterns between data and labels that can be expressed mathematically as functions. Given an input feature, you are telling the system what the expected output label is, thus you are supervising the training. The ML system will learn patterns on this labeled data. In the future, the ML system will use these patterns to make predictions on data that it did not see during training.

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:

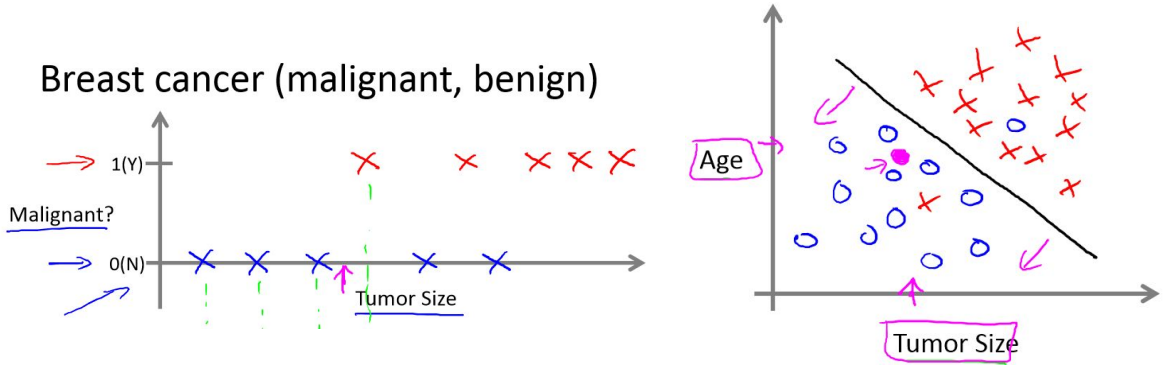


Given data about the size of houses on the real estate market, try to predict their price. Price as a function of size is a continuous output, so this is a regression problem.

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.

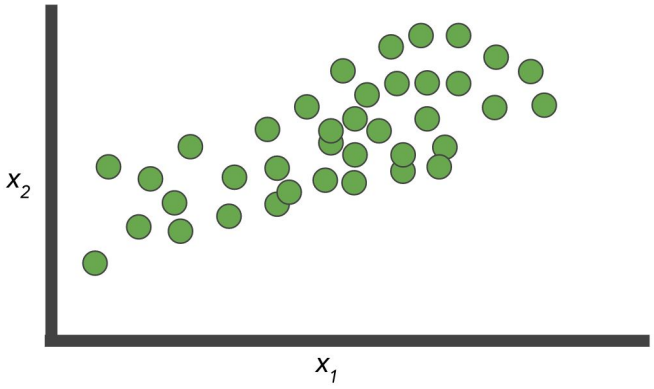
(b) Classification - Given a patient with a tumor, we have to predict whether the tumor is malignant or benign.



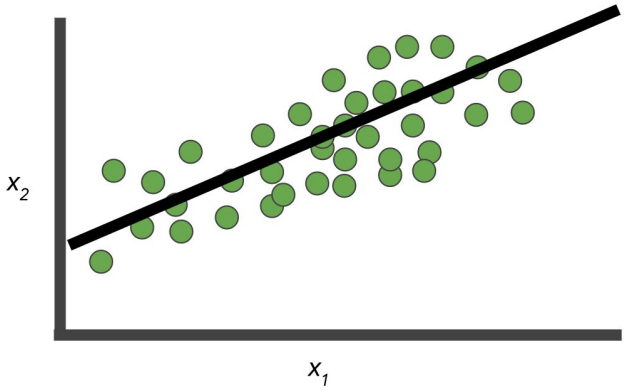
Unsupervised Learning

In unsupervised learning, the goal is to identify meaningful patterns in the data. To accomplish this, the machine must learn from an unlabeled data set. In other words, the model has no hints how to categorize each piece of data and must infer its own rules for doing so.

