Machine Learning (Coursera, Stanford), DeepLearning.AI

↳ Grew out of work in AI
↳ New capability for computers

ex. ⇒ 1) Database mining
2) Application can't program by hand
↳ Autonomous helicopter, handwriting recognition, most of natural language processing (NLP), ~~Computer~~ Computer vision
3) Self-customizing programs
↳ Amazon, Netflix product recommendation
4) Understanding human learning

⇒ What is machine learning?
↳ Field of study that gives computers the ability to learn without being explicitly programmed. (Arthur Samuel {1959})

↳ 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. (Tom Mitchell {1998})

Machine learning algorithms:-
↳ Supervised ↳ Unsupervised
others. = Reinforcement, Recommender

→ Supervised Learning

{* Classification Problem
↳ Discrete valued output (0, 1 or more)}

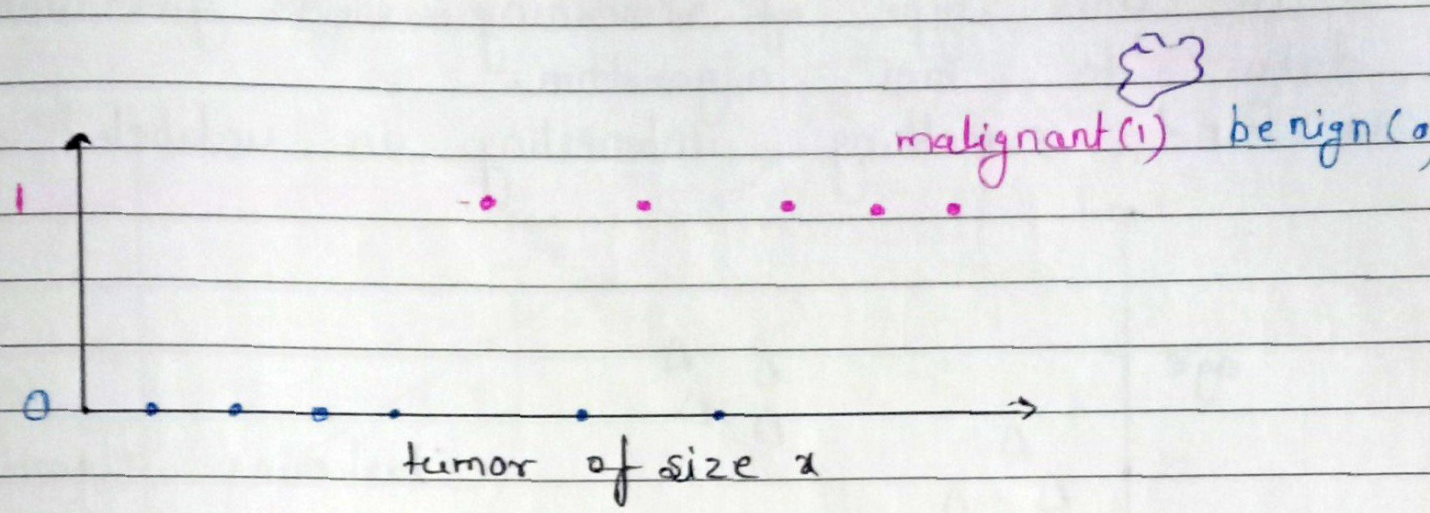
X	→	Y
input		output label

↳ Learn from being given "right answers".

input (x)		output (y)	Application
email	→	spam? (0/1)	spam filtering
audio	→	text transcripts	speech recognition
English	→	Spanish	machine translation
ad, user info	→	click (0/1)	online advertising
image, radar info	→	position of other cars	self driving car
image of phone	→	defect (0/1)	visual inspection

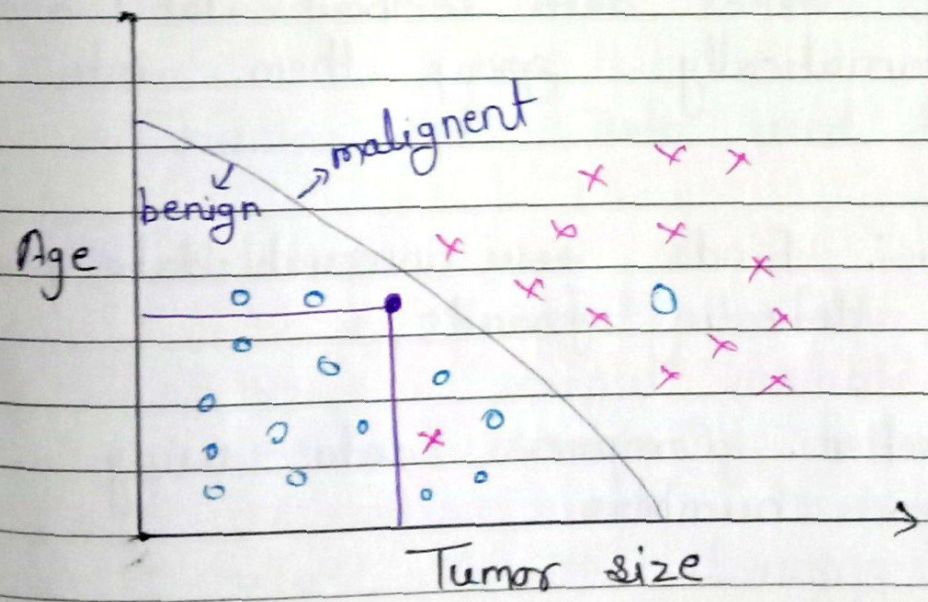
* Regression : Housing price prediction
↳ Predict a number from infinity many possible outputs.

★ Classification : Breast cancer detection



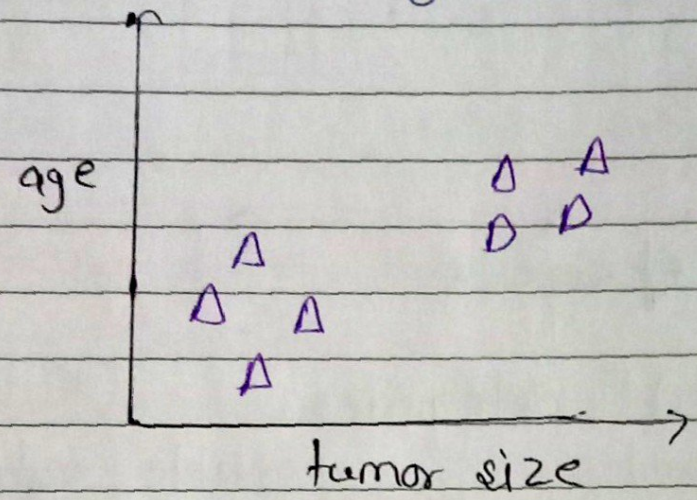
↳ This algorithm predict categories.
↳ predict small number of possible outputs

Two or more inputs :-



Unsupervised Learning

↳ In this type of learning, we give unlabled data to our algorithm.
↳ Find something interesting in unlabeled data.



Clustering algorithm
↳ Google news
↳ DNA microarray
↳ grouping customers

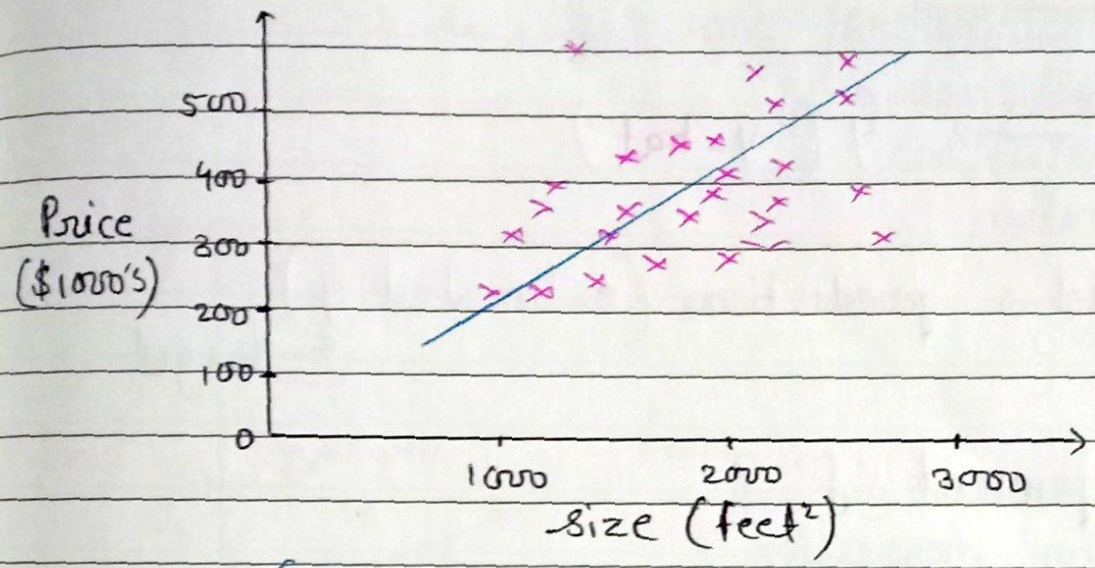
↳ Clustering algo. takes data without label and tries to automatically group them into clusters.

↳ Anomaly detection: Finds ~~use~~ unusual data points.
↳ ex.: detecting frauds

↳ Dimensionality reduction: Compress data using fewer numbers.

Regression Model :-

⇒ Linear Regression Model :-



{ House sizes and price }

★ Terminology →

↳ Training Set :- Data used to train the model

↳ Notation

↳ x = "input" variable, feature

↳ y = "output" variable, "target" variable

↳ m = number of training examples

↳ (x, y) = single training example

↳ $(x^{(i)}, y^{(i)})$ = i^{th} training example

$\boxed{\text{training set}}$ - {features, targets}

↓

$\boxed{\text{learning algorithm}}$

↓

$x \longrightarrow \boxed{f} \longrightarrow \hat{y}$ ("y-hat")

(hypothesis or function

feature → model → prediction (estimated y)

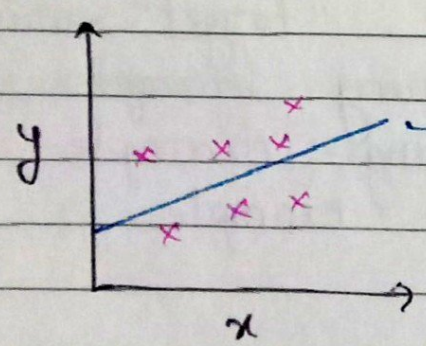
← target

⇒ How to represent 'f'?

↳ for linear regression

↳ $\boxed{f_{w,b}(x) = wx + b}$

↳ f is a function that takes 'x' as input and depending on the values of 'w' and 'b', 'f' will ~~ou~~ output some value of a prediction y-hat ($\hat{y}$).



$f_{w,b}(x) = wx + b$

or $f(x) = wx + b$

↳ linear

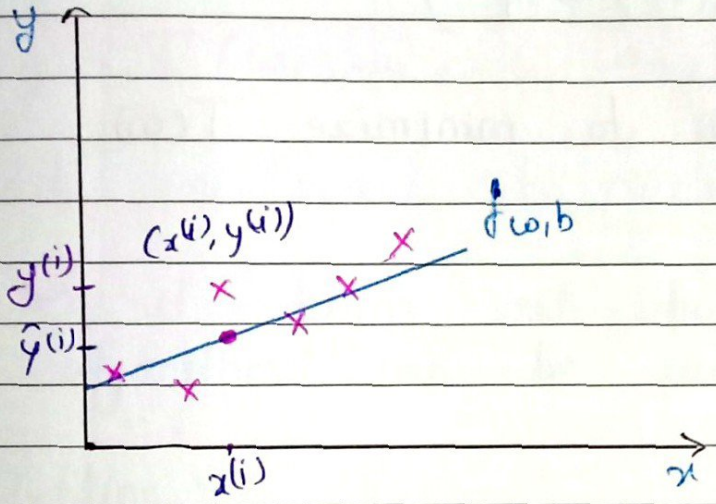
↳ it is linear regression with one variable.

↳ Univariate linear regression.

⇒ Cost Function Formula :-

Model : $f_{w,b}(x) = wx + b$

↳ w, b are parameters of model.
↳ also known as coefficients or weights



$$\hat{y}^{(i)} = f_{w,b}(x^{(i)})$$

$$f_{w,b}(x^{(i)}) = wx^{(i)} + b$$

Find w, b : (so $\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$)

★ <u>Cost function</u> :-

$$\underbrace{J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \Big(\underbrace{\hat{y}^{(i)} - y^{(i)}}_{\text{error}} \Big)^2}_{\text{Squared Error Cost Function}}$$

(m → number of training examples)

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)} \right)^2$$

↳ we find w and b, for which cost fun. is the smallest.

Date
Page

⇒ Cost Function Intution :-

Simplified model ($b=0$) :-

$$f_w(x) = wx \quad \{\text{only one parameter}, w\}$$

↳ $$J(w) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_w(x^{(i)}) - y^{(i)}\right)^2$$

↳ now the goal is to minimize $J(w)$.

Gradient Descent :-

↳ one of the most important in ML

↳ You can use gradient descent to try to minimize any function.

if you have a function.

$$J(w_1, w_2, \ldots, w_n, b)$$

and you want to minimize it.

↳ it turns out that gradient descent algorithm can be used to try to min. it.

Outline :-

↳ start with some w, b.

↳ keep changing w, b to reduce $J(w,b)$ until we settle at or near a minimum

some 'f' may have min. >1

⇒ Implementing Gradient Descent :-

assignment

- $w = w - \alpha \times \frac{\partial}{\partial w} J(w,b)$

  → Derivative → tells you direction

  ↳ learning rate

  ↳ if large → aggressive G.D. procedure
  ↳ huge steps downhill

  ↳ if small → small baby steps downhill

- $b = b - \alpha \frac{\partial}{\partial b} J(w,b)$

↳ Update both ω and b simultaneously

Correct: Simultaneous Update →

$$tmp_\omega = \omega - \alpha \frac{\partial}{\partial \omega} J(\omega, b)$$

$$tmp_b = b - \alpha \frac{\partial}{\partial b} J(\omega, b)$$

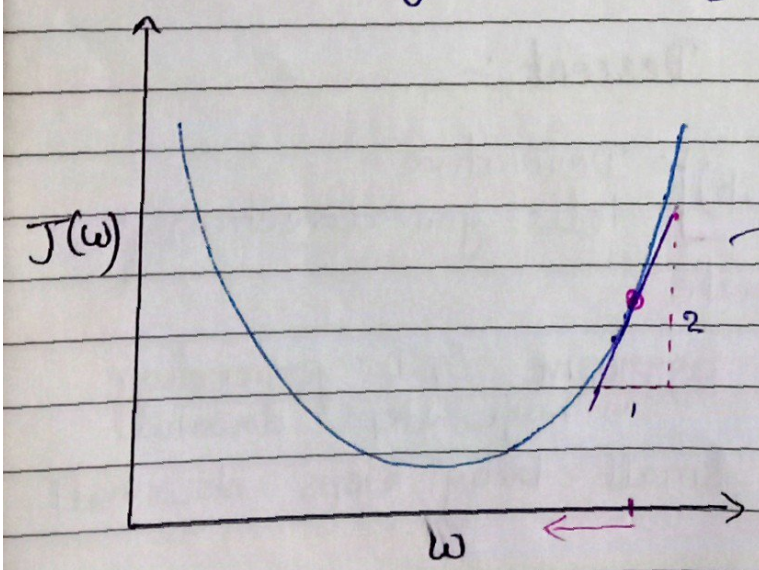
$$\omega = tmp_\omega$$
$$b = tmp_b$$

⇒ Gradient Descent Intution :-

↳ Let's say that you have a cost function $J(\omega)$.

