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.
  - Regression: In 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.
  - Classification: In classification problem, We are instead trying to predict results in a discrete output. In other words, We're trying to map input variables into discrete categories.

Ex 1.

Given data about the size of houses on the real estate market, try to predict their price. Price is a function of size in 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 in to two discrete categories.

Ex 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'Ve 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 ~~on~~ 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.

EX:

- 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~~

- 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 ~~mesh~~ mixture of sounds at a cocktail party.

[ML: Linear Regression With One Variable]

* Model Representation:

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.

* 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 it $h_\theta(x)$ values for θ_0 and θ_1 to get our estimated output $\hat{y}$. In other words, we're 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've the following set of training data:

Input x	output y
0	4
1	1
2	1
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 $h_\theta(x) = 2 + 2x$.

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

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~~ through the data points mapped on the X-Y plane.

* Cost function:

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

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

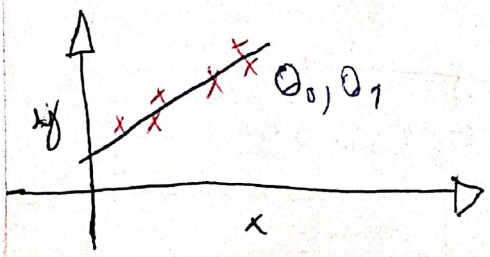
To be clear it out, it is $\frac{1}{2}\bar{x}$ where $\bar{x}$ is the mean of the squares of $h_\theta(x_i) - y_i$, or the difference between the predicted value and the actual value.

This function is otherwise called the "squared error function", or "mean squared error". The mean is halved ($\frac{1}{2m}$) 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. (to divide this into parts mentally)

Now We are able to concretely measure the accuracy of our prediction function against the correct results We have so that We can predict new results We don't have.

If we try to think of it in visual terms, our training dataset is scattered on the x-y plane. We are trying to make straight line (defined by $h_\theta(x)$) which passes through this scattered set of data. Our objective is to get the best possible line. The best possible line will be such as that the average squared vertical distances of the scattered points from the line will be the least. In the best case, the line should pass through all points of our training data set. In such a case the value of $J(\theta_0, \theta_1)$ will be 0.

Video: Cost function: 4:54



$$\underset{\theta_0,\theta_1}{\text{minimize}} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

m → Training examples

$h_\theta(x^{(i)})$ → $h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$

Minimizing this expression

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

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

$$\underset{\theta_0,\theta_1}{\text{minimize}} \; J(\theta_0, \theta_1)$$

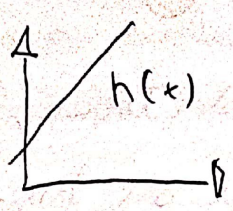
Cost function

Squared error cost function.

Video: Cost function: I J

Hypothesis:

$h_\theta(x) = \theta_0 + \theta_1 x$



Parameters:

θ_0, θ_1

Cost function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$ minimizing $J(\theta_i)$

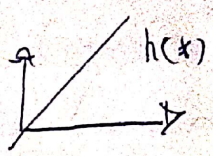
Goal: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Simplified:

$h_\theta(x) = \theta_1 x$

$\theta_0 = 0$

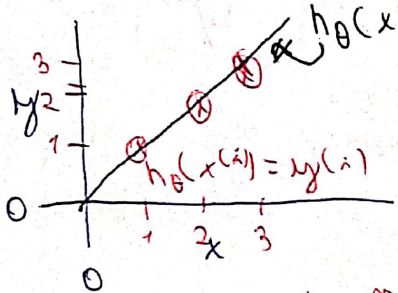
θ_1



$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

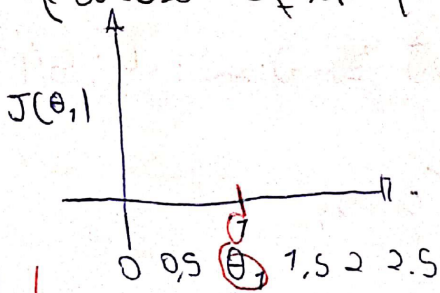
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



$\theta_1 = 0.5$?

