ML

Andrew Ng

Coursera

Week 1

Machine Learning Definition

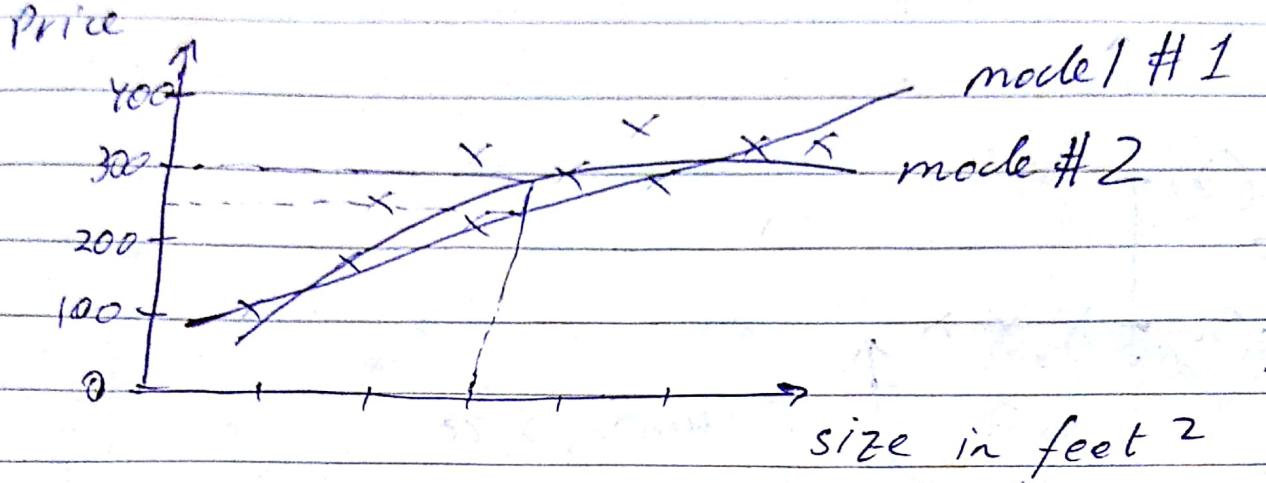
- ~~Arthur~~ S old version: field of study that gives computers the ability to learn without being explicitly programmed
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

new version

Machine Learning algorithms (main)
- Supervised learning
- unsupervised learning

Others types of machine learning algos: Reinforcement learning, recommender systems

Supervised learning ~~example~~ algo example:



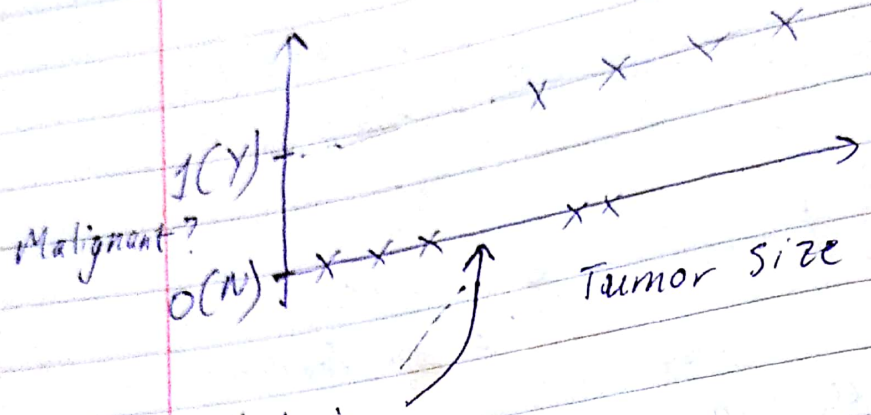
^ Housing price in a particular state
& 2 learning algorithms are applied

Supervised learning - where the "right answers" to a certain task are present in the data set & algo (using the provided data) tries to approximate the price ~~for~~ for a certain house

Also called a regression problem - predict continuous valued output (price in this example)

dangerous / harmful — harmless

Breast cancer example (malignant, benign)



