Week 1 Lecture Notes

Wednesday, April 28, 2021 10:28 AM

ML:Introduction

What is Machine Learning?

Two definitions of Machine Learning 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 two broad classifications:

supervised learning, OR

unsupervised learning.

Wednesday, April 28, 2021 10:29 AM

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*. Here is a description on Math is Fun on Continuous and Discrete Data.

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 Male/Female, We have to predict his/her age on the basis of given picture.

(b) Classification - Given a picture of Male/Female, We have to predict Whether He/She is of High school, College, Graduate age. Another Example for Classification - Banks have to decide whether or not to give a loan to someone on the basis of his credit history.

Unsupervised Learning

Unsupervised learning, on the other hand, *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, i.e., there is no teacher to correct you.

Example:

Clustering: Take a collection of 1000 essays written on the US Economy, and find a way to automatically group these essays into a small number that are somehow similar or related by different variables, such as word frequency, sentence length, page count, and so on

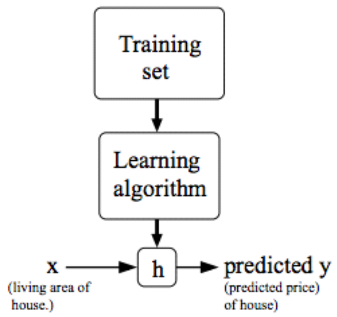
Non-clustering: The "Cocktail Party Algorithm", which can find structure in messy data (such as the identification of individual voices and music from a mesh of sounds at a cocktail party (https://en.wikipedia.org/wiki/Cocktail_party_effect)). Here is an answer on Quora to enhance your understanding. : https://www.quora.com/What-is-the-difference-between-supervised-and-unsupervised-learning-algorithms ?

Wednesday, April 28, 2021 7:49 PM

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.

Recall that in *regression problems*, we are taking input variables and trying to fit the output onto a *continuous* expected result function.

Linear regression with one variable is also known as "univariate linear regression."

Univariate linear regression is used when you want to predict a **single output** value y from a **single input** value x. We're doing **supervised learning** here, so that means we already have an idea about what the input/output cause and effect should be.

Wednesday, April 28, 2021 7:55 PM

The Hypothesis Function

Our hypothesis function has the general form:

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

Note that this is like the equation of a straight line. We give to $h_\theta(x)$ values for θ_0 and θ_1 to get our estimated output $\hat{y}$. In other words, we are trying to create a function called h_θ that is trying to map our input data (the x's) to our output data (the y's).

Example:

Suppose we have the following set of training data:

input x	**output y**
0	4
1	7
2	7
3	8

Now we can make a random guess about our h_θ function: $\theta_0 = 2$ and $\theta_1 = 2$. The hypothesis function becomes $h_\theta(x) = 2 + 2x$.

So for input of 1 to our hypothesis, y will be 4. This is off by 3. Note that we will be trying out various values of θ_0 and θ_1 to try to find values which provide the best possible "fit" or the most representative "straight line" through the data points mapped on the x-y plane.