Exercise:

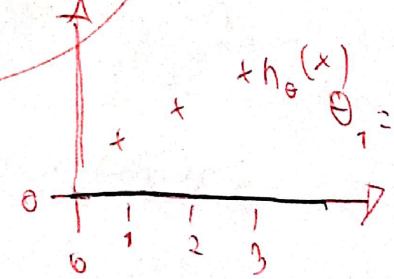
$J(0)$? $m = 3$

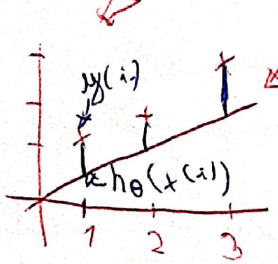
$(0-1)^2 + (0-2)^2 + (0-3)^2 = 14$

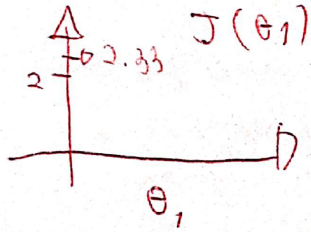
$\frac{1}{2 \cdot 3} = \frac{1}{6} \cdot 14 = \frac{14}{6}$

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_1 x^{(i)} - y^{(i)}\right)^2 = \frac{1}{2m}\left(0^2 + 0^2 + 0^2\right) = 0^2$$