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

and we are trying to minimize the cost by adjusting the parameter ω.



here this slope is +ve.
↳

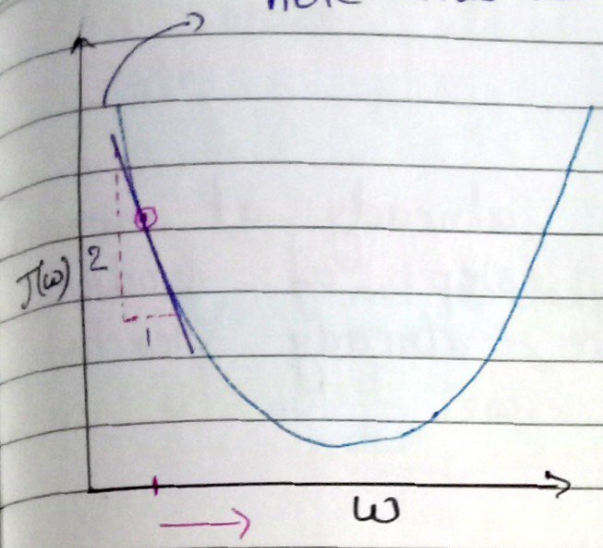
$$\omega = \omega - \alpha \underbrace{\left(\frac{\partial}{\partial \omega} J(\omega)\right)}_{>0}$$

$$\Rightarrow \omega = \omega - \alpha \text{ (+ve number)}$$

this new ω is smaller.

here this slope is -ve

↳ $\frac{d}{dw} J(w) < 0$



$w = w - \alpha$ (-ve number)

↳ this new w is greater

⇒ Learning Rate (α) :-

$$w = w - \alpha \frac{d}{dw} J(w)$$

↳ learning rate

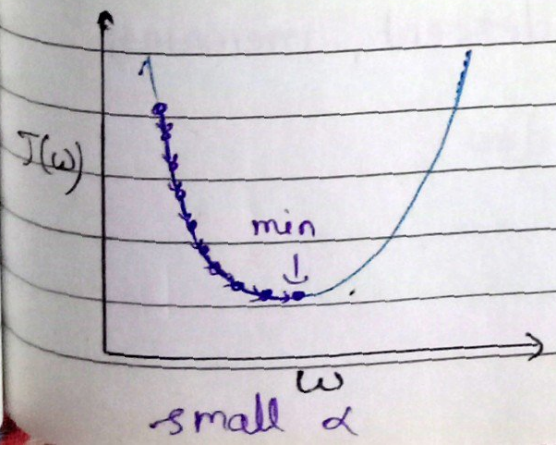
if α is too small ---

↳ ~~Gr~~ Gradient Descent may be slow. It will take very long time because it's going to take these tiny tiny baby steps. And it will need a lot of steps before reaching minimum.

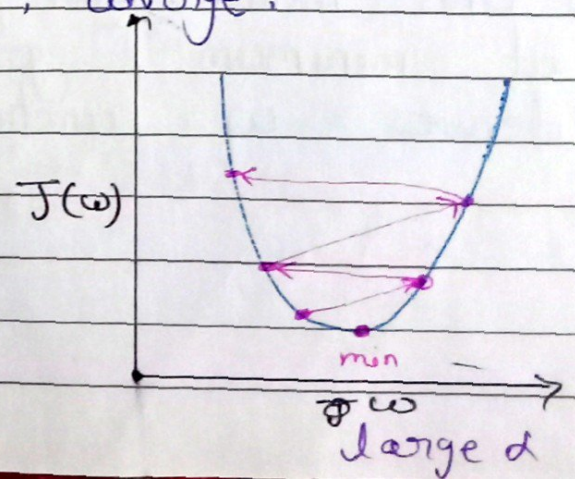
if α is too large ---

↳ Gradient Descent may overshoot, never reach minimum.

↳ Fail to converge, diverge.



small α

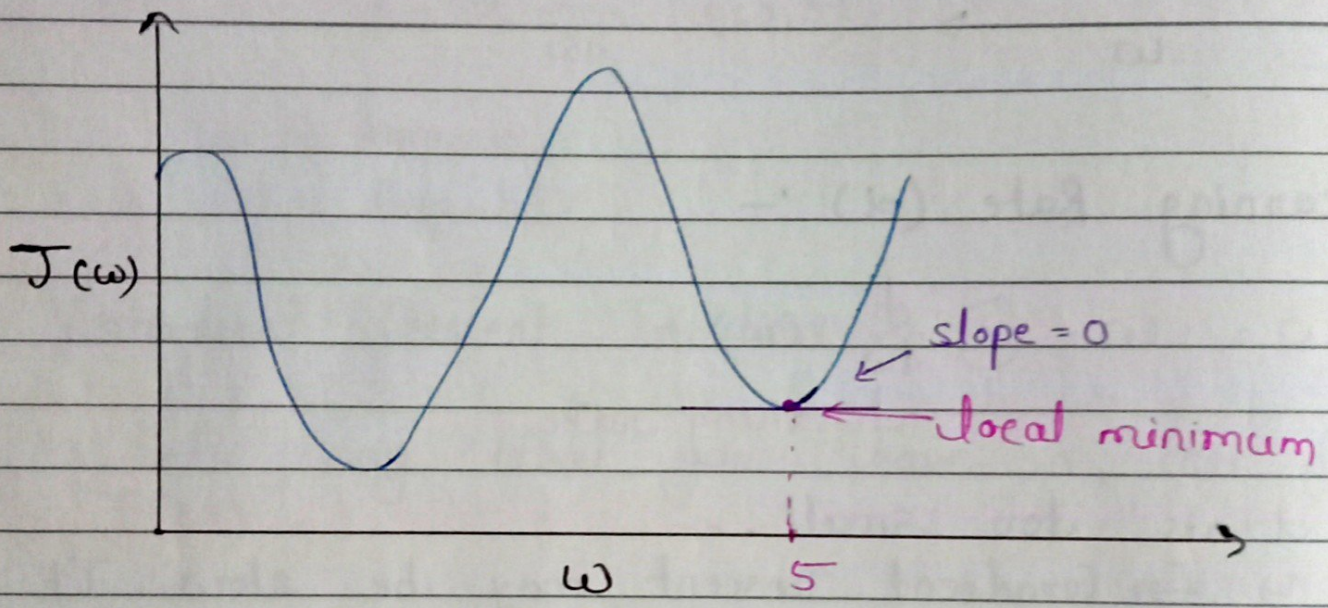


large α

Date
Page

Important Question !!!

What if your cost J is already at minimum What do you think one step of gradient descent will do if you've already reached a minimum?



$$\omega = \omega - \alpha \underbrace{\frac{d}{d\omega} J(\omega)}_{0 \text{ (at local minimum)}}$$

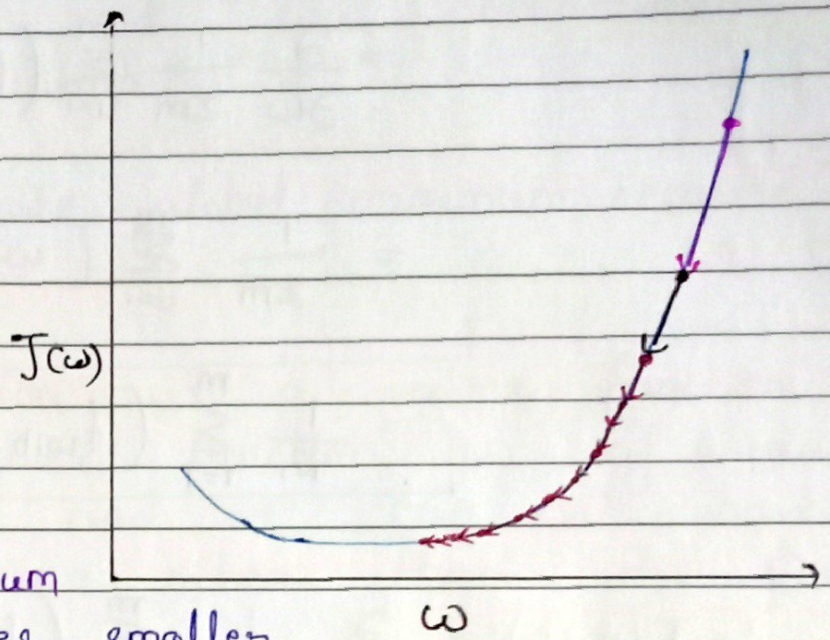
↳ $\omega = \omega - \alpha . 0$

$$\boxed{\omega = \omega}$$

↳ this means that if you are ~~alred~~ already at minimum, gradient descent ~~remains~~ leaves ω unchanged.

Can reach local minimum with fixed learning rate.

$$w = w - \underbrace{\alpha \underbrace{\overbrace{\frac{d}{dw} J(w)}^{\text{not as large}}}_{\text{large}}}_{\text{smaller}}$$



↳ Near a local minimum
  ↳ Derivative becomes smaller
  ↳ Update steps becomes smaller

⇒ Gradient Descent for Linear Regression :-

↳ Linear Regression Model ⇒ $f_{w,b}(x) = wx + b$

↳ Cost Function ⇒ $J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)} \right)^2$

↳ Gradient Descent Algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

}

- $$\frac{\partial}{\partial \omega} J(\omega, b) = \frac{\partial}{\partial \omega} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\omega,b}(x^{(i)}) - y^{(i)} \right)^2$$

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

$$= \frac{1}{\cancel{2}m} \sum_{i=1}^{m} \left(\omega x^{(i)} + b - y^{(i)} \right) \cancel{2} x^{(i)}$$

$$\boxed{= \frac{1}{m} \sum_{i=1}^{m} \left(f_{\omega,b}(x^{(i)}) - y^{(i)} \right) x^{(i)}}$$

- $$\frac{\partial}{\partial b} J(\omega, b) = \frac{\partial}{\partial b} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\omega,b}(x^{(i)}) - y^{(i)} \right)^2$$

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

$$= \frac{1}{\cancel{2}m} \sum_{i=1}^{m} \left(\omega x^{(i)} + b - y^{(i)} \right) \cancel{2}$$

$$\boxed{= \frac{1}{m} \sum_{i=1}^{m} \left(f_{\omega,b}(x^{(i)}) - y^{(i)} \right)}$$

↳ Put the values of these derivates in GD algorithm and update ω & b simultaneously.

↳ Repeat until convergence.

Date
Page

↳ When you are using squared error cost function with linear regression, the cost function does not and will never have multiple local minima.

↳ It has a single global minimum because of its bowl shape.

↳ This cost function is a convex function. Informally, convex function is of bowl-shaped function and it can not have any local minima other than the single global minimum.

"Batch" Gradient Descent :-

↳ each step of all gradient descent uses all the training examples.

↳ There are also other gradient descents which do not look at all training data but a small subset of it.

Multiple Linear Regression :-

⇒ Multiple Features :-

size in feet^2 x_1	no. of bedroom x_2	no. of floors x_3	Age of home(y) x_4	Price ($1000's)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

x_j = j^{th} feature (1, 2, 3, 4)

n = number of features (4)

$\vec{x}^{(i)}$ = feature of ith training example

↳ $\vec{x}^{(2)} = [1416 \quad 3 \quad 2 \quad 40]$

↳ row vector

$x_j^{(i)}$ = value of feature 'j' in 'ith' training example

Model :

previously : $f_{w,b}(x) = wx + b$

now : $f_{w,b}(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$

↳ $f_{w,b}(x) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$

$\vec{w} = [w_1 \quad w_2 \quad w_3 \quad \cdots \quad w_n]$ } Parameters of the model

b is a number

$\vec{x} = [x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_n]$

$$\boxed{f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b}$$

↳ dot product

Date
Page

⇒ Vectorization :-

$\vec{w} = [w_1 \quad w_2 \quad w_3]$ $\quad n=3$

b is a number

$\vec{x} = [x_1 \quad x_2 \quad x_3]$

linear algebra : count from 1

```
w= np. array ([1.0 , 2.5 , -3.3])
b = 4
x = np. array ([10 , 20 , 30])
```

code : count from 0

} python code

without vectorization :

$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$

```
f = w[0] * x[0] + w[1] * x[1] + w[2] * x[2] + b
```

↳ very inefficient if n is large

or

```
f = 0
for j in range (n):
    f = f + w[j] * x[j]
f = f + b
```

} not efficient enough

vectorization :

$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

```
f = np. dot (w, x) + b
```

} much faster

↳ less number of code

Date
Page

ⓐ Numpy

↳ it is a library that extends the base capabilities of python to add a richer data set including more numeric types, vectors, matrices, and many matrix functions.

⇒ vectors

↳ Numpy Arrays :

↳ Numpy's basic data structure is an indexable, n-dimentional array containing elements of same type.

↳ Vector Creation :

```
a = np.zeros(4);          a = [0. 0. 0. 0.]
a = np.zeros((4,));       a = [0. 0. 0. 0.]
a = np.random.random_sample(4)
a = np.arange(4.);        a = [0. 1. 2. 3.]
a = np.array([5,4,3,2]);
```

↳ Operations on Vectors :

↳ indexing (starts from zero)

↳ slicing

```
a = np.arange(10)     a = [0,1,2,3,4,5,6,7,8,9]
a[2:7:1]              a = [2 3 4 5 6]
a[2:7:2]              a = [2 4 6]
a[3:]                 a = [3 4 5 6 7 8 9]
a[:3]                 a = [0,1,2]
a[:]                  a = [0 1 2 3 4 5 6 7 8 9]
```

↳ Single vector operations:

```
a = np.array([1,2,3,4])        a = [1 2 3 4]
b = -a                         b = [-1, -2 -3 -4]
b = np.sum(a)                  b = 10
b = np.mean(a)                 b = 2.5
b = a** 2                      b = [1 4 9 16]
```

↳ Vector Vector element wise operation:

```
a = np.array([1,2,3,4])
b = np.array([-1,-2,3,4])
a+b   =   [0,0,6,8]
```

for this to work correctly, the vectors must be of the same size.

↳ Scalar Vector operations:

```
a = np.array([1,2,3,4])
b = 5 * a                      b = [5 10 15 20]
```

↳ Vector Vector Dot Product:

↳ $a \cdot b = \sum_{i=0}^{n-1} a_i b_i$

$$= \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = [a_0 b_0 + a_1 b_1 + a_2 b_2]$$

↳ this requires the dimentions of two vectors to be the same.

↳
```
a = np.array([1,2,3,4])
b = np.array([-1,4,3,2])
c = np.dot(a,b)
```

$c = 24 \left\{ \begin{matrix} 1 \times -1 + 2 \times 4 + 3 \times 3 \\ + 4 \times 2 \end{matrix} \right\}$

⇒ Matrices

↳ these are 2-D arrays.

↳ Creation:

a = np.array([[5],[4],[3]])

↳ a.shape = (3, 1)

↳ a = [[5]
[4]
[3]]

↳ Operations:

↳ indexing

↳ this include two index [row, column]

@ a = np.arange(6).reshape(3,2)

↳ a = [[0 1]
[2 3]
[4 5]]

↳ slicing

↳ [row(start : stop : step), column(_ : _ : _)]

@ Numpy

Date
Page

⇒ Gradient Descent for multiple linear regression:-

parameters : $\vec{w} = [w_1 \; w_2 \; --- \; w_n], \; b$

model : $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

cost function: $J(\vec{w}, b)$

gradient descent: $\left\{ w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b) \right.$

$\left. b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b) \right\}$

Gradiend Descent

One feature

repeat {

$$w = w - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)} \right) x^{(i)}}_{\frac{\partial}{\partial w} J(w,b)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)} \right)$$

simultaneously update w, b

}

n features ($n \geq 2$)

repeat {

$$w_1 = w_1 - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(x^{(i)}) - y^{(i)} \right) x_1^{(i)}}_{\frac{\partial}{\partial w_1} J(\vec{w}, b)}$$

$$\vdots$$

$$w_n = w_n - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) x_n^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)$$

simultaneously update
w_j (for $j = 1, 2, \ldots, n$) and b

}

→ An alternative to gradient descent

↳ Normal Equation
- ↳ only for linear regression
- ↳ solve for w, b without iterations

Disadvantages:
- ↳ Doesn't generalize to other learning algorithms
- ↳ Slow when no. of features is large ($> 10{,}000$)

What you need to know:
- ↳ Normal equation method may be used in ML libraries that implement linear regression
- ↳ Gradient descent is recommeded.