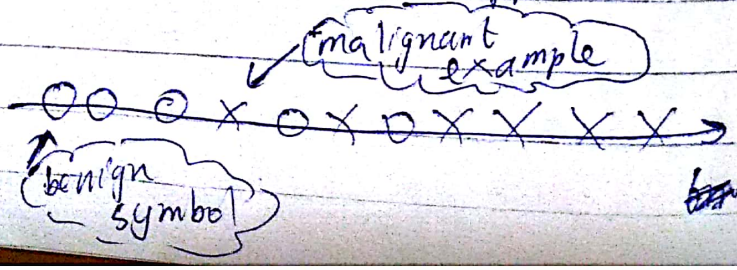
what is the probability of whether the cancer is malignant or benign based on this particular tumor size

^Classification problem example - discrete valued outputs (~~more~~ can be more than 2 values)

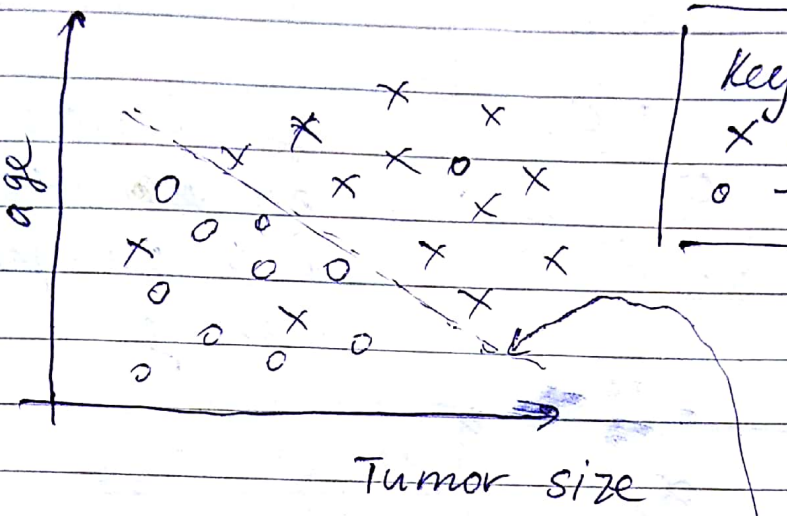
↳ suppose there are 3 types of cancers : 0, 1, 2, 3

0 ← benign
1 ← type 1 cancer
2 ← type 2 cancer
3 ← type 3 cancer

Another visualisation approach:



Example of using more than 1 feature to decide whether tumor is malignant (M or benign (B)

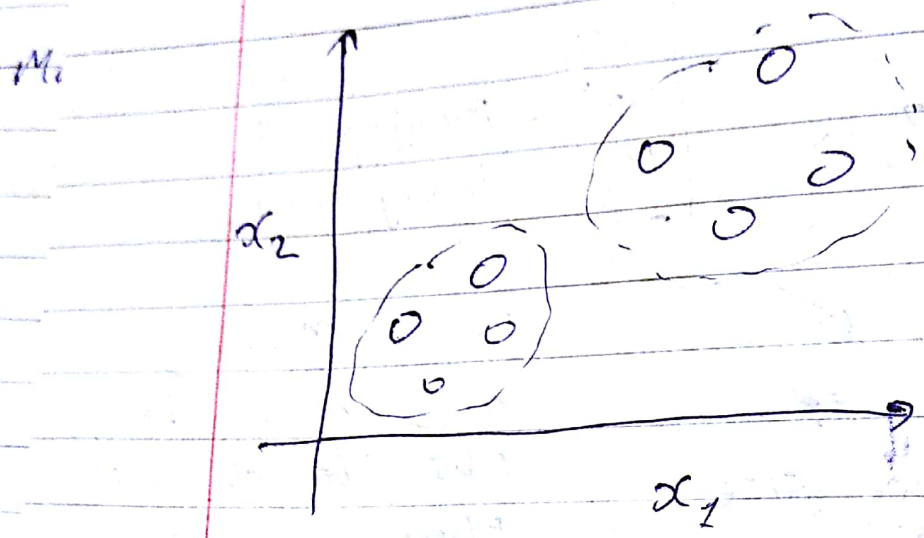


The learning algo must be able to decide ~~there~~ where the boundary lies between M & B

Based on certain values of age & tumor size we can find the probability of the tumor being malignant or benign