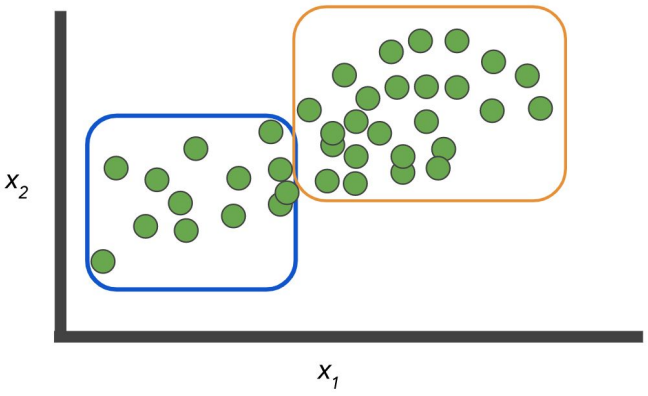
We can derive this structure by clustering the data based on relationships among the variables in the data. With unsupervised learning there is no feedback based on the prediction results.



In the following graph, all the examples are the same shape because we don't have labels to differentiate between examples of one type or another here



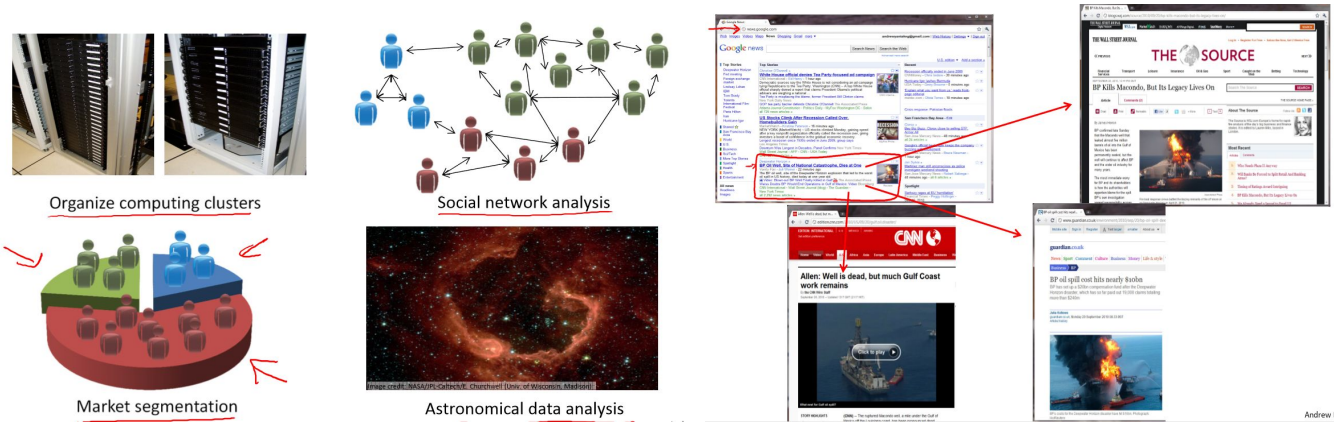
Fitting a line to unlabeled points isn't helpful. We still end up with examples of the same shape on both sides of the line. Clearly we will have to try a different approach.



Here, we have two clusters. (Note that the number of clusters is arbitrary). What do these clusters represent? It can be difficult to say. Sometimes the model finds patterns in the data that you don't want it to learn, such as stereotypes or bias.

However, when new data arrives, we can categorize it pretty easily, assuming it fits into a known cluster. But what if your photo clustering model has never seen a pangolin before? Will the system cluster the new photo with armadillos or maybe hedgehogs?

Examples of Unsupervised Machines Learning



Types of ML Problems

There are several subclasses of ML problem based on what the prediction task looks like. In the table below, you can see examples of common supervised and unsupervised ML problems.