Gradient Descent in Practice :-

⇒ Feature Scall Scaling :-
↳ enables gradient descent to run much faster.

Feature & Parameter values →

price = $w_1 x_1 + w_2 x_2 + b$
x_1 ↓ size, x_2 ↓ no. of bedrooms

x_1 : size (feet²)
range : 300 - 2,000

x_2 : # bedrooms
range : 0 - 5

suppose,
House : $x_1 = 2000$, $x_2 = 5$, price = \$500k

size of the parameters w_1, w_2 ?

$w_1 = 50$, $w_2 = 0.1$, $b = 50$

price = $\underbrace{50 \times 2000}_{100{,}000\,k} + \underbrace{0.1 \times 5}_{0.5\,k} + \underbrace{50}_{50\,k}$

price = \$100,050.5 k
↳ far from true price of the house (\$500 k)

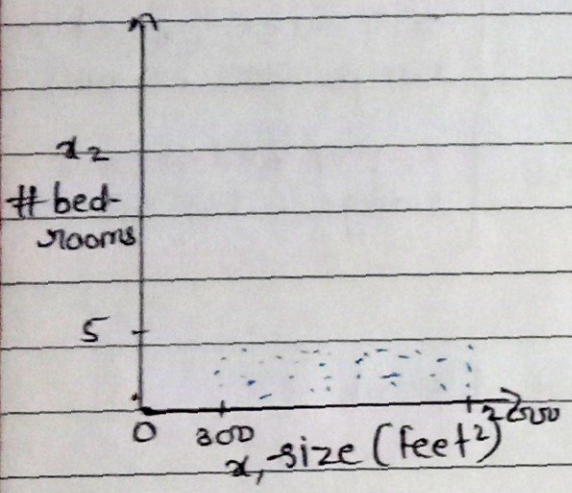
$w_1 = 0.1$, $w_2 = 50$, $b = 50$

price = $\underbrace{0.1 \times 2000}_{200\,k} + \underbrace{50 \times 5}_{250\,k} + \underbrace{50}_{50\,k}$

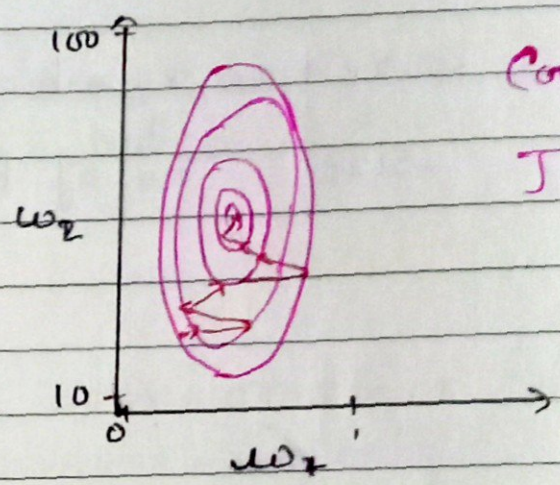
price = \$500k
↳ more reasonable
↳ happens to be the same price as the true price of the house.

↳ when the possible values of a feature is large, it's more likely that a good model will learn to choose a relatively small parameter value (like 0.1) and vice versa.

features

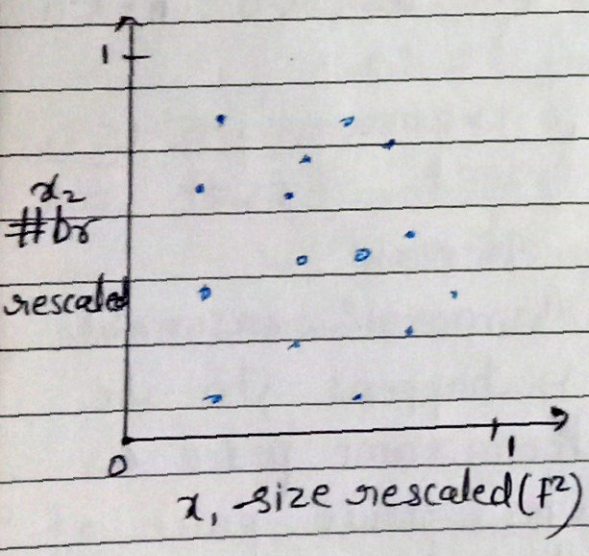


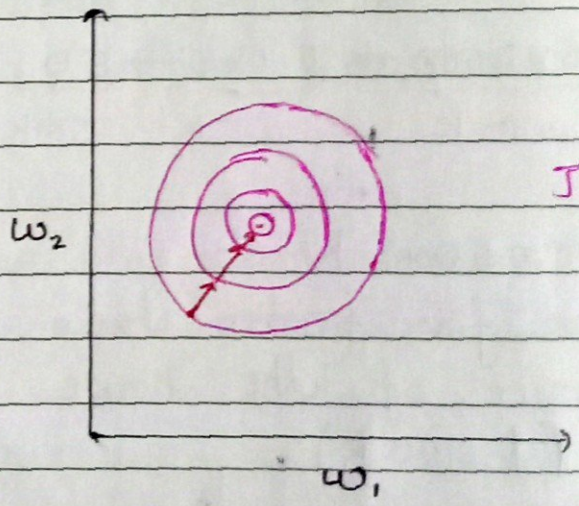
parameters



Contour Plot

$J(\vec{w}, b)$





$J(\vec{w}, b)$

↳ scaling the features makes it easier for gradient descent to minimize the cost function J.

BUT, How to scale features??

↳ if,

$300 \le x_1 \le 2000$

↳ $\boxed{x_{1,scaled} = \frac{x_1}{2000}}$ → max

↓

$0.15 \le x_{1,scaled} \le 1$

&

$0 \le x_2 \le 5$

↳ $\boxed{x_{2,scaled} = \frac{x_2}{5}}$ → max

↳ $0 \le x_2 \le 1$

↳ Mean Normalization

↳ in this way, features will usually range from -1 to 1.

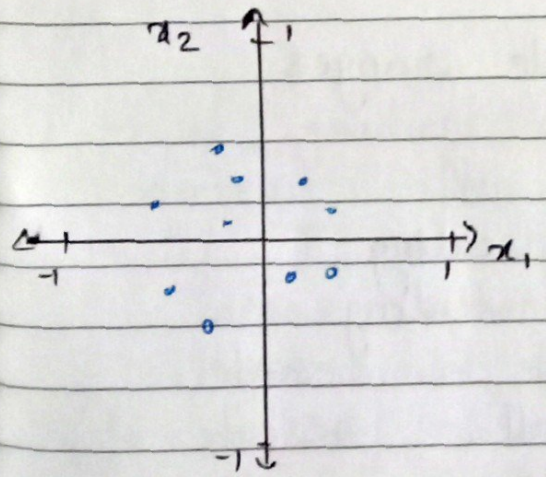
↳ first, take average / mean μ_1.

$300 \le x_1 \le 2000$

↳ $\mu_1 = 600$

↳ $\boxed{x_{1s} = \frac{x_1 - \mu_1}{2000 - 300}}$

↳ $-0.18 \le x_1 \le 0.82$



$0 \le x_2 \le 5$

↳ $\mu_2 = 2.3$

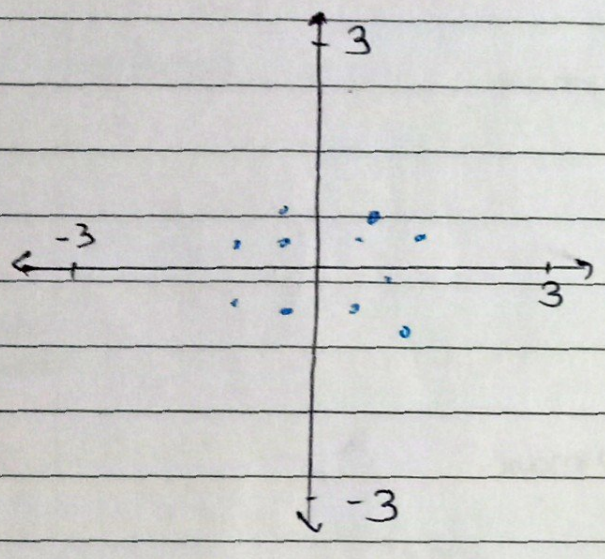
↳ $\boxed{x_{2s} = \frac{x_2 - \mu_2}{5 - 0}}$

↳ $-0.46 \le x_2 \le 0.54$

↳ Z-score Normalization

↳ standard deviation σ

↳ In this method, you have to calculate σ and μ.



↳ $300 \le x_1 \le 2000$

↳ $\mu_1 = 6000$

↳ $\sigma = 450$

↳ $\boxed{x_{1s} = \frac{x_1 - \mu_1}{\sigma_1}}$

↳ $-0.67 \le x_1 \le 3.1$

↳ $0 \le x_2 \le 5$

↳ $\mu_2 = 2.3$, ↳ $\sigma_2 = 1.4$

↳ $\boxed{x_2 = \frac{x_2 - \mu_2}{\sigma_2}}$

↳ $-1.6 \le x_2 \le 1.9$

↳ While rescaling,

↳ aim for about $\boxed{-1 \le x_j \le 1}$ for each feature

↳ $-1 \le x_j \le 1$
$-3 \le x_j < 3$
$-0.3 \le x_j \le 0.3$
} acceptable ranges

↳ $0 \le x_1 \le 3$ okay, no scaling

$-2 \le x_2 \le 0.5$ okay, no scaling

$-100 \le x_3 \le 100$ too large → rescale

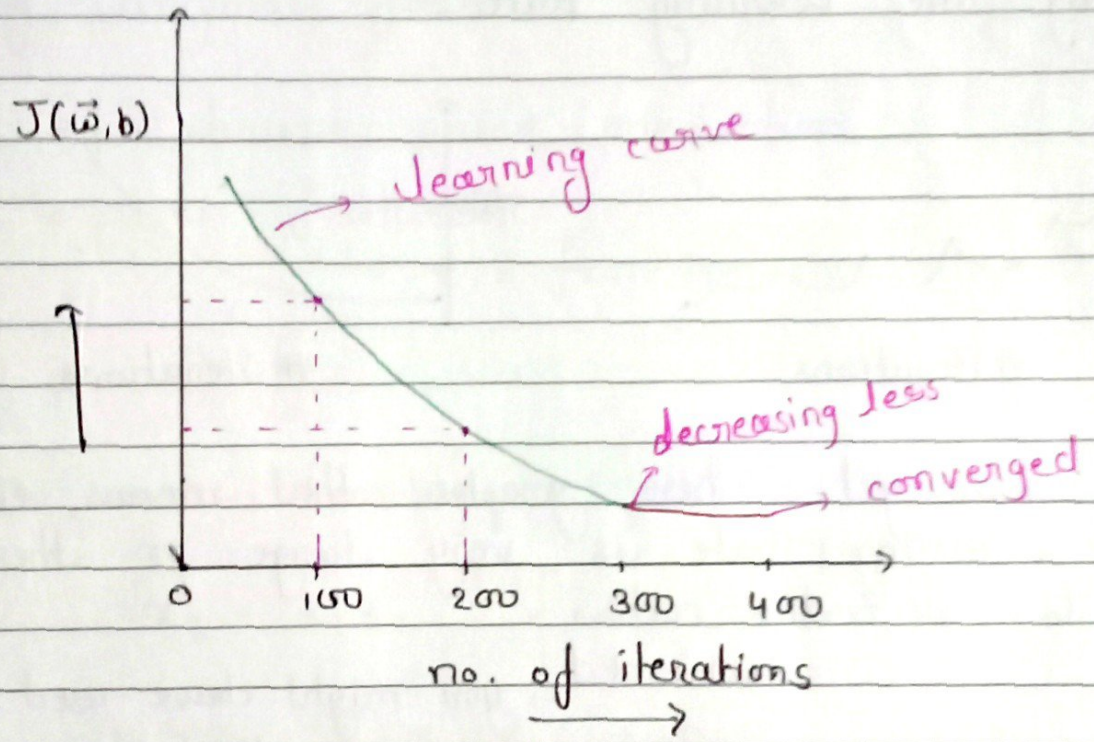
$-0.001 \le x_4 \le 0.001$ too small → rescale

$98.6 < x_5 \le 105$ too large → rescale

⇒ Checking Gradient Descent for Convergence :-

↳ the objective is to minimize $J(\vec{w}, b)$
↳ this can be checked in two ways,
  ↳ plotting a curve of J against iterations
  ↳ automatic convergence test.

↳ Plotting the curve :- ⇒ Preffered



↳ if gradient descent is working properly, J should decrease after every single iteration.
↳ if J ever increases after one iteration, that means either alpha (α) is choosen poorly or there could be bug in the code. ↳ large
↳ no. of iteration for J to be converged can vary.

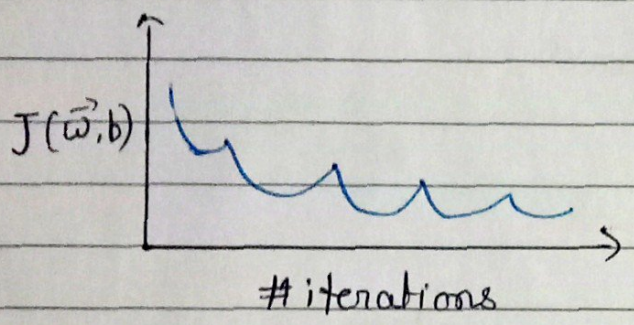
↳ Automatic Convergence Test :-

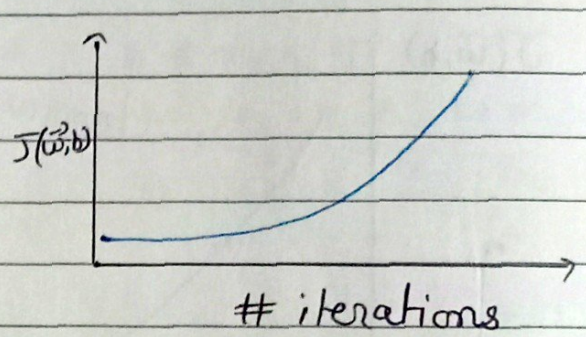
Let, $\varepsilon = 10^{-3}$

↳ if $J(\vec{w}, b)$ decreases by $\leq \varepsilon$ in one iteration, declare convergence.

↳ choosing the right threshold ε is very difficult.

⇒ Choosing The Learning Rate :-





↳ if you get these graphs, that means either the learning rate is very large or there is a bug in the code.

↳ $w = w + \alpha d$ ↳ you might have used a plus sign instead of minus.

↳ with a small enough α, J should decrease on every iteration.

↳ so if gradient descent isn't working, check with a very small α. this will tell you if there is a bug in the code.

values of α to try,

↳ 0.001 ↓ 0.003 ↓ ×3 ↳ 0.01 ↓ 0.03 ↳ 0.1 ↓ 0.3 ↳ 1

⇒ Feature Engineering :-

↳ Choosing or entering the right features is a critical step to making the algorithm work well.

↳ it is the process of using intuition to design new features, by transforming or combining original features.

e.g. of house price prediction.