Most machine learning problems have many more features. Support vector machines allow you to deal with infinite features with a mathematical trick

Unsupervised learning - data does not contain labels to be predicted (unlike supervised learning). Main objective is to find structure in the dataset



2 clusters are identified with unsupervised learning

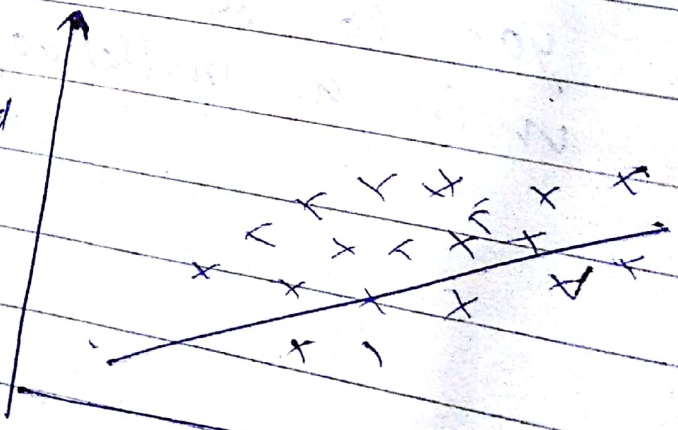
^Clustering problem

Linear regression

A model for modeling the relationship between a numerical variable y & one or more explanatory variables (or features) as a linear function.

regression problems usually

example of supervised learning



Important notation:

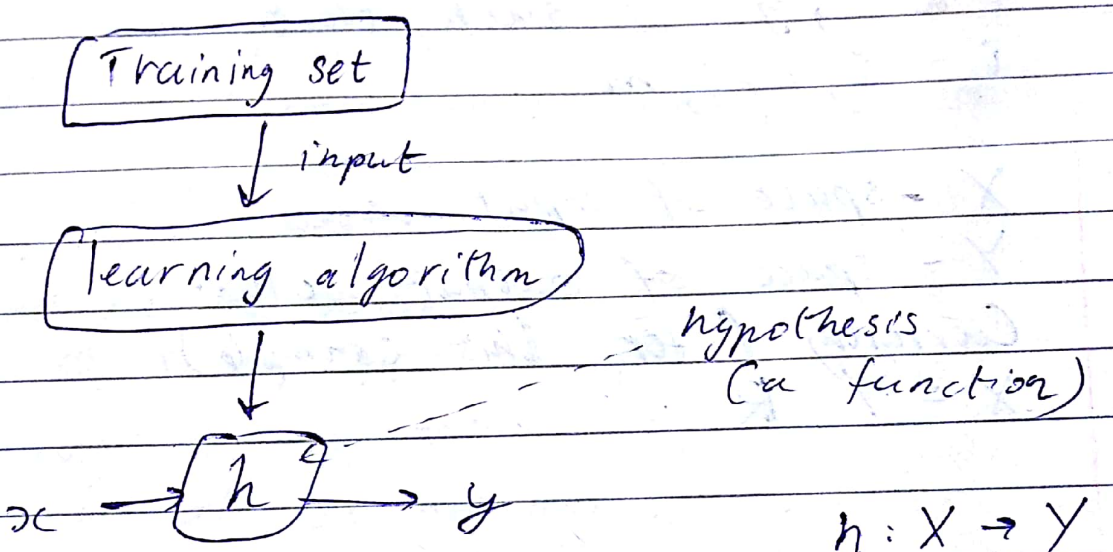
m - number of training examples

x - input variable / feature

y - output variable / label or target variable

(x, y) - single training example (a single row in dataset)

$(x^{(i)}, y^{(i)})$ - ith training example (ith row in dataset)

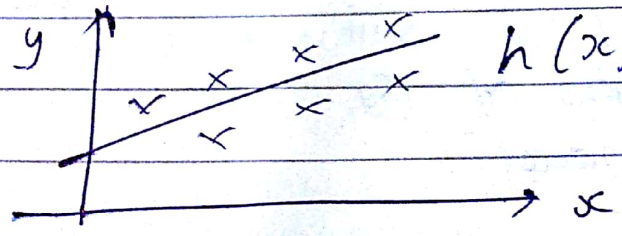


$h : X \to Y$

h maps from x's to y's

Representing h: (only for the current time being)