Type of ML Problem	Description	Example
Classification	Pick one of N labels	Cat, dog, horse, or bear
Regression	Predict numerical values	Click-through rate
Clustering	Group similar examples	Most relevant documents (unsupervised)
Association rule learning	Infer likely association patterns in data	If you buy hamburger buns, you're likely to buy hamburgers (unsupervised)
Structured output	Create complex output	Natural language parse trees, image recognition bounding boxes
Ranking	Identify position on a scale or status	Search result ranking

Contrasting Cases

As you walk through each example, note the types of data used and how that data informed the product design and iterations. Think about how the examples compare to and contrast from each other. Click on each product name button to see more information below.

SMART REPLY | YOUTUBE WATCH NEXT | CUCUMBER SORTING

Suggested short responses to emails.

Smart Reply is an example of ML that utilizes Natural Language Understanding (NLU) and generation, sequence-to-sequence learning, to make replying to a flooded inbox far less painful.

SMART REPLY | YOUTUBE WATCH NEXT | CUCUMBER SORTING

YouTube Watch Next uses ML to generate the list of video recommendations after you've watched a video on YouTube. It is a large scale recommendation system using deep networks to generate and rank potential videos.

- Deep Neural Networks for YouTube Recommendations

SMART REPLY | YOUTUBE WATCH NEXT | CUCUMBER SORTING

See how a cucumber farmer is using machine learning to sort cucumbers by size, shape, color, and other attributes.

- How a Japanese cucumber farmer is using deep learning and TensorFlow

Thought Questions

Think about the similarities and differences between each of the above cases. Click on the arrow to expand the section and reveal the answers.

What user problem did these systems solve?

In all three cases there was motivation to build an ML system to address a real problem users were facing.

- Smart Reply: responding to emails can take up too much time
- YouTube: there are too many videos on YouTube for one person to navigate and find videos they like
- Cucumber sorter: the cucumber sorting process is burdensome

What does output from these systems look like?

Each is a bit different.

- Smart Reply: three short suggested responses at the bottom of an email
- YouTube: suggested videos along the right-hand side of the screen
- Cucumber sorter: directions to a robot arm that sorts cucumbers into their correct categories

What data sources were used?

In all three cases the large amounts of historical data had information closely tied to what we wanted to do.

- Smart Reply: conversation data (email messages and responses)
- YouTube: watch time, click-through rate, watch history, search history
- Cucumber sorter: exemplary cucumber data (size, shape, weight, etc.)

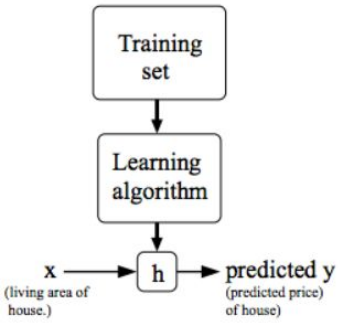
Scientific Method

To address the challenges of transitioning to ML, it is helpful to think of the ML process as an experiment where we run test after test after test to converge on a workable model. Like an experiment,the process can be exciting, challenging, and ultimately worthwhile.

Step	Example
1. Set the research goal.	I want to predict how heavy traffic will be on a given day.
2. Make a hypothesis.	I think the weather forecast is an informative signal.
3. Collect the data.	Collect historical traffic data and weather on each day.
4. Test your hypothesis.	Train a model using this data.
5. Analyze your results.	Is this model better than existing systems?
6. Reach a conclusion.	I should (not) use this model to make predictions, because of X, Y, and Z.
7. Refine hypothesis and repeat.	Time of year could be a helpful signal.

Model Representation

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.