↳ given features

↳ x_1 = frontage , x_2 = depth

↳ $f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + b$

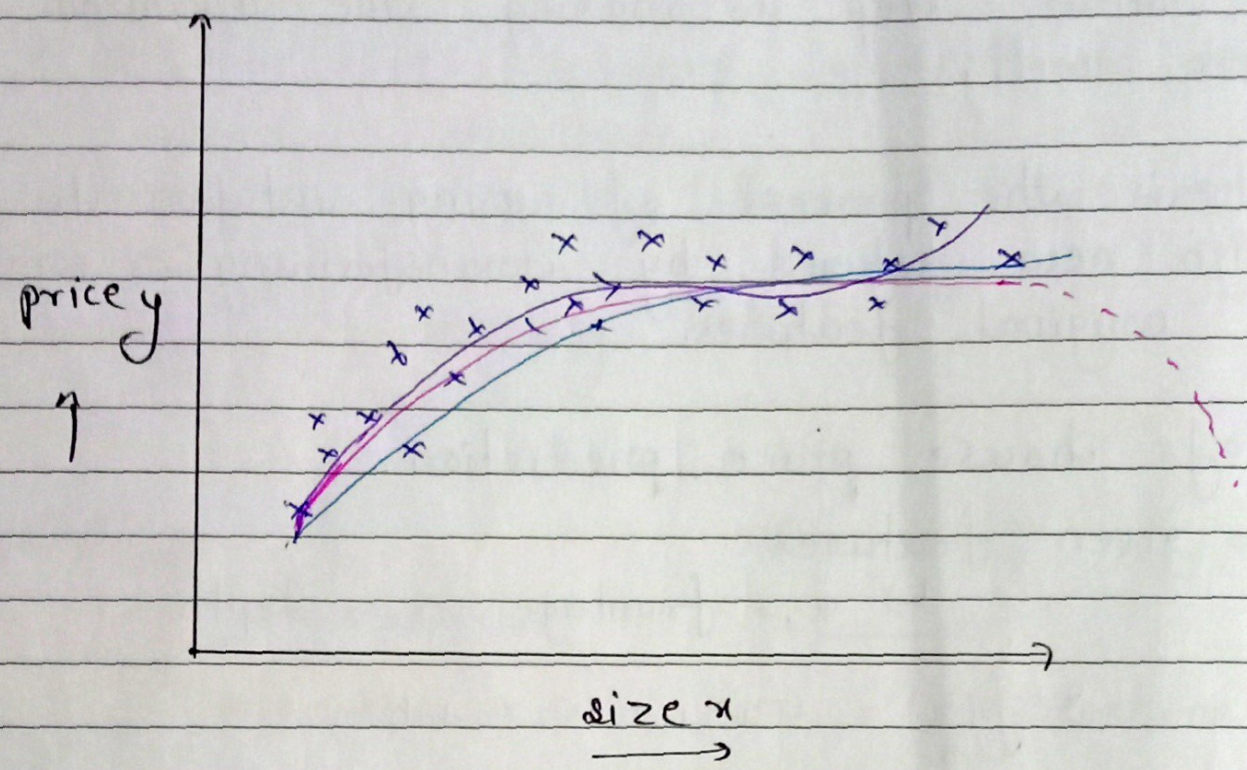
↳ area = frontage x depth {as area is more imp. feature of price}

$x_3 = x_1 x_2$ {we created a new features}

↳ $f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$

⇒ Polynomial Regression :-

↳ multiple linear regression + feature engineering



↳ $f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$ {size + size²}

↳ this may give you a better fit but it comes back down.

↳ $f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + b$

→ if you create features like this, features scaling becomes more imp.

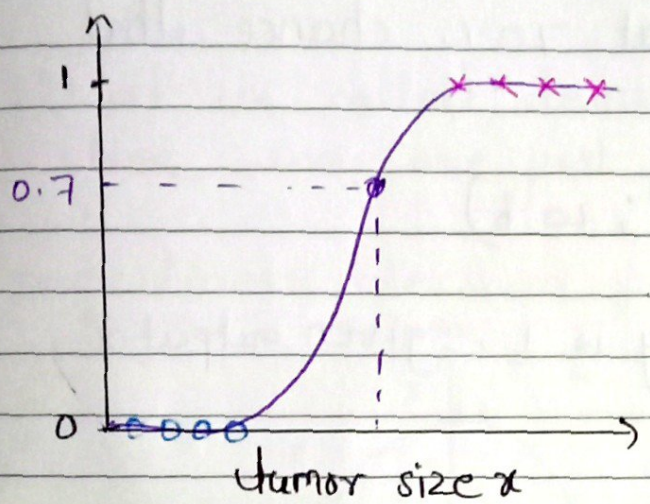
↳ if x : 1 – 10^3 , x^2 : 1 – 10^6 , x^3 : 1 – 10^9

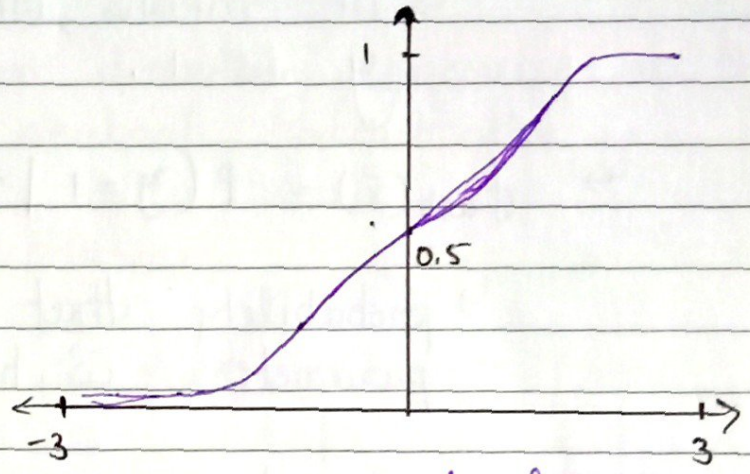
↳ $f_{\vec{w},b}(x) = w_1 x + w_2 \sqrt{x} + b$

Classification with Logistic Regression :-

↳ These are the problems where ~~th~~ we have to predict categories instead of infinite no. of outputs.

↳ Linear regression is not a good algorithm for this problem.





Sigmoid function
↳ out put funct^n
↳ outputs b/w 0 and 1
↳ $g(z) = \frac{1}{1+e^{-z}}$
↳ $0 < g(z) < 1$

$$f_{w,b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

↓

$$z$$

↓

$$\boxed{g(z) = \frac{1}{1+e^{-z}}}$$

↳ $$\boxed{g(z) = g(\vec{w} \cdot \vec{x} + b) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}}$$

↳ logistic regression

Interpretation of logistic regression output :-

$$f_{w,b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

↳ think of it as "probability" that class is 1.

↳ if, $f_{w,b}(\vec{x}) = 0.7$

↳ it means, that 70% chance that y is 1.

↳ $f_{w,b}(\vec{x}) = P(y=1 \mid \vec{x}; w, b)$

↳ probability that y is 1, given output $\vec{x}$, parameters $\vec{w}, b$.

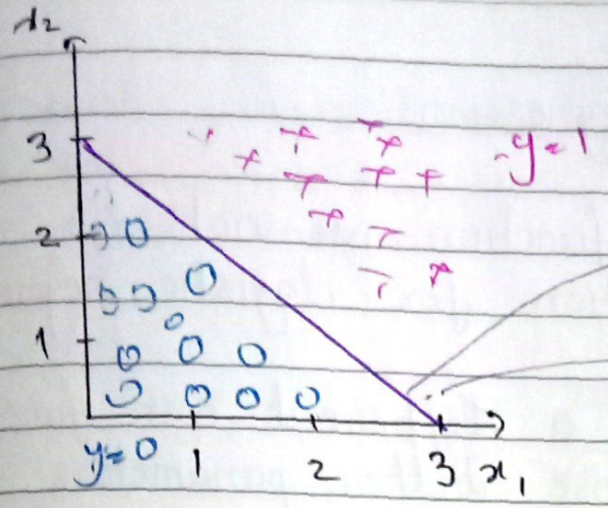
⇒ Decision Boundary :-

is $f_{w,b}(\vec{x}) \geq 0.5$?

↳ Yes: $\hat{y} = 1$ No: $\hat{y} = 0$

→ when is $f_{w,b}(\vec{x}) \geq 0.5$?

$$g(z) \geq 0.5$$
$$z \geq 0$$
$$\vec{w}\cdot\vec{x} + b \geq 0 \qquad \vec{w}\cdot\vec{x} + b < 0$$
$$\hat{y} = 1 \qquad \hat{y} = 0$$



$$f_{w,b}(\vec{x}) = g(z) = g(w_1x_1 + w_2x_2 + b)$$

$$-3$$

Decision boundary:

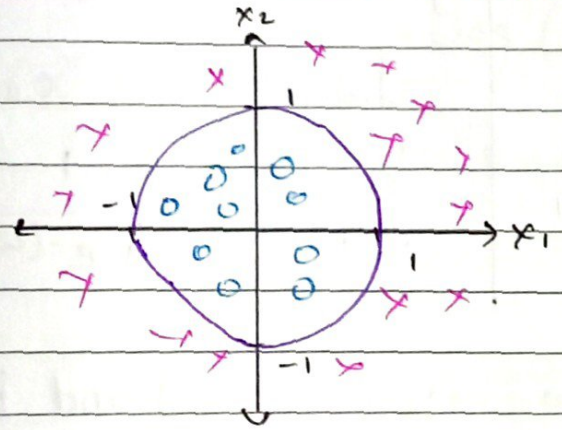
↳ $z = \vec{w}\vec{x} + b = 0$

$= x_1 + x_2 - 3 = 0$

$= \boxed{x + x_2 = 3}$

↳ it is called decision boundary because on the line, we are just neutral about the y.

non-linear decision boundaries :-



$\Rightarrow z = x_1^2 + x_2^2 - 1 = 0$

↳ $\boxed{x_1^2 + x_2^2 = 1}$

↳ you can have more complex decision boundaries like ellipse or any other shape of higher order polynomial.

Cost Function for Logistic Regression :-

↳ The squared error cost function is not an ~~ided~~ ideal cost function for logistic regression.

↳ We'll take a look at a different cost function that can help us choose better parameters.

→ Training Set :-

tumor size x_1	----	Patient's age x_2	malingnant y
10		52	1
2		73	0
5		55	0
12		49	1
...		...	

$i = 1, \ldots, m$: training exam

$j = 1, \ldots, n$: features

target y is 0 or 1?

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

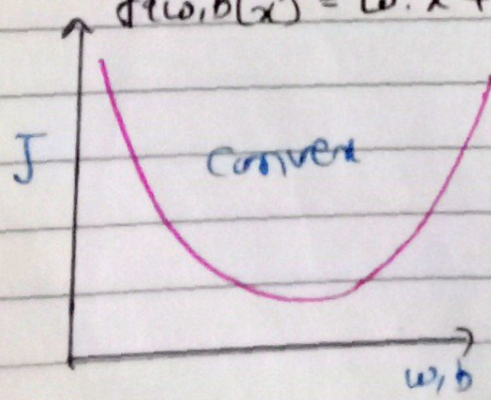
how to choose $\vec{w} = [w_1 \; w_2 \; \ldots \; w_n]$ and b ?

Squared error cost :-

↳ $$J(\vec{w},b) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(f_{\vec{w},b}(x^{(i)}) - y^{(i)} \right)^2$$

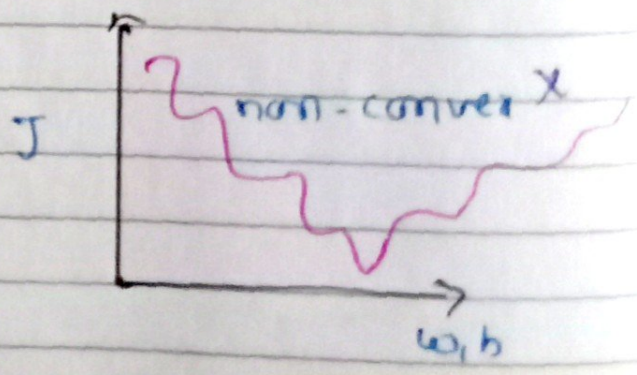
linear regression

$f_{w,b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$



logistic regression

$$f_{w,b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$



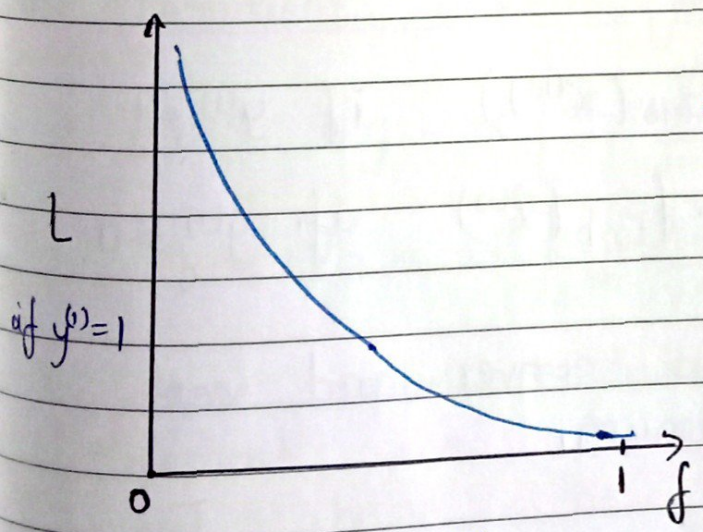
↳ with squared error cost, graph is non-convex, which means you don't necessarily find a global minimum.

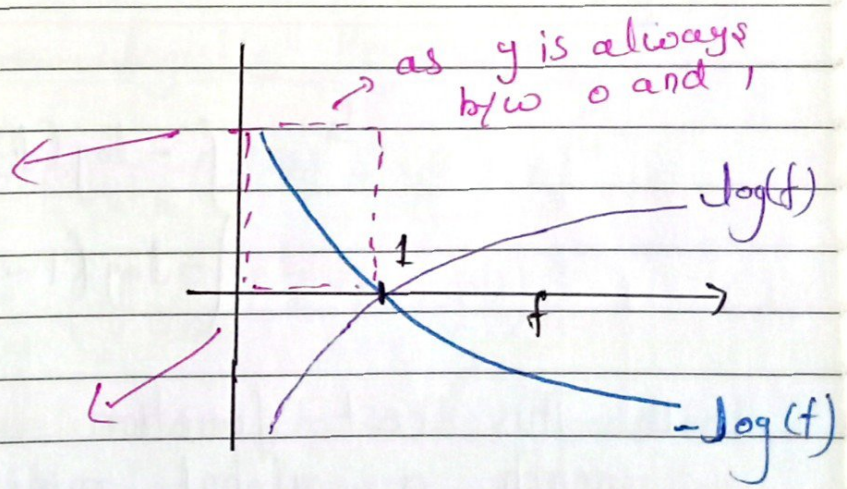
↳ let,

$$J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{1}{2}\left(f_{\vec{w},b}(x^{(i)}) - y^{(i)}\right)^2}_{\text{loss, } L(f_{\vec{w},b}(x^{(i)}),\, y^{(i)})}$$

→ Logistic loss function :-

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(x^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(x^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

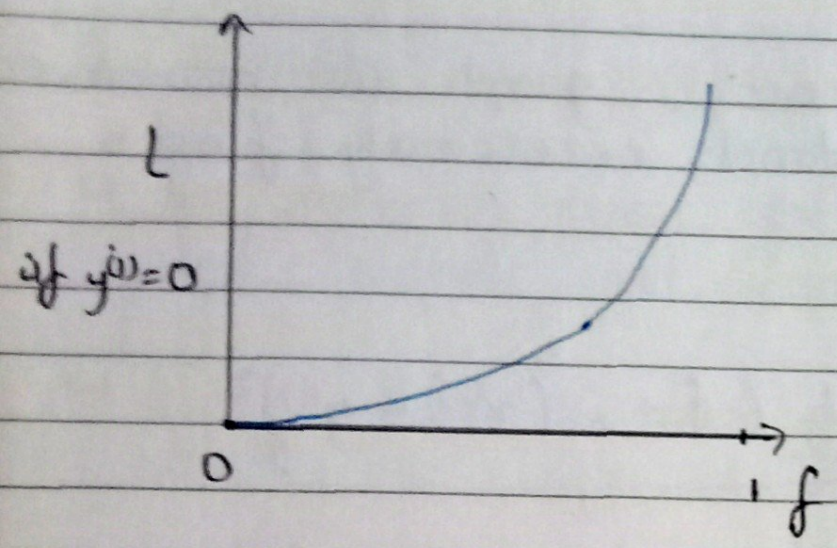


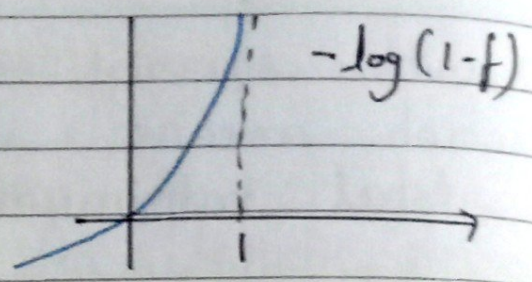


as $f_{\vec{w},b}(x^{(i)}) \to 1$
↳ Loss = 0

and

as $f_{\vec{w},b}(x^{(i)}) \to 0$
↳ Loss $\to \infty$





as $f_{\vec{w},b}(x^{(i)}) \to 0$ and as $f_{\vec{w},b}(x^{(i)}) \to 1$

↳ Loss = 0 ↳ Loss = ∞

⇒ <u>Cost</u>

$$\boxed{J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)}}$$

$$= \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=1 \\ -\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=0 \end{cases}$$

↳ This cost function is convex and can reach a global minimum.