$h_\theta(x) = \theta_0 + \theta_1 x$ } linear function



$h(x) = \theta_0 + \theta_1 x$

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

↳ linear regression with one variable or univariate linear regression (x is the variable)

The entire data set:
A list with m training examples
$(x^{(i)}, y^{(i)})$ such that
$i = 1, 2 \ldots, m$

X - space of input values
Y - space of output values
Currently ($X = Y = \mathbb{R}$) for this example

Cost function

Will help figure out how to fit the best possible line for data

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

How do we choose these parameters? → parameters of the model

predicted value

Main idea of a cost function:
Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

Formally:

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

makes math & calculations easier

means:
find me the values of θ_0 & θ_1 that causes this expression to be at its minimum

To add some more notation:

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

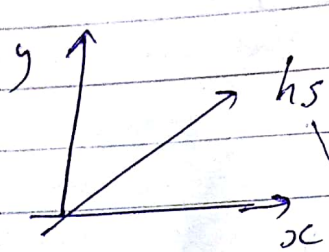
cost function / squared error func

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

- The accuracy of a hypothesis function is measured with a cost function
- The cost function is an average difference thus it is divided by m
- As a convenience for the computation of gradient descent the cost function is multiplied with $\frac{1}{2}$

Assume simplified $h_\theta(x)$ for now:

$h_{\theta s}(x) = \theta_1 x \quad (\theta_0 = 0)$



$h_s(x)$ will always go through the origin

and goal is: $\min_{\theta_1} J(\theta_1)$

s for simplified

$h_\theta(x)$

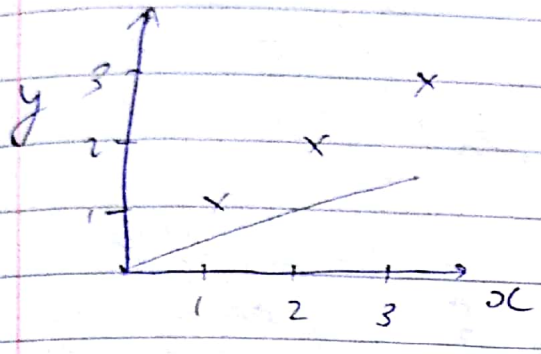
for fixed θ_1, this is a function of x

$J(\theta_1)$

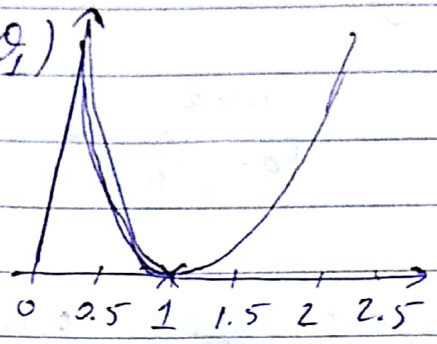
function of parameter θ_1 & controls the slope of line

The question rests: why is $J(\theta)$ an average rather than an overall aggregate of cost? Reason is such that if 2 models are to be compared we don't want the # of points to increase the overall cost hence average is used (compare model fit across datasets)

$h_\theta(x)$



$J(\theta_1)$



the training set & the test set can be divided into different proportion hence there must be a metric that must measure cost independent of data points

$\theta_1 = 0.5$

$$J(0.5) = \frac{1}{2m}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right] = \frac{1}{6}(3.5) \approx 0.58$$

The minimum of $J(\theta_1)$ occurs at $\theta_1 = 1$

↙ used in other places as well

Gradient descent

Will help us minimise the value of $J(\theta_1, \theta_2)$ by changing values of θ_1, θ_2

Outline of procedure:

- start with some θ_1, θ_0
- keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at ~~in~~ minimum

$a = b$ → true or false

$a := b$ → assign value of b to a

Gradient descent is actually an algo that can be applied on any such problem:

$$\min_{\theta_0 \dots \theta_n} J(\theta_0, \dots, \theta_n)$$

repeat until convergence {

learning rate (controls how big the steps are)

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

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

}

θ_0 & θ_1 must be updated simultaneously. Serial Implementation:

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

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

$\theta_0 := temp0$

$\theta_1 := temp1$

Incorrect simultaneous implementation :