Thursday, April 29, 2021 9:58 AM

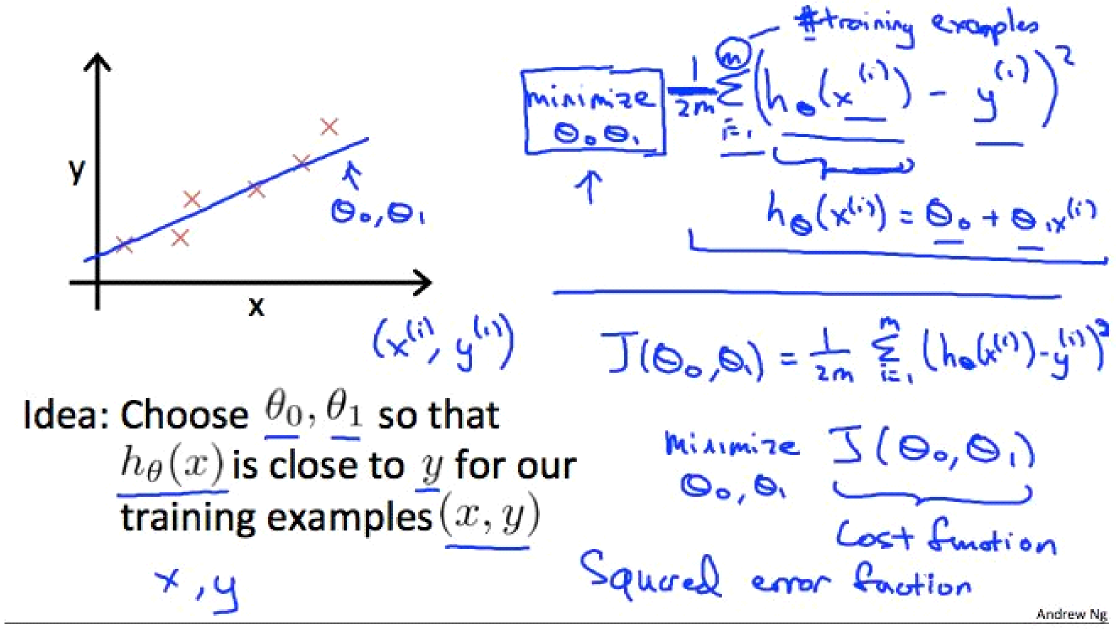
Cost Function

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.

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

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.

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



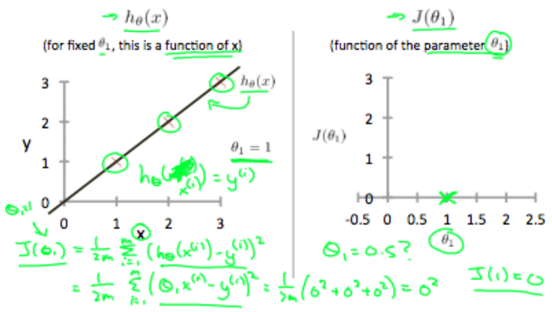
y
x
θ_0, θ_1
$(x^{(i)}, y^{(i)})$
Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)
x, y
# training examples
$\underset{\theta_0 \theta_1}{\text{minimize}} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$
$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
$\underset{\theta_0, \theta_1}{\text{minimize}} \; J(\theta_0, \theta_1)$
Cost function
Squared error function
Andrew Ng

Thursday, April 29, 2021 12:16 PM

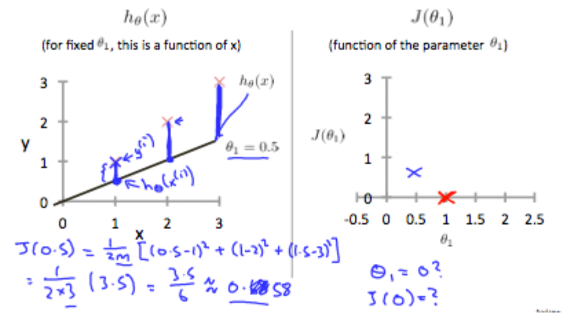
Cost Function - Intuition I

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)$h_\theta(x)$) which passes through these scattered data points.

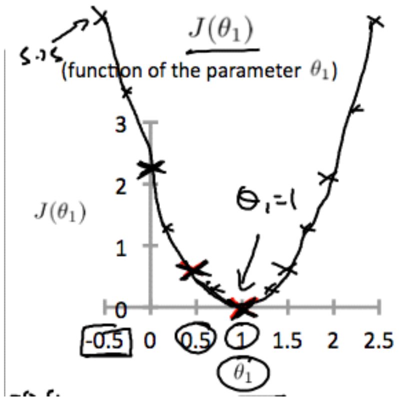
Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. Ideally, the line should pass through all the points of our training data set. In such a case, the value of J(\theta_0, \theta_1)$J(\theta_0,\theta_1)$ will be 0. The following example shows the ideal situation where we have a cost function of 0.



When \theta_1 = 1$\theta_1=1$, we get a slope of 1 which goes through every single data point in our model. Conversely, when \theta_1 = 0.5$\theta_1=0.5$, 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:



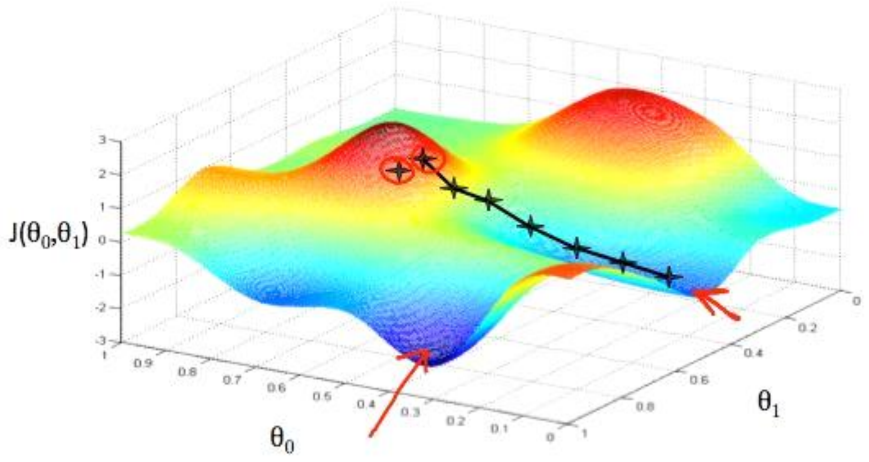
Thus as a goal, we should try to minimize the cost function. In this case, \theta_1 = 1$\theta_1=1$ is our global minimum.

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 θ_0 and 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 θ_0 on the x axis and 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 $J(\theta_0, \theta_1)$. 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:

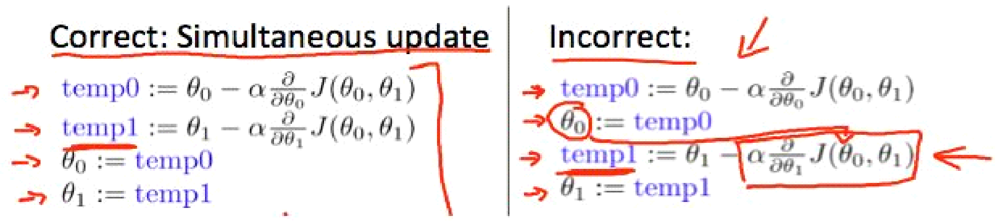
repeat until convergence:

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

where

j=0,1 represents the feature index number.

At each iteration j, one should simultaneously update the parameters $\theta_1, \theta_2, ..., \theta_n$. Updating a specific parameter prior to calculating another one on the $j^{(th)}$ iteration would yield to a wrong implementation.



Correct: Simultaneous update

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

$\theta_0 := \text{temp0}$

$\theta_1 := \text{temp1}$

Incorrect:

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\theta_0 := \text{temp0}$

$\text{temp1} := \theta_1 - \boxed{\alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)}$

$\theta_1 := \text{temp1}$

Friday, April 30, 2021 11:33 AM

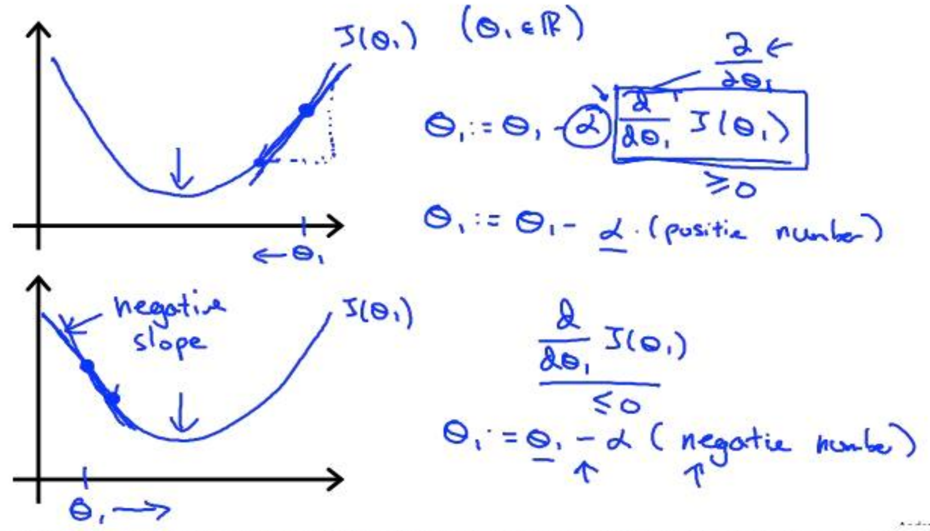
Gradient Descent Intuition

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 :

Repeat until convergence:

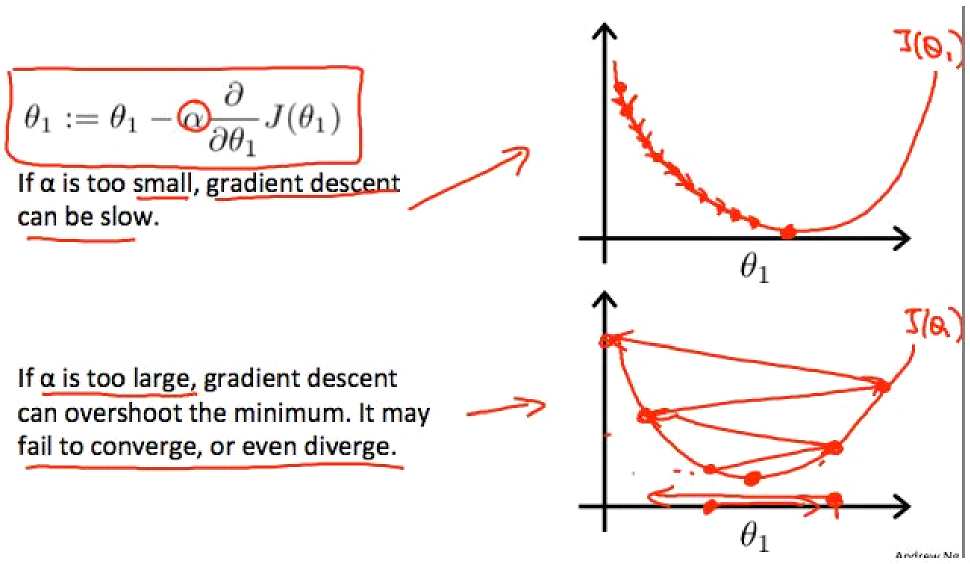
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Regardless of the slope's sign for $\frac{d}{d\theta_1} J(\theta_1)$, θ_1 eventually converges to its minimum value. The following graph shows that when the slope is negative, the value of θ_1 increases and when it is positive, the value of θ_1 decreases.



$J(\theta_1)$
$(\theta_1 \in \mathbb{R})$
$\frac{\partial}{\partial \theta_1}$
$\theta_1 := \theta_1 - \alpha \boxed{\frac{d}{d\theta_1} J(\theta_1)}$
≥ 0
$\theta_1 := \theta_1 - \alpha \cdot$ (positive number)
$\leftarrow \theta_1$
negative slope
$J(\theta_1)$
$\frac{d}{d\theta_1} J(\theta_1)$
≤ 0
$\theta_1 := \theta_1 - \alpha$ (negative number)
$\theta_1 \rightarrow$

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.



$$\boxed{\theta_1 := \theta_1 - \bigcirc\!\!\!\!\!\alpha \frac{\partial}{\partial\theta_1} J(\theta_1)}$$

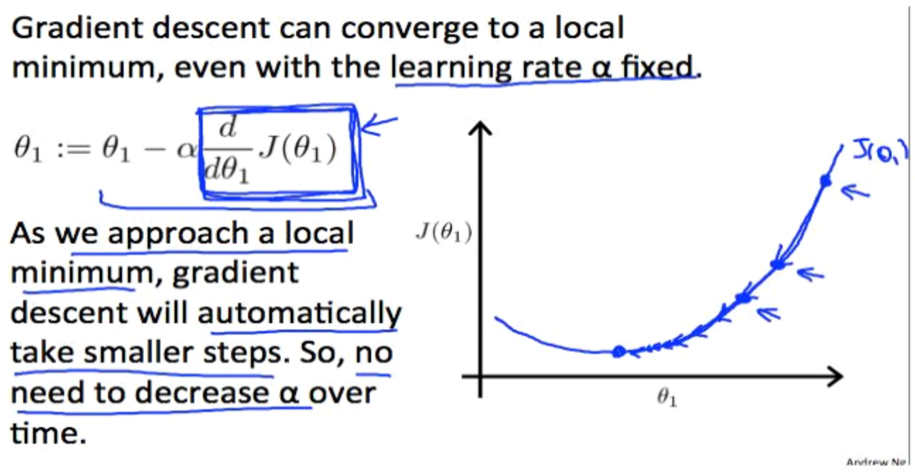
If α is too small, gradient descent can be slow.

If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

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

The intuition behind the convergence is that $\frac{d}{d\theta_1}J(\theta_1)$ approaches 0 as we approach the bottom of our convex function. At the minimum, the derivative will always be 0 and thus we get:

$$\theta_1 := \theta_1 - \alpha * 0$$



Gradient descent can converge to a local minimum, even with the learning rate α fixed.

$$\theta_1 := \theta_1 - \alpha \boxed{\frac{d}{d\theta_1} J(\theta_1)}$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.