⇒ Simplified Cost Function for Logistic Regression :-

$$\boxed{L = y^{(i)} \log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) - (1-y^{(i)}) \log\left(1 - f_{\vec{w},b}(x^{(i)})\right)}$$

↳ if $y^{(i)} = 1$:

↳ $L = -\log(f(\vec{x}))$

↳ if $y^{(i)} = 0$;

↳ $L = -\log(1 - f(\vec{x}))$

cost $J = \frac{1}{m} \sum_{i=1}^{m} \left[L\left(f_{\vec{w},b}(x^{(i)}), y^{(i)}\right)\right]$

$$\boxed{J = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(f(\vec{x}^{(i)})\right) + (1-y^{(i)}) \log\left(1 - f(\vec{x}^{(i)})\right)\right]}$$

Gradient Descent for Logistic Regression :-

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right] x_j^{(i)}\right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)\right]$$

↳ these equation looks similar to the gradient descent we learnt for linear regression but here the value of $f_{\vec{w},b}(\vec{x}^{(i)})$ is changed.

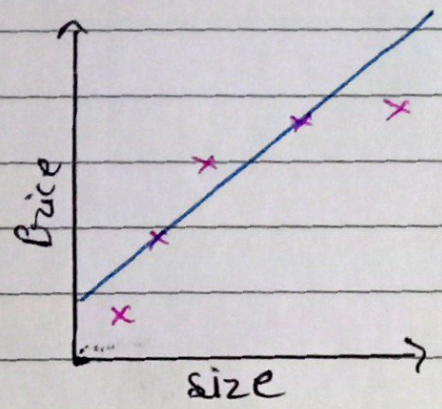
$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x} + b)}}$$

(a) LR model using scikit learn

The Problem of Overfitting :-

Regression example :-

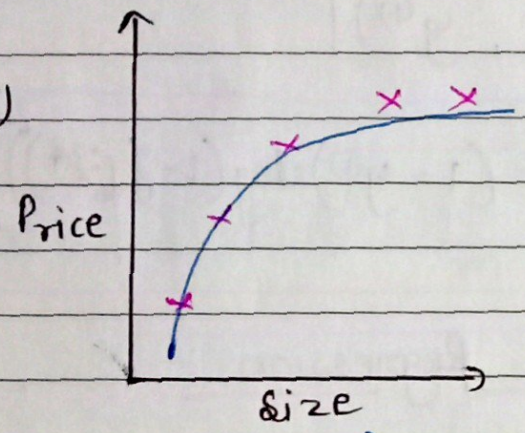
(i)



$w_1 x + b$

→ underfit / high bias
→ does not fit the training set well
→ too few feature

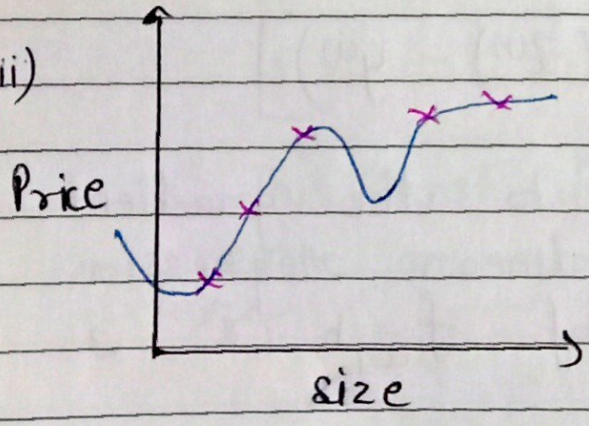
(ii)



$w_1 x + w_2 x^2 + b$

→ just right
→ fits training set pretty well
→ generalization

iii)



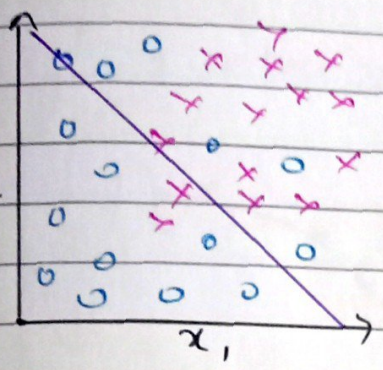
$w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$

→ overfit / high variance
→ fits the training set extremely well
→ too many features

Date
Page

classification example :-

i)

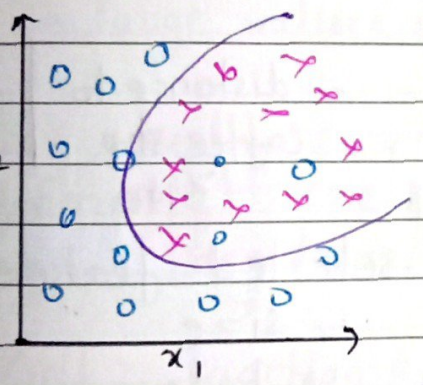


→ underfit / high bias

$z = w x_1 + w_2 x_2 + b$

$f_{\vec{w},b}(\vec{x}) = g(z)$

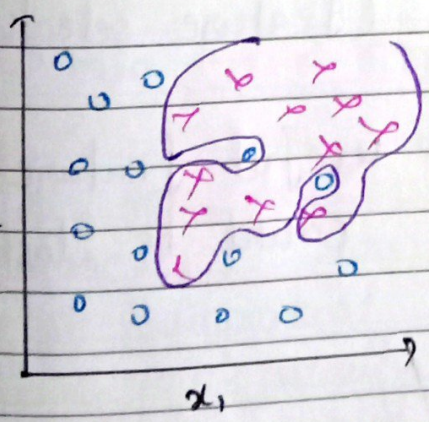
ii)



→ just right

$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2$
$+ w_2 x_2^2 + w_5 x_1 x_2$
$+ b$

iii)

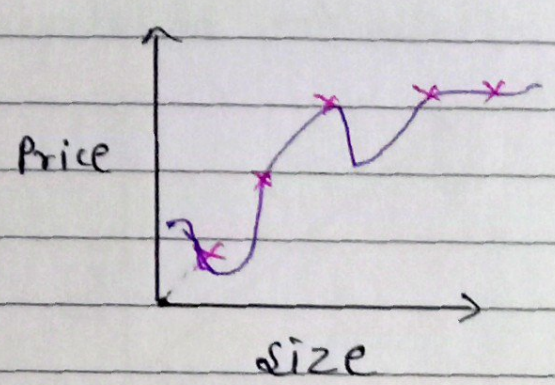


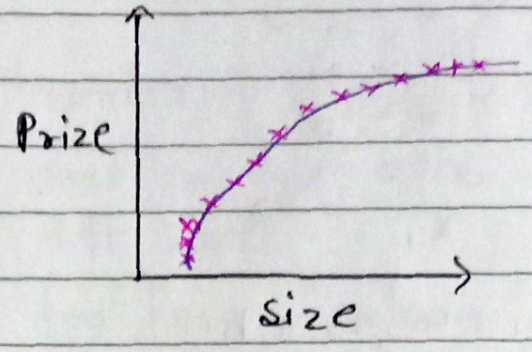
→ overfits

$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_2$
$+ w_4 x_1^2 x_2^2 + w_5 x_1^2 x_2^3$
$+ \quad --- \quad + b$

⇒ Addressing Overfitting :-

↳ Collect more training examples





↳ Select features to include / exclude

Size	bedrooms	floors	age	--- distance to coffee shop	Price
x_1	x_2	x_3	x_4	-- x_{100}	

all features
+
insufficient data
↓
overfit

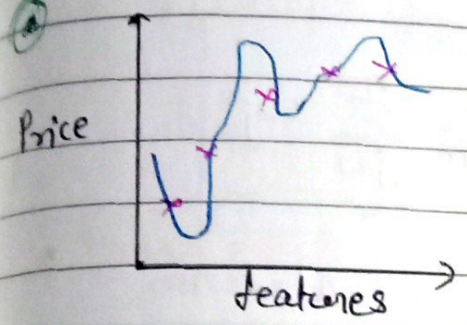
selected features
↳ size
↳ bedrooms
↳ age
↓
just right
(feature selection)

useful features could be lost

Date
Page

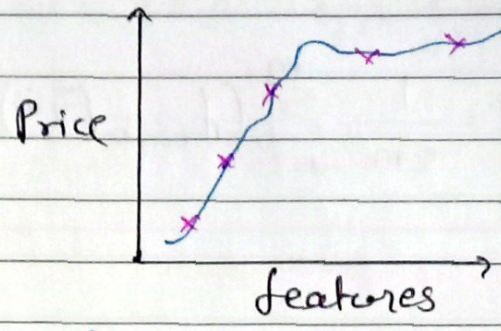
↳ Regularization

↳ reduce the size of parameters w_j



$f(x) = 28x - 385x^2 + 39x^3 - 174x^4 + 100$

↳ large values of w_j



$f(x) = 13x - 0.23x^2 + 0.000014x^3 + 0.0001x^4 + 10$

↳ small values of w_j

↳ we usually regularize only w, not b. It generally creates very small or no effect by regularizing b.

⇒ Cost function with regularization :-

↳ in previous example (a), make w_3 & w_4 really small (≈ 0)

↳ $$\min \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + 1000 w_3^2 + 1000 w_4^2$$

↳ if you have a lot of features, say 100 features, you don't know which are the most important and which one to penalize. So, you usually penalize all the features and it makes your model ~~more~~ smoother, simpler and less prone to overfitting.

⇒ Regularized logistic regression :-

Cost function :-

$$J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)}) \log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

Gradient Descent :-

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b)$$

} simultaneous update

where,

$$\frac{\partial}{\partial w_j} J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_j^{(i)} + \frac{\lambda}{m} w_j$$

$$\frac{\partial}{\partial b} J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

↳ $f_{\vec{w},b}(\vec{x}) = \dfrac{1}{1+e^{(\vec{w}\cdot\vec{x}+b)}}$

⇒ Regularized Linear Regression :-

↳ $J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$

Gradient Descent :-

↳ $w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b)$

↳ $b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b)$

Implementing gradient descent :-

repeat {

$$w_j = w_j - \alpha \underbrace{\left[\frac{1}{m}\sum_{i=1}^{m}\left[(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})\, x_j^{(i)}\right] + \frac{\lambda}{m} w_j\right]}_{\frac{\partial}{\partial w_j} J(\vec{w},b)}$$

$$b = b - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial b} J(\vec{w},b)}$$

} simultaneous update

$$w_j = w\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})\, x_j^{(i)}$$

↳ Suppose you have,

$w_1, w_2, w_3 \ldots, w_n, w_0, b$ ~~feat~~ parameters

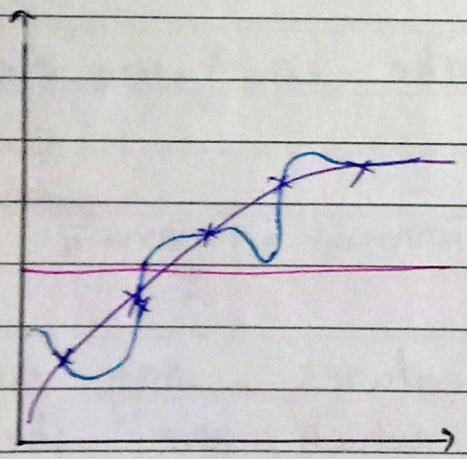
$$\boxed{J(\vec{w},b) = \frac{1}{2m}\sum_{j=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}_{\text{regularization term}}}$$

λ = regularization parameter

$\lambda > 0$

↳ here we will also choose a number for λ.

↳ here we are not penalizing b, but some algorithm also includes $\frac{\lambda}{2m} b^2$ which makes a very little effort.



i) if λ is very small

ii) if λ is very large

iii) if λ is neither too large nor too small.

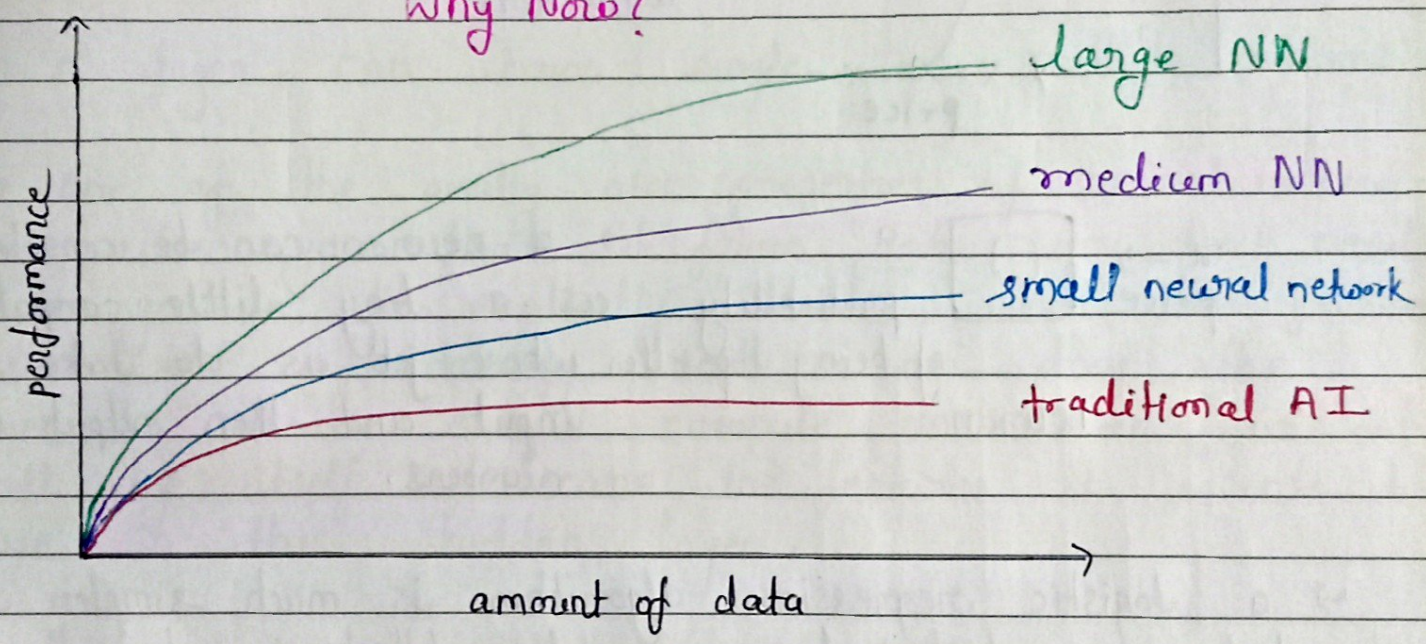
Advanced Learning Algorithms

Neural Network Intuition :-

⇒ Neurons and the brain :-

↳ Neural networks are algorithms that try to mimic the human brain.
↳ Work in neural network started back in 1950s,
↳ ~~Ga~~ Gained popularity in 1980s and early 1990s but fell out of favor again.
↳ Resurgence from around 2005
↳ speech → images → NLP → ----

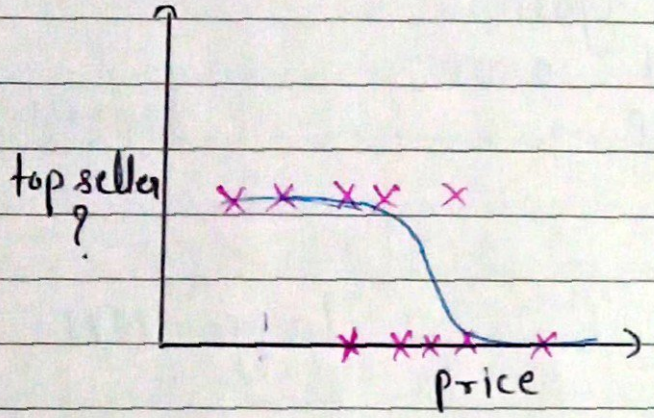
Why Now?