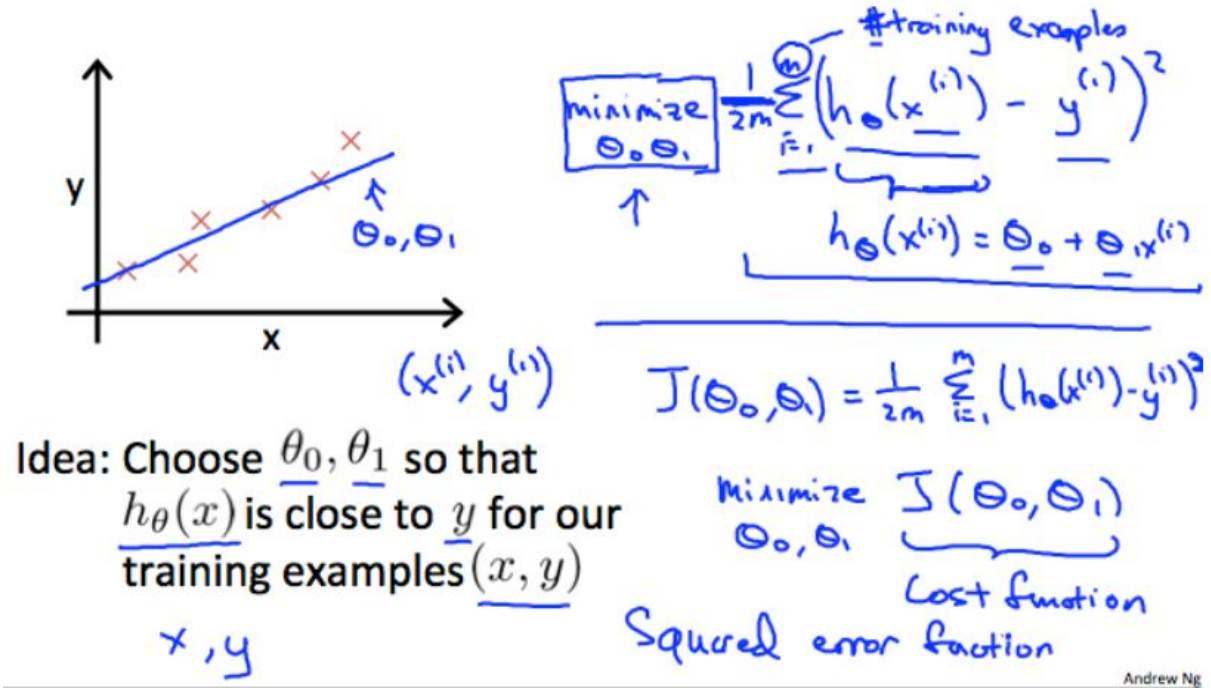
Cost Function

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.

$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

To break it apart, it is $\frac{1}{2}\,\bar{x}$ where $\bar{x}$ is the mean of the squares of $h_\theta(x_i) - y_i$, or the difference between the predicted value and the actual value.

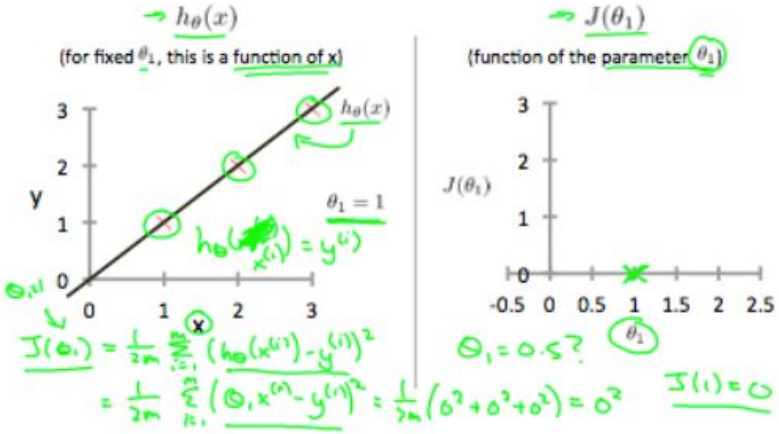
M is the number of training examples. This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved (½) as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the ½ term. The following image summarizes what the cost function does:



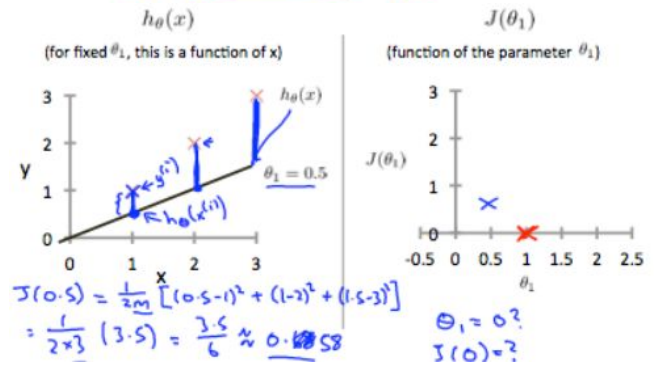
where theta 0 and theta 1 are the parameters or the weights used with X values to get the good predictions, so we must choose good value of theta such that our cost function can be minimized.

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(θ(*x*)) which passes through these scattered data points.

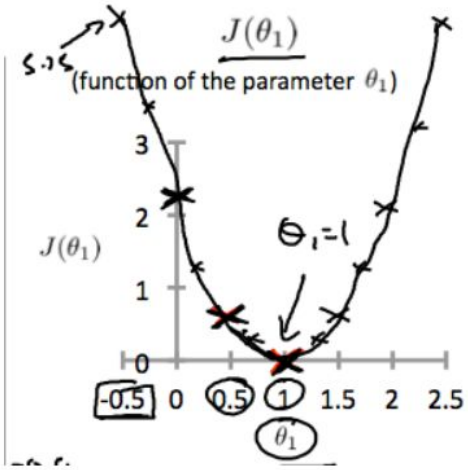
Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. Ideally, the line should pass through all the points of our training data set. In such a case, the value of *J*(θ0, θ1) will be 0.



The following example shows the ideal situation where we have a cost function of 0.



When θ1=1, we get a slope of 1 which goes through every single data point in our model. Conversely, when θ1=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 is our global minimum.

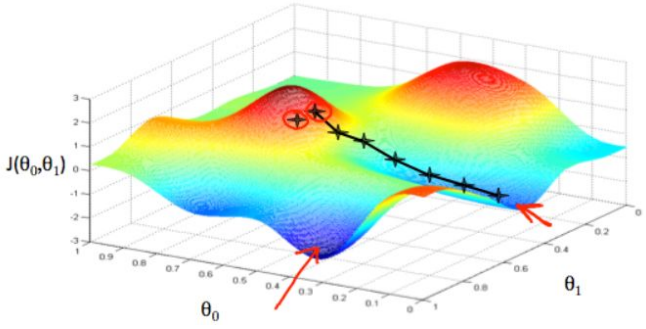
Theta 0 is also known as intercept which shows the point where the estimated regression line crosses the y axis. It is the value of the estimated response $f(x)$ for x = 0. The value of Theta 1 determines the slope of the estimated regression line which is known as coefficient.

Your graph can be ascending, descending or straight line based on the values of theta (Intercept or Coefficient).

Gradient Descent

We use gradient descent to update the parameters of our model. Parameters refer to coefficients in Linear Regression and weights in neural networks.
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(intercept and coefficients or bias) in the hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields $\theta 0$ and $\theta 1$ (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 $\theta 0$ on the x axis and $\theta 1$ 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.