$J(1) = 0$







$$J(0.5) = \frac{1}{2m}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$$

$$= \frac{1}{2 \times 3}(3.5) = \frac{3.5}{6} \approx 0.58$$

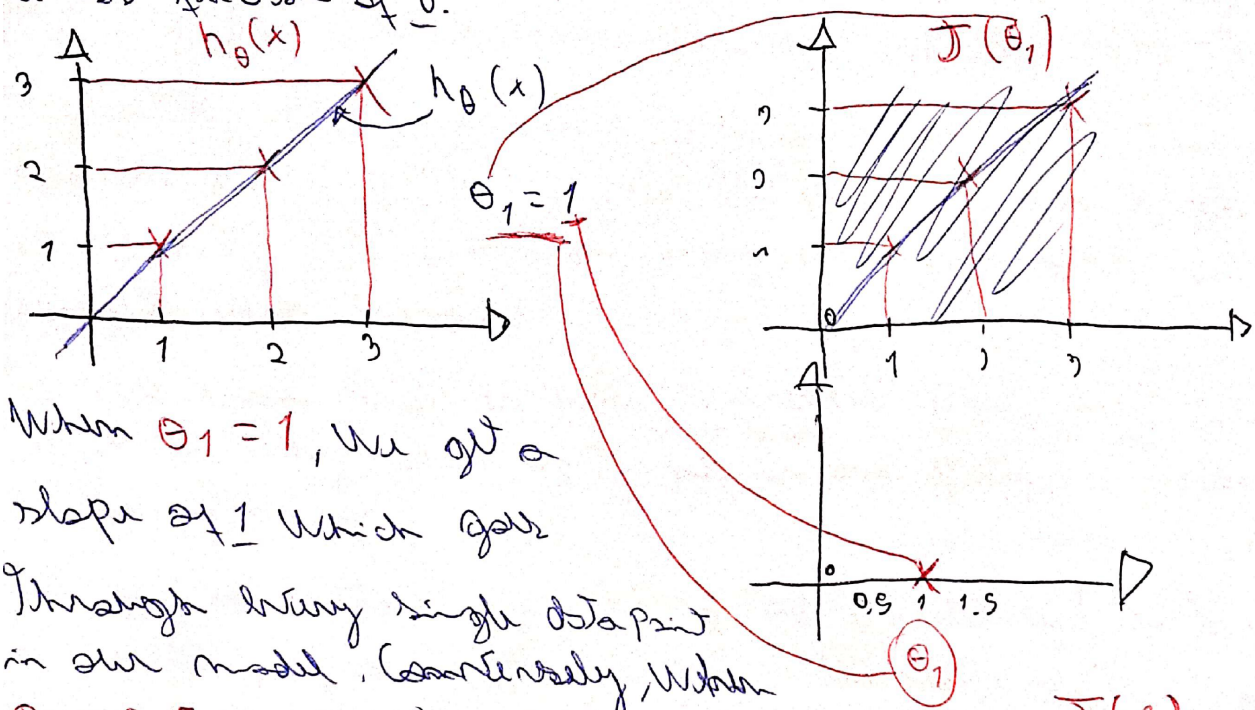
!! Vuelve Cost fun... I.1

9:40

- (Cost) function - Intuition I

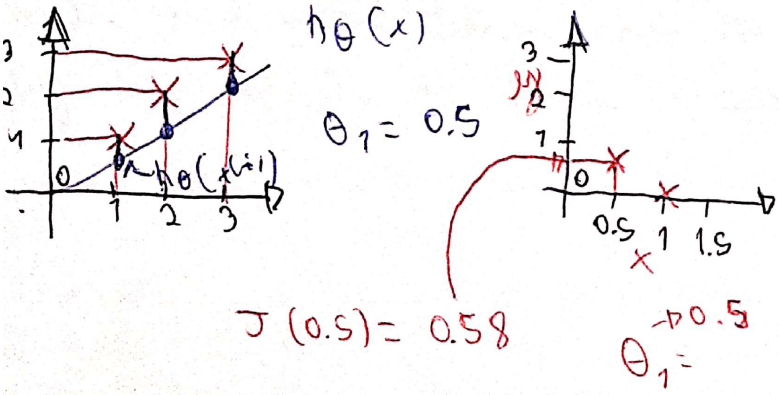
If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make a straight line (defined by $h_\theta(x)$) which passes through these scattered points. Our objective is to get the best possible line. The best possible line will be such that the average 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 $J(\theta_0, \theta_1)$ will be 0. The following example shows the ideal situation where we've a cost function of 0.



When $\theta_1 = 1$, we get a slope of 1 which goes through every single data point in our model. Conversely, when $\theta_1 = 0.5$, we see the vertical distance from our fit to the data points increase.

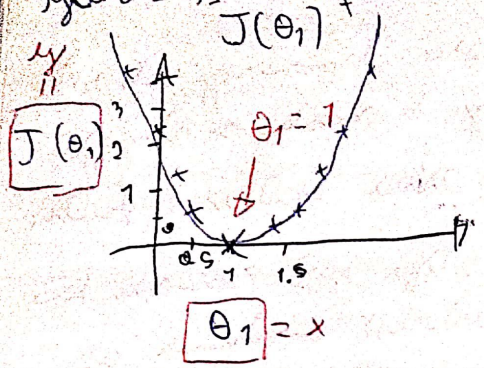
$J(1) = 0$



$J(0.5) = 0.58$

$\theta_1 \to 0.5$

This invokes our cost function ... yields 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.

- Cost function - Intuition II:

ML: 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 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 estimator). This can be kind of confusing; we are moving up to a higher level of abstraction. We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting 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 θ parameters.

We will know that we have succeeded when our cost function is at the very bottom of the pit in our graph, i.e. when its value is the minimum.

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, and the size of each step is determined by the parameter α, which is called our learning rate.

The gradient descent algorithm is:

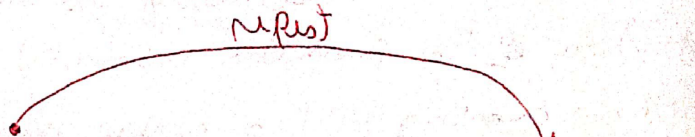
Repeat until convergence:

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

Where

$j = 0, 1$ represents the feature index number.

Intuitively, this could be thought of as:



Repeat until convergence:

$\theta_j := \theta_j - \alpha$ [Slope of tangent aka derivative in j dimension] [ ]

- Gradient Descent for Linear Regression:

When specifically applied to the case of Linear Regression, a new form of gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to (the derivation of the formulas are out of the scope of this course, but a really great one can be found here):

Repeat until convergence: {

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

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

}

When m is the size of the training set, θ_0 a constant that will be changing simultaneously with θ_1 and x_i, y_i are values of the given training set (data).

Note that we have separated out the two cases for θ_j into separate equations for θ_0 and θ_1; and that for θ_1 we are multiplying x_i at the end due to the ~~the~~ derivative.

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

Gradient Descent:

Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some θ_0, θ_1 (say $\theta_0 = 0, \theta_1 = 0$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum.

Assignment

$\theta_j := 0$ $\quad \theta_j := \alpha$ $\quad \alpha := b$ $\quad \alpha := \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

$\alpha := b$ (set $\alpha = to\ b$) $\quad a := a + 1$

derivate term

Truth assertion

$a = b$ ✗

$a = a + 1$ ✗

Learning Rate

Simultaneously update θ_0 and θ_1 !