⇒ Demand Prediction :-

↳ look at the product and try to predict, will this product be a top seller or not?

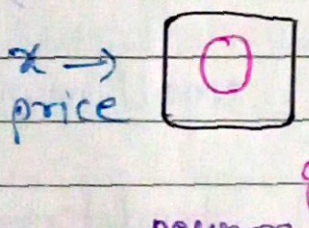
↳ in this example, you are selling t-shirts. You have collected the data and now you want to predict whether this t-shirt will be a top seller or not.



x = price — input

$a = f(x) = \dfrac{1}{1+e^{-(wx+b)}}$ — output

↳ activation



→ a neuron can be considered as a tiny little computer, whose job is to take some input and then output some numbers.

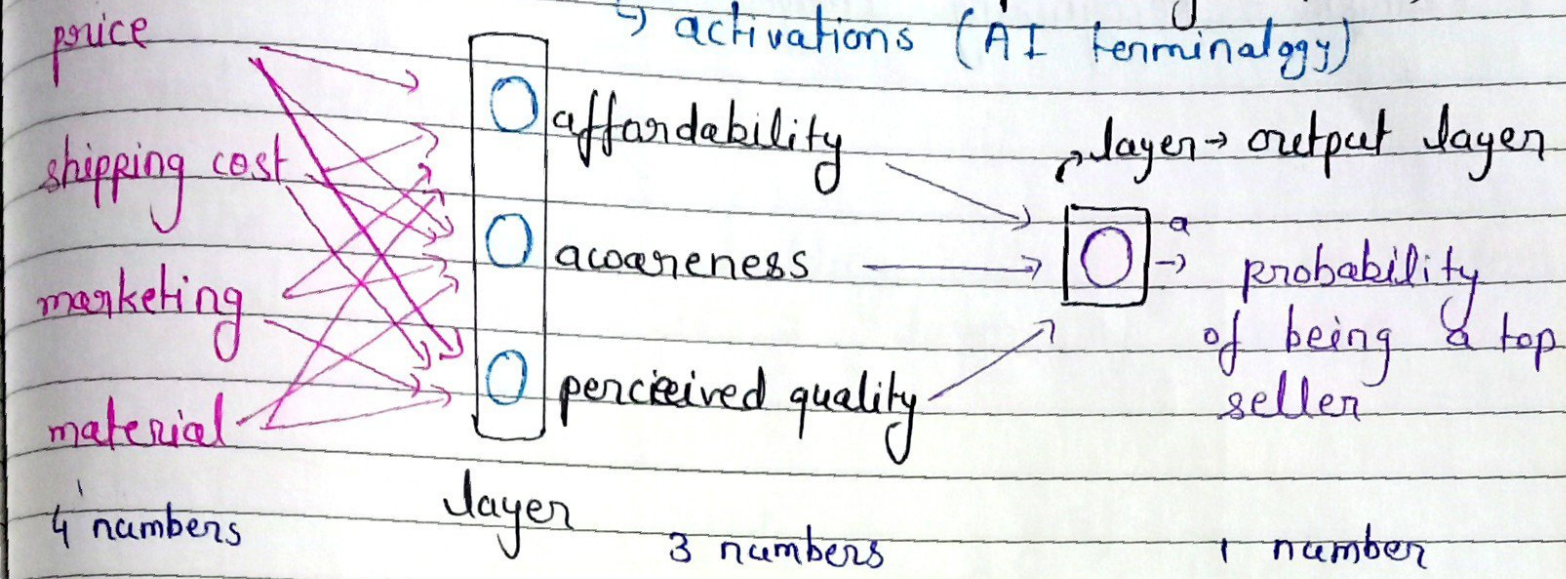
↳ a logistic regression algorithm is much simpler that what any biological neuron does. Which is why the artificial neural network is such a vastly over-simplified model of human brain.

↳ Let's now look at a more complex example of demand prediction. In this example, we are gonna have four features to make predictions.

"hidden layer"

$\vec{x}$ features (input layer) $\vec{a}$ factors (on which probability depends)

↳ activations (AI terminology)



↳ layer is a group of neurons which takes as input the same or similar features, and that in turn outputs a few numbers together.

↳ a layer can have single or multiple neurons.

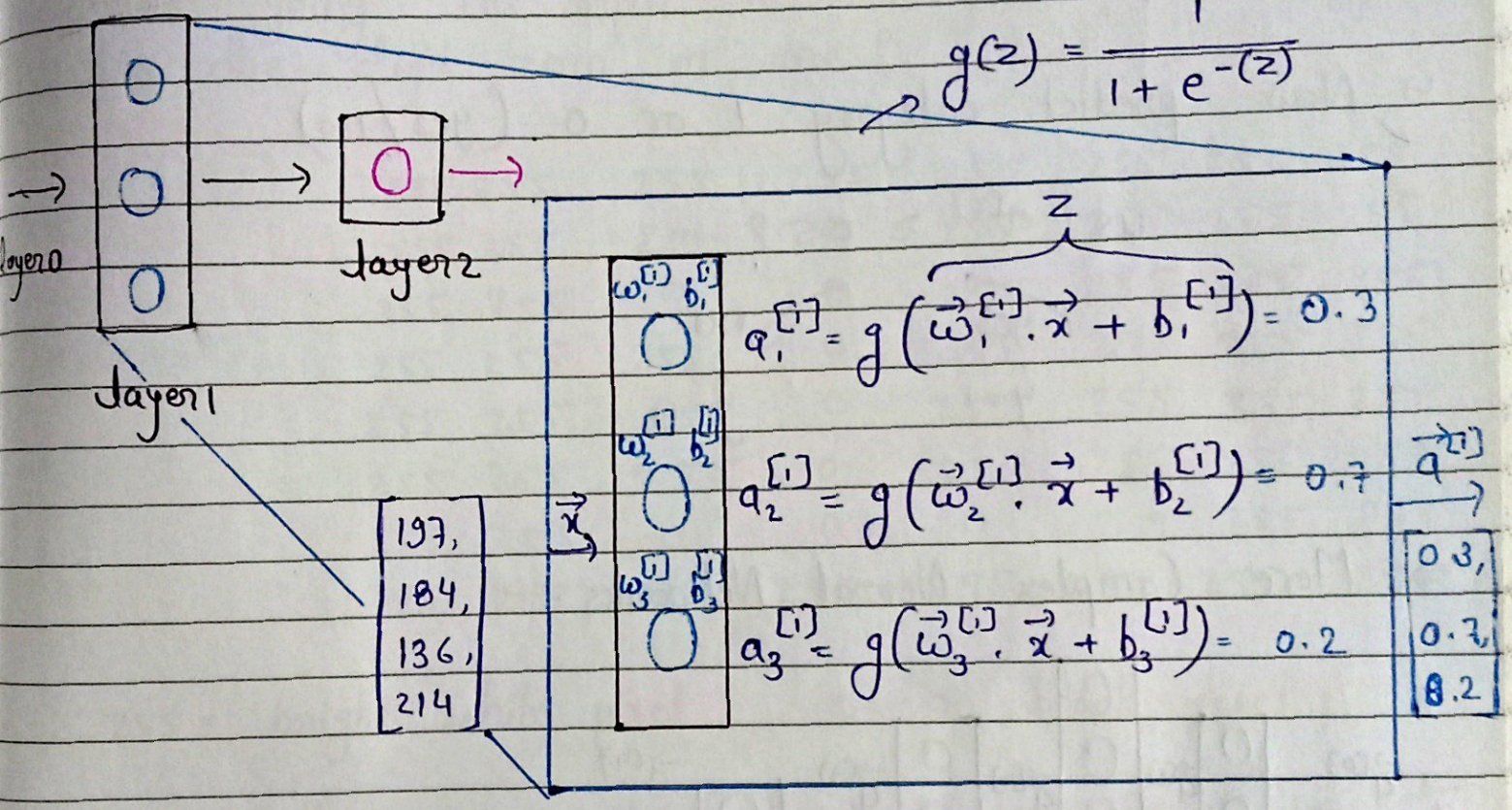
↳ one of the really nice properties of neural network is when you train it from data, you don't need to go in to explicitly decide what other features such as affordability and so on.. that the neural network should compute instead or figure out all by itself what are the features it wants to use in this hidden layer.

↳ There can be multiple hidden layer in a model. It's you who have to decide that how many hidden layers you want and how many neurons each hidden layer may have.

Neural Network Model :-

⇒ Neural Network Layer :-

↳ The fundamental building block of most modern neural networks is a layer of neurons.

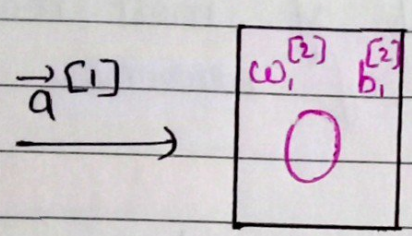


↳ When you build neural network with multiple layers, it'll be useful to give the layers different numbers. by convention, first hidden layer is called layer 1 and so on. The input layer is also sometimes called layer 0.

↳ In order for to introduce notation to help us distinguish between the different layers, we use

for i^{th} layer : $x^{[i]}$

↳ The output of layer 1 becomes the input for layer 2.



$$a_1^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]}) = 0.84 \xrightarrow{a^{[2]}}$$

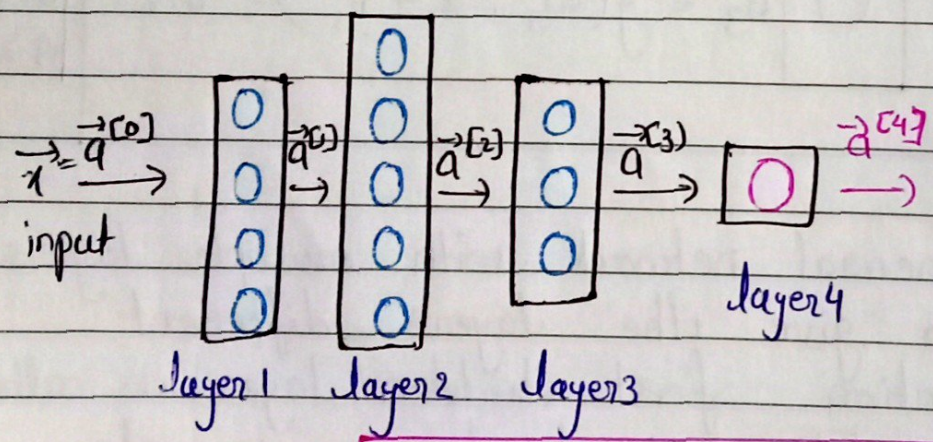
↳ Now predict category 1 or 0 (yes/no)

is $a^{[2]} \geq 0.5$?

yes → $\hat{y} = 1$

no → $\hat{y} = 0$

⇒ More Complex Neural Networks :-



$$\boxed{a_j^{[l]} = g(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})}$$

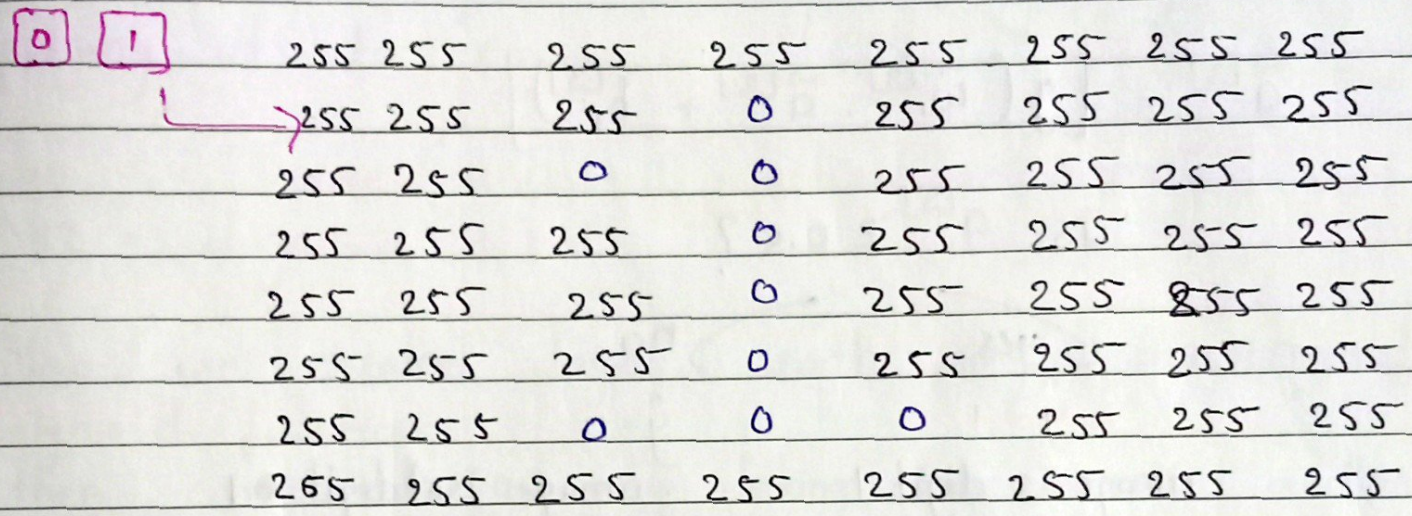
↳ activation function

↳ there are more activation functions other than sigmoid function.

⇒ Inference : Making Predictions (Forward Propagation) :-

↳ We are going to use handwritten digit recognition. And for simplicity we are just going to distinguish between the handwritten digits zeros and one.