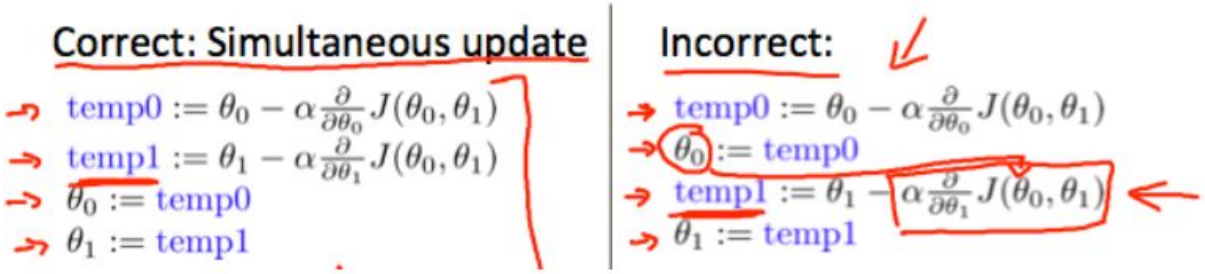
For example, the distance between each 'star' in the graph above represents a step determined by our parameter α. A smaller α would result in a smaller step and a larger α results in a larger step. The direction in which the step is taken is determined by the partial derivative of $J(\theta 0, \theta 1)$. 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:

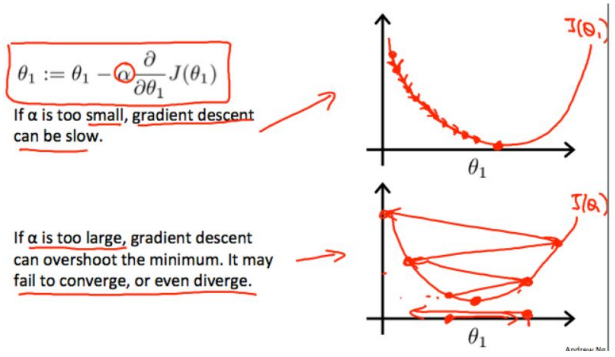
repeat until convergence:

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

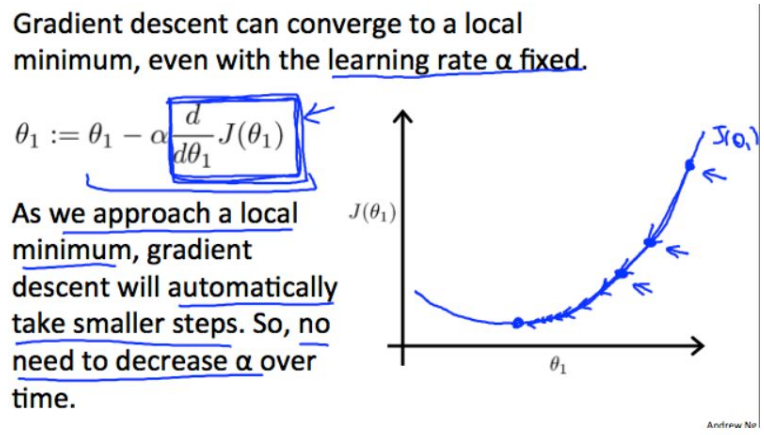
Where j = 0,1 represents the feature index number. At each iteration j, one should simultaneously update the parameters $\theta 1, \theta 2, \ldots, \theta n$. Updating a specific parameter prior to calculating another one on the *j*(*th*) iteration would yield to a wrong implementation.



If slope (which is partial derivative of cost function) <= 0 (negative) then value of theta will increase else it will decrease.



On a side note, we should adjust our parameter 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.



Gradient Descent For Linear Regression

repeat until convergence: {

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

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

}

where m is the size of the training set, $\theta 0$ a constant that will be changing simultaneously with $\theta 1$ and *xi*,*yi* are values of the given training set (data).

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.

Multivariate Linear Regression

Linear regression with multiple variables is also known as "multivariate linear regression". We now introduce notation for equations where we can have any number of input variables.