Test:

$\theta_0 = 1$

$\theta_1 = 2$

$\theta_j := \theta_j + \sqrt{\theta_0 \theta_1}$ (for $j = 0$ and $j = 1$)

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

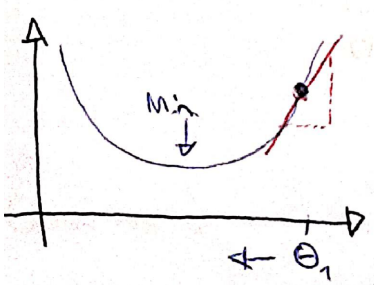
$temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

~~Temp~~ $\theta_0 := temp0$

$\theta_1 := temp1$

Gradient Descent ~~function~~:

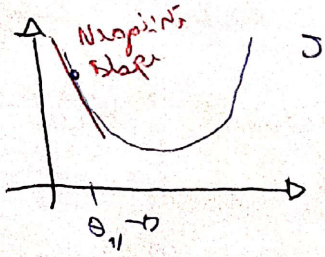
$\min_{\theta_1} J(\theta_1) \quad \theta_1 \in \mathbb{R}$



$J(\theta_1)$ $(\theta_1 \in \mathbb{R})$

$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$ (2 ✗)

$\theta_1 := \theta_1 - \alpha \cdot$ (positive number)



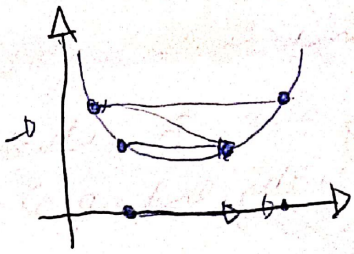
$\frac{\partial}{\partial \theta_1} J(\theta_1)$

$\theta_1 := \theta_1 - \alpha$ (Negative Number)

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

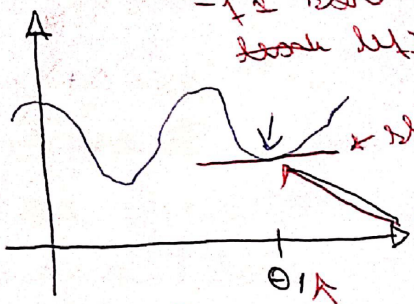
If α is too small, gradient descent can be slow.

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



#

If I am already at the local minimum my θ_1 will be left unchanged.



current value of θ_1

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

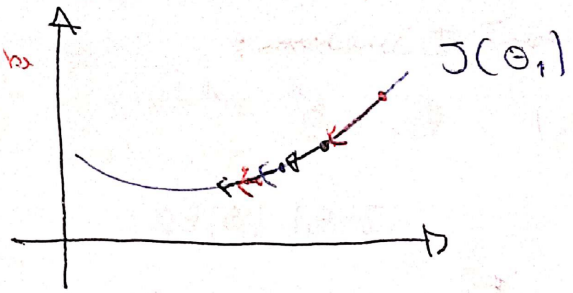
$= 0$

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

$\theta_1 := \theta_1$

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

$\theta_1 := \theta_1 - \alpha \boxed{\frac{d}{d\theta_1} J(\theta_1)}$ derivative term will be close to 0



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

Gradient Descent for linear Regression:

Gradient descent algorithm

Repeat until convergence {

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

(for $j = 1$ and $j = 0$)

}

Linear Regression model

$h_\theta(x) = \theta_0 + \theta_1 x$

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

$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

$$\frac{\partial}{\partial \theta_j} (\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)^2$$

$$\theta_0 \; j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 \; j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

Gradient descent algorithm

Repeat until convergence {

$\theta_0 := \theta_0 - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \right]$ ← $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\theta_1 := \theta_1 - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \right]$ ← $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

}

Update θ_0 and θ_1 simultaneously

//

Convex function"

Bowl-shaped

=

"Batch" Gradient Descent

"Batch" Each step of gradient descent uses all the training examples.

$\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$

Solution:

Then = he52
Thom = do gw

1. Gradient Descent can converge when if α is kept fixed. (But α cannot be too large, or else it may fail to converge.)

2. For the specific choice of cost function $J(\theta_0, \theta_1)$ used in linear Regression there is no local optima (other than the global optimum)

[ML: Linear Algebra Review]

- Matrices and Vectors

Matrices are 2-dimensional arrays:

[a b c d
e f g h
i j k l]

The above matrix has four rows and three columns, so it is a 4x3 matrix.