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

~~$temp = \theta_1$~~

$\theta_0 := temp0$

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

$\theta_1 := temp1$

If updates are not simultaneous then the algorithm won't be called gradient descent & it may behave in strange ~~ways~~ unexpected ways since there is a change in the algorithm

The issue with gradient descent is that there might ~~be~~ be many various local minima & the question of which local minima you will arrive to depends on the initial values of θ_0 & θ_1

repeat until convergence {

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

(for θ_0 and θ_1)

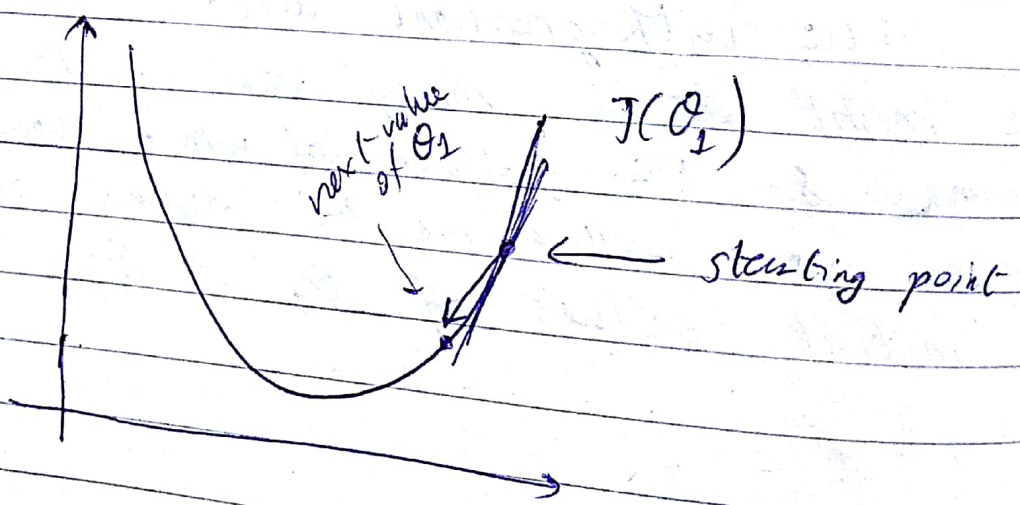
}

α — learning rate

(determines how big the update is on each iteration)

Suppose simpler cost function:

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

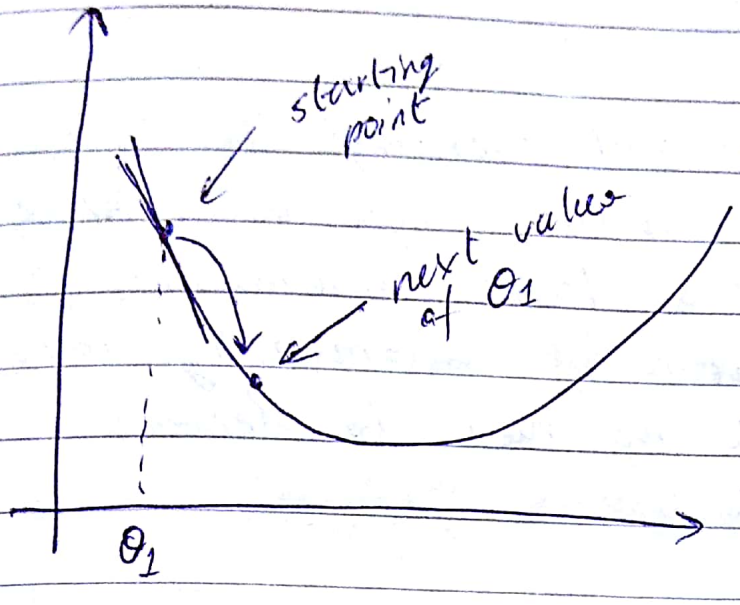


$$\theta_1 := \theta_1 - \alpha \underbrace{\frac{d}{d\theta_1} J(\theta_1)}_{\text{calculates slope at a particular value}}$$