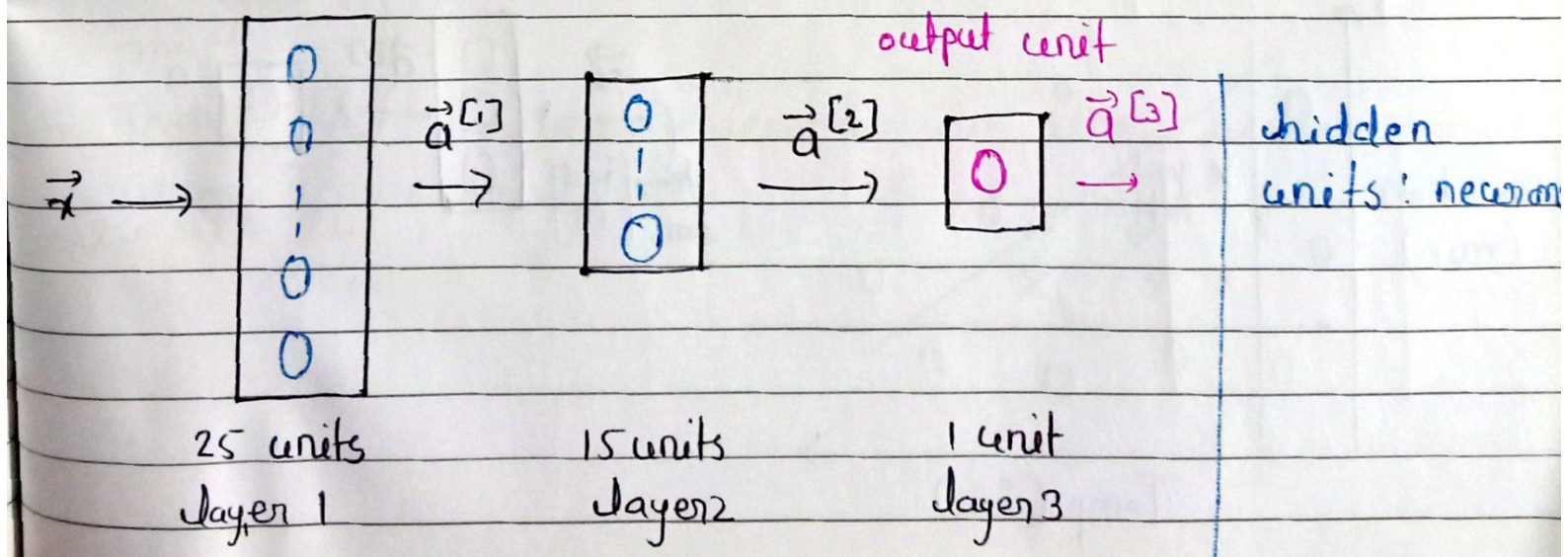
↳ So, it's just a binary classification problem where we're going to input an image and classify, is this the digit zero or one?



255	255	255	255	255	255	255	255
255	255	255	0	255	255	255	255
255	255	0	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	0	0	0	255	255	255
255	255	255	255	255	255	255	255

255 = bright white pixel 0 = a black pixel

↳ Given these 64 input features, we're going to use the neural network with two hidden layers.



$$\vec{a}^{[1]} = \begin{bmatrix} g(\vec{w}_1^{[1]} \cdot \vec{a}^{[0]} + b_1^{[1]}) \\ \vdots \\ g(\vec{w}_{25}^{[1]} \cdot \vec{a}^{[0]} + b_{25}^{[1]}) \end{bmatrix}$$

$$\vec{a}^{[2]} = \begin{bmatrix} g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]}) \\ \vdots \\ g(\vec{w}_{15}^{[2]} \cdot \vec{a}^{[1]} + b_{15}^{[2]}) \end{bmatrix}$$

$$\vec{a}^{[3]} = \left[g(\vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]}) \right]$$

is $a_1^{[3]} \geq 0.5$?

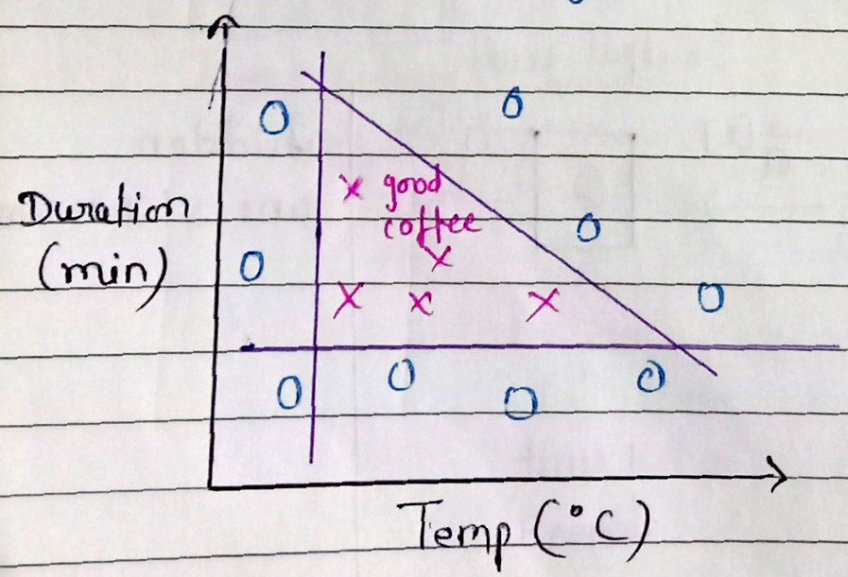
yes → $\hat{y} = 1$ → image is digit 1

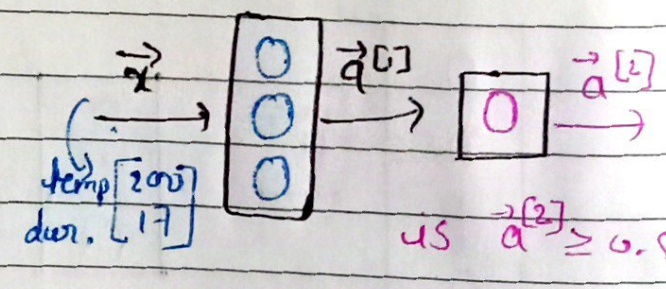
no → $\hat{y} = 0$ → image isn't digit 1

TensorFlow Implementation :-

⇒ Inference in Code :-

↳ Coffee Roasting





is $\vec{a}^{[2]} \geq 0.5$?

```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

// we set the array of two input features :- temp & dur.
// we created layer1 as first hidden layer with 3 neurons or hidden units and sigmoid as the activation function.
// Dense is another name for the layers.
// Next we compute a1 by taking layer_1 and applying it to x.

```
layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
```

// Now, we created layer_2 with single neuron and sigmoid function.
// then computed a2 by taking layer_2 and applying it to a1.

```
if a2 >= 0.5:
    yhat = 1
else:
    yhat = 0
```

↳ That's how we do inference in the neural network using TensorFlow

↳ Model for digit classification -

```
x = np.array([[0.0, ... 245, ... 240 ... 0]])
layer_1 = Dense(units = 25, activation = 'sigmoid')
a1 = layer_1(x)

layer_2 = Dense(units = 15, activation = 'sigmoid')
a2 = layer_2(a1)

layer_3 = Dense(units = 1, activation = 'sigmoid')
a3 = layer_3(a2)

if a3 >= 0.5:
    yhat = 1
else:
    yhat = 0
```

⇒ Data in TensorFlow :-

↳ Note about numpy arrays

$$\begin{array}{c} \text{3 columns} \\ \text{2 rows} \left\{ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \right. \end{array}$$

2×3

```
x = np.array([[1,2,3],
              [4,5,6]])
```

→ represents a row

→ groups together all the rows

$$\begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -.5 & -.6 \\ 7 & 8 \end{bmatrix}$$

4×2

```
x = np.array([[0.1, 0.2],
              [-3.0, -4.0],
              [-0.5, -0.6],
              [7.0, 8.0]])
```

Date
Page

$x = np.array([[200, 17]])$ $\quad [200 \;\; 17]$ $\quad$ 1×2

$x = np.array([[200], [17]])$ $\quad \begin{bmatrix} 200 \\ 17 \end{bmatrix}$ $\quad$ 2×1

$x = np.array([200, 17])$ → this is just a 1-D array with no rows or columns but just a list of numbers.

↳ when we are working with linear regression and logistic regression, we use these 1-D vectors to represent the input features x.

↳ With tensorflow, the convention is to use matrices to represent the data.

↳ So, while working with numpy and tensorflow, we get to use two data types numpy array and tf.Tensor.

↳ These two data types can be switched into one another easily.

x.numpy() - convert Tensor to np array.

Date
Page

⇒ Building a Neural Network :-

```
layer_1 = Dense (units = 3 , activation = 'sigmoid')
layer_2 = Dense (units = 1 , activation = 'sigmoid')

model = Sequential ([layer_1 , layer_2])
// we created the neural network by combining all
the layers in this way.

x = np.array([ [200.0 , 17.0],
               [120.0 , 5.0],
               [425.0 , 20.0],
               [212.0 , 18.0]])
y = np.array([1 , 0 , 0 , 1])

model.compile (...)
model.fit (x , y)
model.predict (x_new)
```

↳ Or , we can create the model in another way.

```
model = Sequential ([
        Dense (units = 3 , activations = 'sigmoid')
        Dense (units = 1 , activation = 'sigmoid')])
```

Vectorization :-

How Neural Networks are Implemented Efficiently :-

For Loops vs. Vectorization

```
x = np.array([200, 17])
W = np.array([[1, -3, 5],
              [-2, 4, -6]])
b = np.array([-1, 1, 2])
def dense(a_in, W, b):
    units = W.shape[1]
    a_out = np.zeros(units)
    for j in range(units):
        w = W[:, j]
        z = np.dot(w, a_in) + b[j]
        a_out[j] = g(z)
    return a_out
```

```
X = np.array([[200, 17]])
W = np.array([[1, -3, 5],
              [-2, 4, -6]])
B = np.array([[-1, 1, 2]])
def dense(A_in, W, B):
    Z = np.matmul(A_in, W) + B
    A_out = g(Z)
    return A_out
```

matrix multiplication

Neural Network Training :-

⇒ TensorFlow Implementation :-

```
model = Sequential ( [
        Dense ( units = 25 , activation = 'sigmoid'),
        Dense ( units = 15 , activation = 'sigmoid'),
        Dense ( units = 1 , activation = 'sigmoid'),
  ])
```

step 1