A vector is a matrix with one column and many rows:

[w
x
y
z]

So vectors are a subset of matrices. The above vector is a 4x1 matrix

*Notation and Terms:

- A_{ij} refers to the element in the i^{th} row and j^{th} column of matrix A.
- A vector with 'n' rows is referred to as an 'n'-dimensional vector.
- v_i refers to the element in the i^{th} row of the vector.
- In general, all our vectors and matrices will be 1-indexed. Note that for many programming languages. The arrays are 0-indexed
- Matrices are usually denoted by uppercase names while vectors are lowercase.
- "Scalar" means that an object is a single value, not a vector or matrix.
- $\mathbb{R}$ refers to the set of scalar real numbers
- $\mathbb{R}^n$ refers to the set of n-dimensional vectors of real numbers.

- Addition and Scalar Multiplication

Addition and subtraction are element-wise, so you simply add or subtract each corresponding element:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} + \begin{bmatrix} u & x & y \\ z & v & w \end{bmatrix} = \begin{bmatrix} a+u & b+x & c+y \\ d+z & e+v & f+w \end{bmatrix}$$

To add or subtract two matrices, their dimensions must be the same.

In scalar multiplication, we simply multiply every element by the scalar value:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} * x = \begin{bmatrix} a*x & b*x & c*x \\ d*x & e*x & f*x \end{bmatrix}$$

- Matrix - Vector Multiplication

We map the columns of the vector onto each row of the matrix, multiplying each element and summing the result.

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a*x + b*y + c*z \\ d*x + e*y + f*z \end{bmatrix}$$

The result is a vector. The vector must be the second term of the multiplication. The number of columns of the matrix must equal the number of rows of the vector.

An $m \times n$ matrix multiplied by an $n \times 1$ vector results in an $m \times 1$ vector.

Matrix-Matrix Multiplication

We multiply two matrices by breaking it into several vector multiplications and concatenating the result

$$[a\ b\ c\ d\ e\ f] * [u\ x\ y\ z] = [a*u + b*y\ \ a*x + b*z\ \ c*u + d*y\ \ c*x + d*z\ \ e*u + f*y\ \ e*x + f*z]$$

An $m \times n$ matrix multiplied by an $n \times o$ matrix results in an $m \times o$ matrix. In the above example, a 3×2 matrix times a 2×2 matrix resulted in a 3×2 matrix.

To multiply two matrices, the number of columns of the first matrix equal the number of rows of the second matrix

- Matrix Multiplication Properties
  - Not commutative. $A * B \neq B * A$
  - Associative. $(A * B) * C = A * (B * C)$

The identity matrix, when multiplied by any matrix of the same dimensions, results in the original matrix. It's just like multiplying numbers by 1. The identity matrix simply has 1's on the diagonal (upper left to lower right diagonal) and 0's elsewhere

$$[1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1]$$

When multiplying the identity matrix after some matrix ($A * I$), the square identity matrix should match the other matrix's columns. When multiplying the identity matrix before some other matrix ($I * A$), the square identity matrix should match the other matrix's rows.

- Inverse and Transpose

The inverse of a matrix A is denoted A^{-1}. Multiplying by the inverse results in the identity matrix

A non square matrix does not have an inverse matrix. We can compute inverses of matrices in octave with the pinv(A) function [1] and in matlab with the inv(A) function. Matrices that don't have an inverse are singular or degenerate.

The Transposition of a matrix is like rotating the matrix 90° in clockwise direction and then reversing it. We can compute transposition of matrices in matlab with the transpose(A) function or A'.

$$A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}$$

$$A^T = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

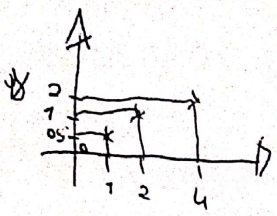
In other words:

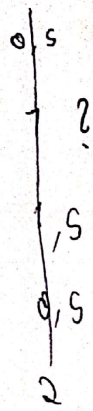
$$A_{ij} = A^T_{ji}$$

[Test: Linear Regression With one Variable]

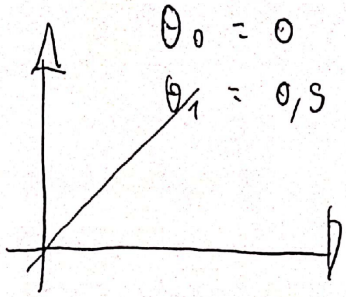
4x

$h_\theta(6) = -1 + 2.6 = 11$





0,5
0,5
2
1
1,5



$\frac{1}{2m}$ [

3) $\theta_0 = -2$, $\theta_1 = 0.5$

2) $J(J) = \frac{1}{2m}\left[(1-4)^2 + (2-1)^2 + (3-3)^2 + (4-1)^2\right]$

9 1 0 9

House sizes:

- 2104
- 1416
- 1534
- 852

Have 3 competing hypotheses:

1. $h_\theta(x) = -40 + 0.25x$
2. $h_\theta(x) = 200 + 0.1x$
3. $h_\theta(x) = -150 + 0.4x$

Matrix

$$\begin{bmatrix} 1 & 2104 \\ 1 & \vdots \\ 1 & \vdots \\ 1 & \vdots \end{bmatrix} \times \underbrace{\begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix}}_{\text{Matrix}} = \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 113 & 285 & 191 \end{bmatrix}$$

Prediction of 1° h_θ | Prediction of 2° h_θ

Matrix Transpose

Ex:

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$$

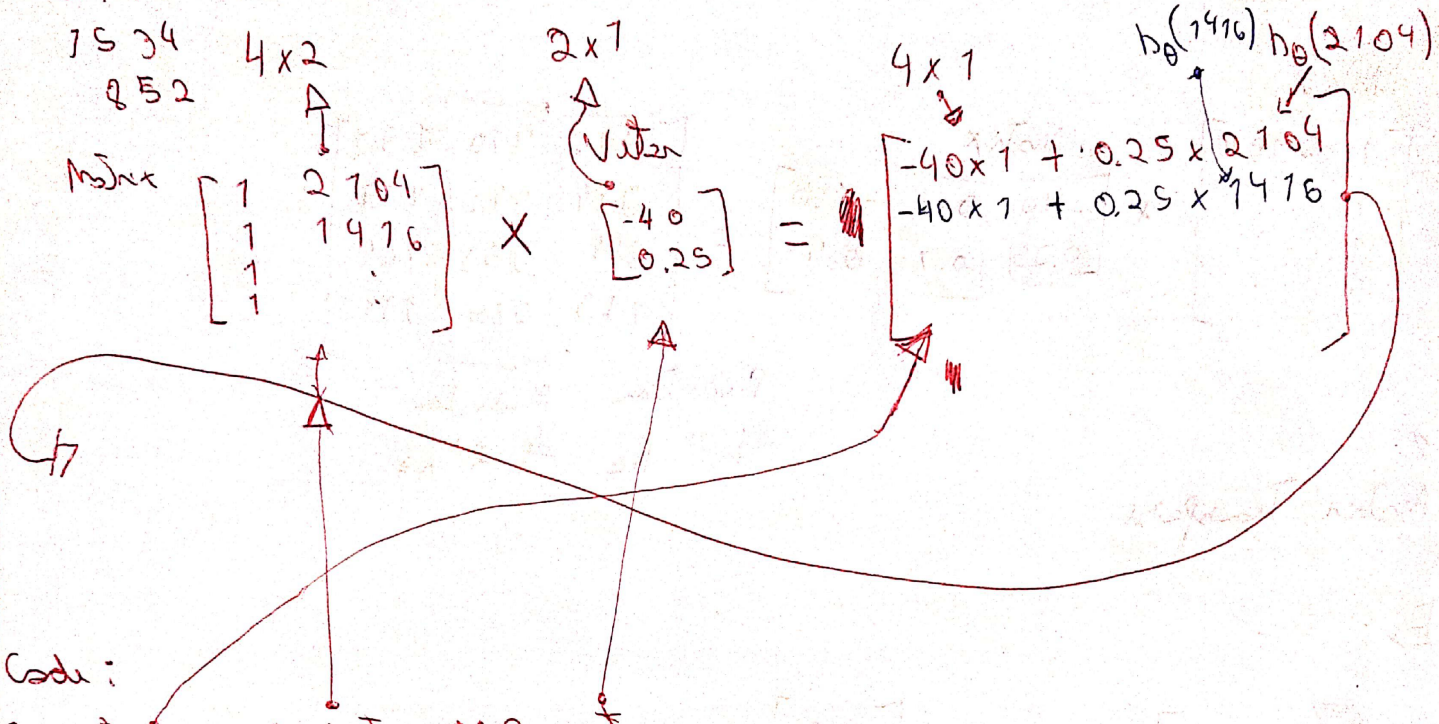
Notes

- Matrix Vector Mult. 1:44:

House Sizes:

2104
1416
1534
852

$h_\theta(x) = -40 + 0.25x$



Code:

```
Prediction = DataMatrix * Parameters
```

=

Mult. a matrix by a vector of 1 it's equal to sum all the numbers of the row in the matrix.