$\frac{d}{d\theta_1} J(\theta_1) > 0$ at starting point here

$\theta_1 := \theta_1 - (+ve)$, thus θ_1 decreases in value

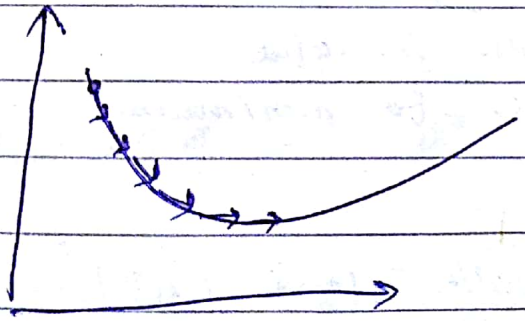
Alternatively:



$\frac{d}{d\theta_1} J(\theta_1) < 0$ at the starting point

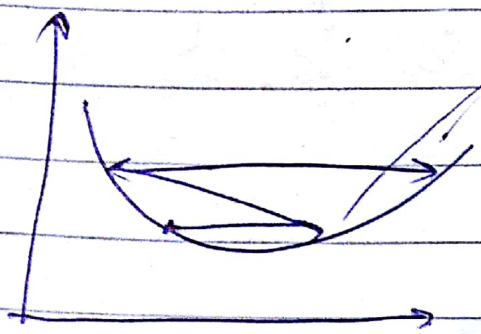
$\theta_1 := \theta_1 - \alpha\,(-ve)$ thus θ_1 increases in value

If α is too small, gradient descent can be slow



if one step is taken & it overshoots then the next step will overshoot more as the gradient is higher in value

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



If the initial value of $J(\theta_1)$ reaches a local minimum then the gradient descent algo will stop as at the minimum $\frac{d}{d\theta_1} J(\theta_1) = 0$ hence no more change in value

Gradient descent will converge to local minimum with fixed value of α

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

↳ will take smaller steps due to decreasing slope hence decrease or increase will be smaller in value

$$\theta_1 := \theta_1 - \alpha \underbrace{\frac{d}{d\theta_1} J(\theta_1)}_{\text{smaller in value as closer to minimum}}$$

Algo keeps taking smaller steps until converges to minimum

Combining gradient descent & linear regression model

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

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

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

Linear regression

Gradient descent algo

repeat until convergence {

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

for $j = 1$ & $j = 0$

}

Finding out value of slope:

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

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

For $j=0$: $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) =$

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

For $j=1$: $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) =$

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

Gradient descent for linear regress

repeat until convergence {

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

}

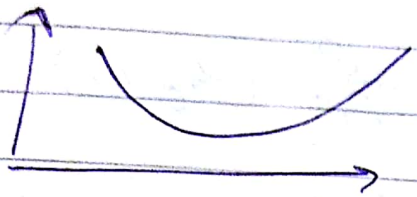
update θ_0 & θ_1 simultaneously

Issue with gradient descent:

- susceptible to local optima
  - ↳ different minimum value depending on how θ_0 & θ_1 are initialised

simply $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

The cost function for linear regression always is a bow shaped function (a convex function)



} convex - always has 1 single global optimum

Hence gradient descent will always converge to the global optimum for linear regression

Gradient descent algo also called "batch" gradient descent

↳ batch because each step gradient descent uses all the training examples

There are other different versions of existing gradient descent

Finding out the value of the slope (general method):

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 =$$

$$= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) =$$

$$= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right)$$

$$= (h_\theta(x) - y)(x_j)$$

Linear Algebra

Matrix: rectangular array of numbers

Dimension of matrix: row x columns

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

2×3 matrix

or

$\mathbb{R}^{2 \times 3}$ matrix

Since A is a matrix, then:

A_{ij} = "i, j entry" in the ith row & jth column

so:

$A_{11} = 1$, $A_{23} = 6$

Vector - special type of matrix - an $n \times 1$ matrix

↑ only 1 column

example:

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 175 \end{bmatrix}$$ ← 4-dimensional vector or $\mathbb{R}^4$ vector

y_i = ith element so $y_1 = 460$, $y_2 = 232$

Upper case letters are used to refer to matrices
Lower case letters are used to refer to scalars & vectors

Matrix Addition

you can only add matrices of the same dimension

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0,5 \\ 2 & 5 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0,5 \\ 4 & 10 \\ 3 & 2 \end{bmatrix}$$