$x_j^{(i)}$ = value of feature j in the i^{th} training example
$x^{(i)}$ = the input (features) of the i^{th} training example
m = the number of training examples
n = the number of features

Vectorize form of hypothesis function: Theta.transpose * X

Gradient Descent for Multiple Variables

repeat until convergence: {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j} := 0...\text{n}$$

}

Gradient Descent in Practice I - Feature Scaling

We can speed up gradient descent by having each of our input values in roughly the same range. **This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very**

uneven. The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ideally:
$-1 \leq x(i) \leq 1$
or
$-0.5 \leq x(i) \leq 0.5$
These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.
Two techniques to help with this are **feature scaling and mean normalization**. Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero. To implement both of these techniques, adjust your input values as shown in this formula:

$$x_i := \frac{x_i - \mu_i}{s_i}$$

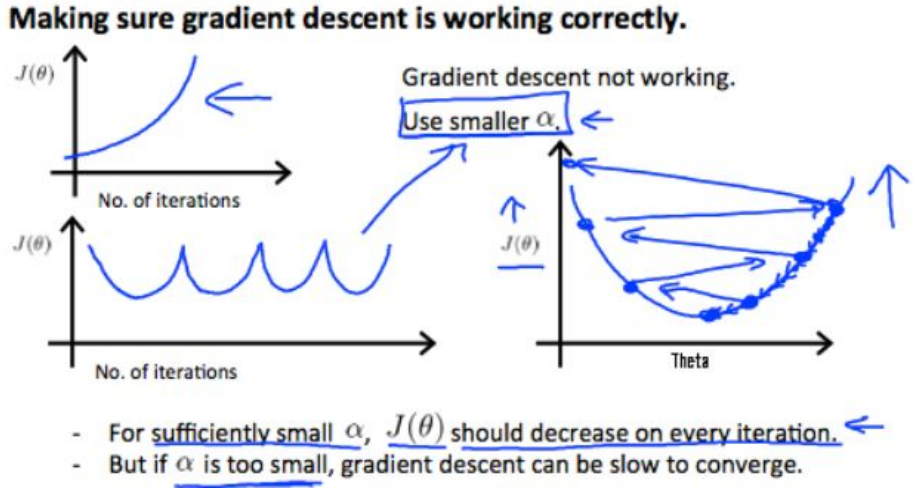
Where *μi* is the average of all the values for feature (i) and *si* is the range of values (max - min), or *si* is the standard deviation.

Gradient Descent in Practice II - Learning Rate

Debugging gradient descent. Make a plot with *number of iterations* on the x-axis. Now plot the cost function, J(θ) over the number of iterations of gradient descent. If J(θ) ever increases, then you probably need to decrease α.

Automatic convergence test. Declare convergence if J(θ) decreases by less than E in one iteration, where E is some small value such as 10^{-3}. However in practice it's difficult to choose this threshold value.

It has been proven that if learning rate α is sufficiently small, then J(θ) will decrease on every iteration.



To summarize:
If α is too small: slow convergence. If α is too large: may not decrease on every iteration and thus may not converge.

Features and Polynomial Regression

We can improve our features and the form of our hypothesis function in a couple different ways. We can combine multiple features into one. For example, we can combine

x_1 and x_2 into a new feature x_3 by taking x_1*x_2.

Polynomial Regression

Our hypothesis function need not be linear (a straight line) if that does not fit the data well. We can change the behavior or curve of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).
For example, if our hypothesis function is
$h_\theta(x) = \theta_0 + \theta_1 x_1$ then we can create additional features based on x_1, to get the quadratic
function.
$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$ or the cubic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$ To make it a square root function, we could do:

One important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important.
eg. if x_1 has range 1 - 1000 then range of *square(*x_1) becomes 1 - 1000000 and that of cube(x_1) becomes 1 - 1000000000

Gradient descent gives one way of minimizing J. Let's discuss a second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm. In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θ_j's, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:

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

There is no need to do feature scaling with the normal equation.
The following is a comparison of gradient descent and the normal equation:

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of $X^T X$
Works well when n is large	Slow if n is very large