3×2 3×2 3×2

Scalar Multiplication

$$3 \times \underset{3\times2}{\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix}} = \underset{3\times2}{\begin{bmatrix} 3 & 0 \\ 6 & 15 \\ 9 & 3 \end{bmatrix}} =$$

$$= \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} \times 3$$

$$\underset{2\times2}{\begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix}} / 4 = \frac{1}{4} \underset{2\times2}{\begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix}} = \begin{bmatrix} 1 & \\ 3/2 & \end{bmatrix}_{2\times}$$

The dimensions will remain the same

Vector - Matrix Multiplication

$$\underset{3\times2}{\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix}} \underset{2\times1}{\begin{bmatrix} 1 \\ 5 \end{bmatrix}} = \underset{3\times1}{\begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}}$$

$1 \times 1 + 3 \times 5 = 16$

$4 \times 1 + 0 \times 5 = 4$

$2 \times 1 + 1 \times 5 = 7$

Suppose the following house size data:

2104
1416
1534
852

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

We need to evaluate a prediction on each of sizes. This can be done through matrices & vectors:

$$\underset{4\times 2}{\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix}} * \underset{2\times 1}{\begin{bmatrix} -40 \\ 0.25 \end{bmatrix}} =$$

$$= \begin{bmatrix} -40\times 1 + 0.25\times 2104 \\ -40\times 1 + 0.25\times 1416 \\ -40\times 1 + 0.25\times 1534 \\ -40\times 1 + 0.25\times 852 \end{bmatrix}$$

$h_\theta(2104)$ ← $h_\theta(1416)$ ← ⋮

4×1 matrix

1st way In code the above operation can be written in a single line as such:

prediction = Data matrix * parameters

(2nd way) Compare this to using a loop:

```
for i = 1:4
    prediction(i)
```

By using the 2nd way the code is simplified by a great degree & it is computationally efficient as well

Matrix - Matrix multiplication

$$\underset{2\times 3}{\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix}} \times \underset{3\times 2}{\begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix}} = \underset{2\times 2}{\begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} = \begin{bmatrix} 11 \\ 9 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

Hence:

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}$$

Application:

Suppose we have the following houses & we wish to compute its predictions:

2104, 1416, 1534, 852

3 competing hypothesis are available:

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

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix} \times \begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix} =$$

$$* \\ * \\ = \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 173 & 285 & 191 \end{bmatrix}$$

An efficient manner of applying 3 hypothesis on 4 data points & can be written with 1 line of code as well

3rd hypothesis predictions

each column is a list of predictions from one hypothesis

2nd hypothesis predictions

First hypothesis predictions

Matrix Multiplication properties

$A \times B \neq B \times A$
if A & B are matrices i.e. matrix multiplication is not commutative

Matrix multiplication is associative:
$A \times B \times C = (A \times B) \times C = A \times (B \times C)$

Identity matrix:

Denoted by I or $I_{n \times n}$.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}_{2 \times 2} \quad \text{or} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{3 \times 3}$$

For any matrix A:

$$A \times I = I \times A = A$$

$m \times n \quad n \times n \quad m \times m \quad m \times n \quad m \times n$

↑ ↑ different

because if A is $n \times m$ then:
$A A^{-1} = I_n$ & $A^{-1} A = I_m$
& $I_n \neq I_m$

Matrix inverse

must be a square matrix

If A is an $m \times m$ matrix and if it has an inverse,

$$\underset{(n\times m)}{A}\ \underset{(m\times n)}{A^{-1}} = \underset{(m\times n)}{A^{-1}}\ \underset{(n\times m)}{A} = I$$

E.g. $\begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix} \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_{2\times 2}$

A $\quad$ A^{-1} $\quad$ AA^{-1}

Some matrices do not have an inverse — singular matrix or degenerate matrix

e.g: $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

Matrix Transpose

convert rows into columns

$$A = \underset{2\times 3}{\begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}}, \quad A^T = \underset{3\times 2}{\begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}}$$

Let A be a $m \times n$ matrix, and let $B = A^T$
Then B is an $n \times m$ matrix and
$B_{ij} = A_{ji}$