```
model.compile( loss = BinaryCrossentropy())
```

step 2

```
model.fit ( X , Y , epochs = 100)
```

step 3

↳ no. of steps in gradient descent

⇒ Training Details :-

Steps	logistic regre.	neural network
① specify how to compute output given input x and parameters w, b.	$z = np.dot(w, x) + b$ $f_x = 1 / (1 + np.exp(-z))$	model = Sequential ([Dense (...) Dense (...)])
2) specify loss & cost	$loss = -y \times np.log(f_x) - (1-y) \times np.log(1 - f_x)$ $cost = \frac{1}{m} \sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$	binary cross entropy model.compile(loss = BinaryCrossentropy())
3) train on data to minimize $J(\vec{w}, b)$	$w = w - \alpha \times dj_dw$ $b = b - \alpha \times dj_db$	model.fit(X, Y, epochs=100)

Date
Page

1. Create the model

↳ This code snippet specifies the entire architecture of the neural network.
↳ It tell you about the hidden layers, hidden units and activation functions.
↳ We also know what are parameters.

→ —"—

2. Loss and Cost Functions

handwritten digit classification problem → binary classification

↳ $L = -y \log(f(\vec{x})) - (1-y) \log(1 - f(\vec{x}))$

↳ in tf, this is called binary cross entropy loss.

→ —"—

→ in case of linear regression problem, you can use mean squared error function.

3. Gradient Descent

repeat {

$$w_j^{(l)} = w_j^{(l)} - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

$$b_j^{(l)} = b_j^{(l)} - \alpha \frac{\partial}{\partial b_j} J(\vec{w}, b)$$ } ~~cop~~ compute derivatives for gd using "back propagation".

→ —"—

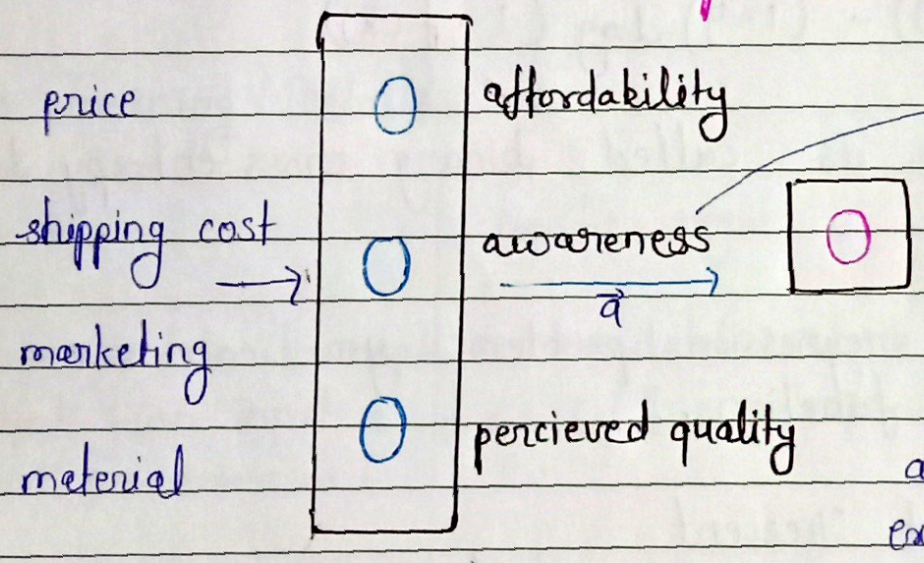
Activation Functions :-

⇒ Alternatives to the sigmoid activation :-

↳ So far, we have been using the sigmoid activation function in all the nodes. because we were building up neural networks by taking logistic regression and creating lr units and string them together.

↳ There are other activation functions which can make your NN much more powerful.

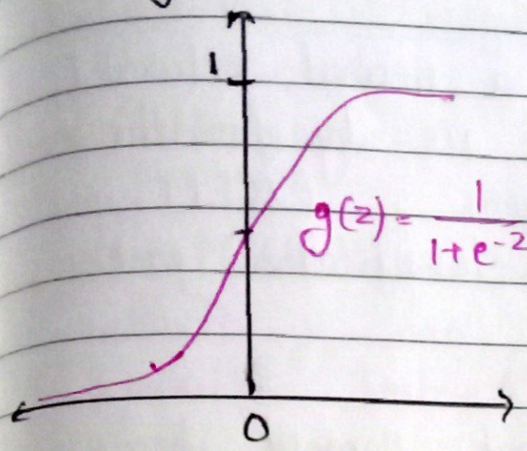
→ Recall the demand prediction example



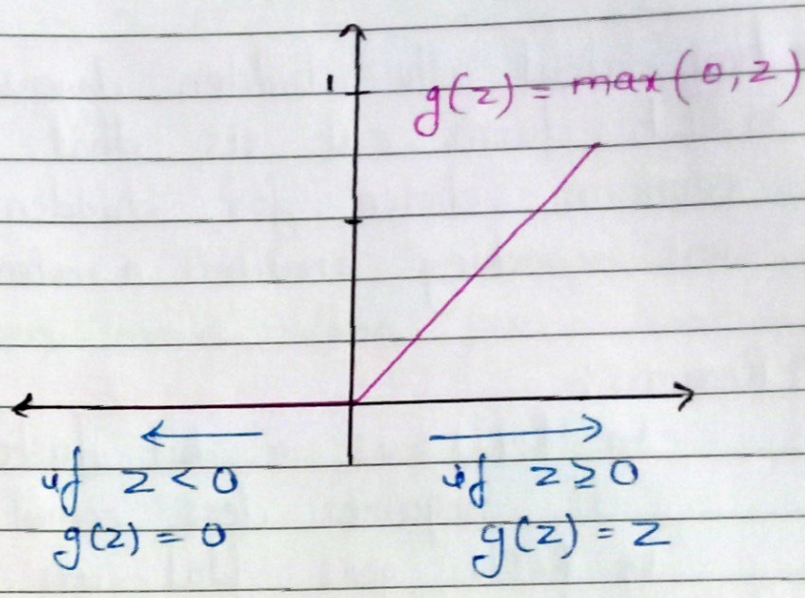
here we assumed that it is binary. Either people are aware or not. But in real, this may not be binary. people can be a little bit aware, somewhat aware, extremely aware or it could have gone completely viral.

↳ So, rather than modeling awareness as just a number between 0 and 1, maybe it should be any non negative number.

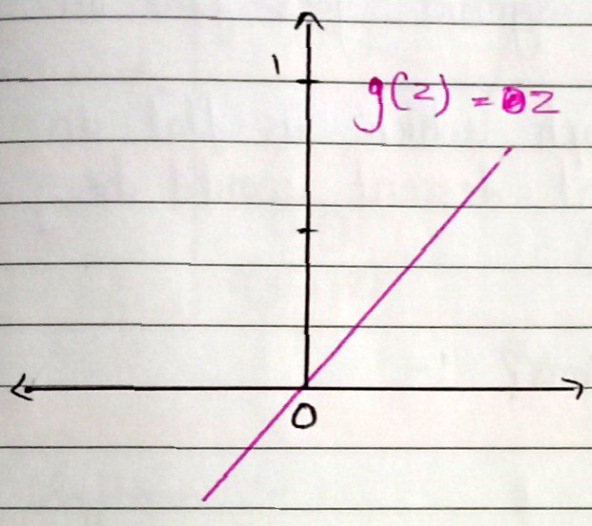
Sigmoid



ReLU - Rectified Linear Unit



Linear Activation Function



→ Sometimes, while using Linear Activation function, people will say we're not using any activation function.

∴ $g(z) = z$

↳ means $g()$ is useless.

↳ z is enough in itself.

⇒ Choosing Activation Functions :-

↳ Let's see how you can choose activation functions in different neural networks.

↳ We start with choosing activation function for output layer depending upon what the target label is.

↳ Binary Classification → $y = 0/1$ → Sigmoid

↳ Regression → $y = +/-$ → Linear Activation Function

↳ Regression → $y = 0$ or $+$ → ReLU

Date
Page

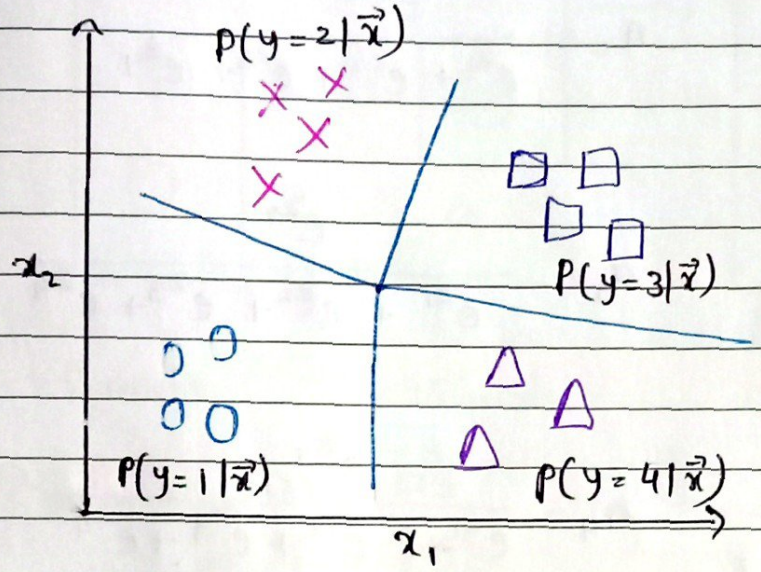
↳ How about the hidden layers of a neural network!?
↳ It turns out is that ReLU is by far the most common choice for hidden layers EXCEPT you are working with a binary classification problem.

↳ Reasons:-
↳ ReLU is a bit faster that sigmoid because it requires less ~~comut~~ computation.
↳ ReLU goes flat ~~in~~ ~~at~~ only in one part of the graph whereas sigmoid goes flat in two places.
↳ If you have a graph which is flat in many places, gradient descent would be really slow.

Multiclass Classification :-

⇒ Multiclass :-

↳ Multiclass Classification Problems are problems in which y can take on more than just two possible values.



⇒ Softmax :-

↳ It is a generalization of logistic regression algorithm which is a binary classification algorithm to the multiclass classification context.

Logistic Regression (2 possible outputs)

$$z = \vec{w} \cdot \vec{x} + b$$

$$\times \quad a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$$

$$O \quad a_2 = 1 - a_1 = P(y=0|\vec{x})$$

Date
Page

Softmax Regression (more than 2 possible outputs)

X $z_1 = \vec{w}_1 \cdot \vec{x} + b_1$ $\qquad a_1 = \dfrac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

O $z_2 = \vec{w}_2 \cdot \vec{x} + b_2$ $\qquad a_2 = \dfrac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

□ $z_3 = \vec{w}_3 \cdot \vec{x} + b_3$ $\qquad a_3 = \dfrac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

△ $z_4 = \vec{w}_4 \cdot \vec{x} + b_4$ $\qquad a_4 = \dfrac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

ⓐ $a_1 + a_2 + a_3 + a_4 - - - + a_n = 1$

General Case →

$y = 1, 2, 3, 4, - - - -, N$

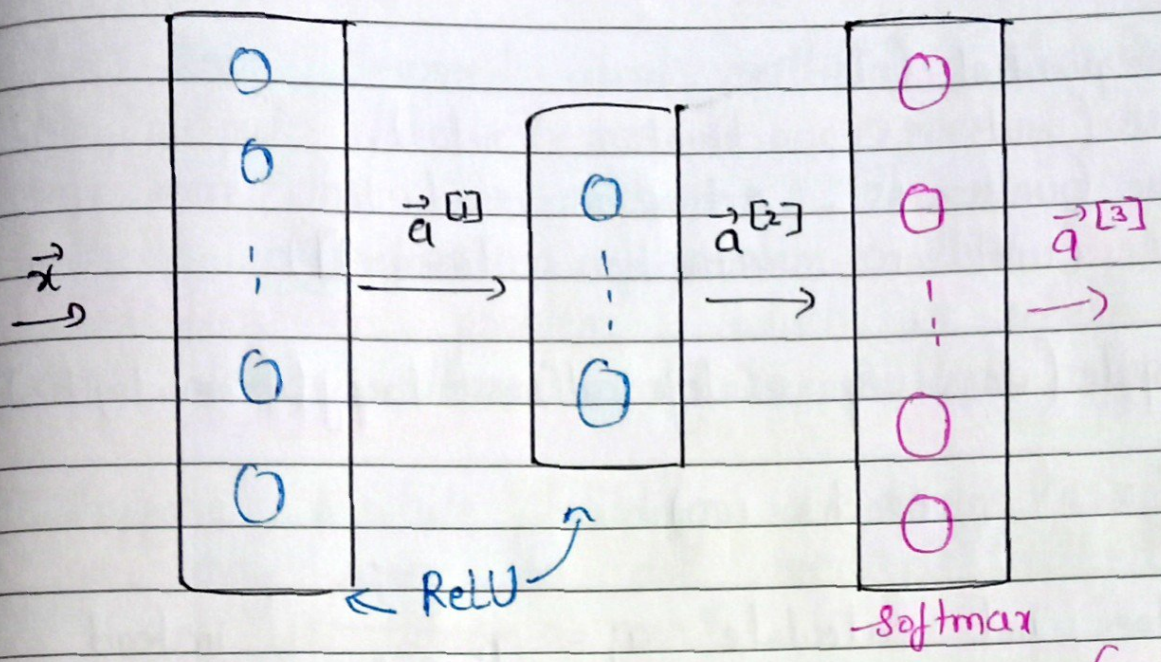
① $z_j = \vec{w}_j \cdot \vec{x} + b_j \qquad j = 1, - -, N$

② $a_j = \dfrac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}}$

③ $a_1 + a_2 + - - + a_n = 1$

④ $\text{loss} = -\log a_j$ for $y = j$

Neural Network with Softmax Output :-



$$z_j^{[3]} = w_j^{[3]} \cdot \vec{a}^{[2]} + b_j^{[3]} \quad , \quad a_j^{[3]} = \frac{e^{z_j^{[3]}}}{\sum_{k=1}^{10} e^{z_k^{[3]}}}$$

```
model = Sequention ([                                    step.1
    Dense (units = 25 , activation = 'relu'),
    Dense (units = 15 , activation = 'relu'),
    Dense (units = 10 , activation = 'softmax')])

model.compile (loss = SparseCategoricalCrossentropy())   step.2

model.fit (X , Y, epochs = epochs = 100)                 step.3
```

↳ There is a better version of code we'll see later.

⇒ Improved Implementation of Softmax :-

```
model = Sequential ([
        Dense (units = 25 , activation = 'relu'),
        Dense (units = 15 , activation = 'relu'),
        Dense (units = 10 , activation = 'linear')])

model.compile (loss = SparseCategoricalCrossEntropy (from_logits = True))

model.fit (X , Y , epochs = 100)
```

↳ this does not calculate $a_j = \frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}}$, instead it directly calculates loss function as,

$$L(\vec{a}, y) = \begin{cases} -\log \frac{e^{z_1}}{e^{z_1} + \cdots + e^{z_{10}}} & \text{if } y = 1 \\ \vdots & \\ -\log \frac{e^{z_{10}}}{e^{z_1} + \cdots + e^{z_{10}}} & \text{if } y = 10 \end{cases}$$

which gives more accurate results. (No roundoff error)

<u>predict</u> :

```
logits = model(X)

f_x = tf.nn.softmax(logits)
```

Classification with Multiple Outputs :-

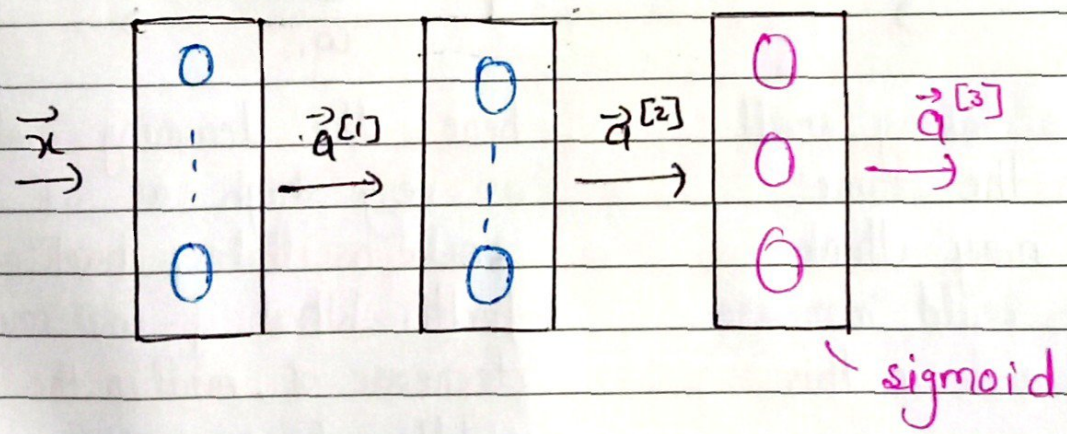
↳ You have learned about multiclass classification, where the output label Y can be any one of two or potentially many more than two categories. There's a different type of classification problem called a multi-label classification problem, which is where associate of each image, there could be multiple outputs.

↳ Suppose, while building a driving assistant program, you may want to ask questions like :-

is there a car?
is there a bus?
is there a pedestrian?

$$y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad / \quad y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad | \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \quad \cdots$$

Here you will get 3 outputs.



Additional Neural Network Concepts :-

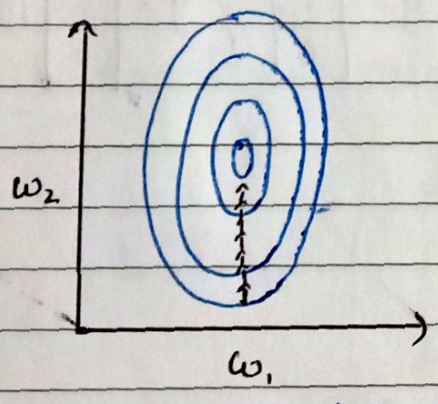
⇒ Advance Optimization :-

↳ There are now some optimization algorithms for minimizing cost function, that are even better than gradient descent.

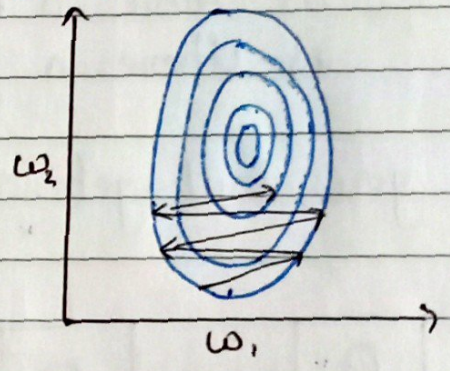
recall gradient descent -

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

↳ learning rate



here, we are taking small tiny steps in the same direction. We may think, what if we could increase the learning rate in this case.



here, the learning rate is very high as it is ~~oscilat~~ oscillating back and forth. What if we could decrease α ~~midl~~ in the middle of processing.

The 'Adam' algorithm does it for us!!!
↳ adaptive moment estimation

```
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-3))
```

Advice for applying Machine Learning :-

⇒ Deciding what to try next :-

Debugging a learning algorithm

You have implemented regularized linear regression on housing prices

$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

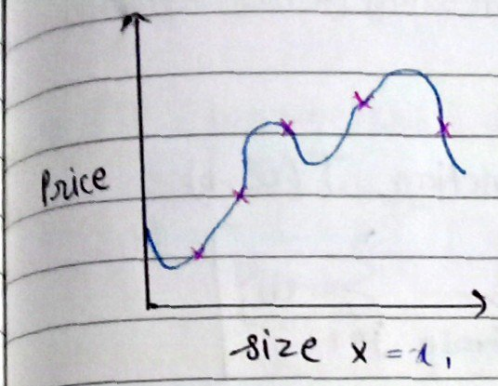
But it makes unacceptably large errors in predictions. What do you try next?

- → Get more training examples.
- → Try smaller set of features
- → Try getting ~~addith~~ additional features
- → Try adding polynomial features (x_1^2, x_2^2, x_1x_2 etc)
- → Try decreasing λ
- → Try increasing λ

Diagnostic = A test that you run to gain insight into what is/isn't working with a learning algorithm, to gain guidance into improving its performance.

Diagnostic can take time.
But doing so would be very good use of your time.

⇒ Evaluating a Model :-



model fits the training data well but will fail to generalize to new examples not in the training set.

$f_{\vec{w},b} = w_1 x + w_2 x^2 + \dots + w_n x^n + b$

x_1 = size in feet^2
x_2 = no. of bedrooms
x_3 = no. of floors
x_4 = age of home in years

→ with only one feature, we can plot a 2-D graph and see how our model is working.
But, if the no. of features is large, you can not plot the graph and observe it.
In this case, we have other methods the check our model.

↳ If you have a training set, rather than using all the data to train the model, you can instead split the training set into two subset.

size	price
2104	400
1600	330
2400	369
70% 1416	323
3000	540
1985	300
1534	315
1427	199
30% 1380	212
1494	243

m_{train} = no. of training ex.

training set →

$(x^{(1)}, y^{(1)})$
$(x^{(2)}, y^{(2)})$
⋮
$(x^{(m_{train})}, y^{(m_{train})})$

test set

m_{test} = no. of test ex.

$(x^{(1)}_{test}, y^{(1)}_{test})$
⋮
$(x^{(m_{test})}_{test}, y^{(m_{test})}_{test})$

Train / Test procedure for linear regression (with squared error cost) :-

→ Fit parameters by minimizing cost function $J(\vec{w}, b)$

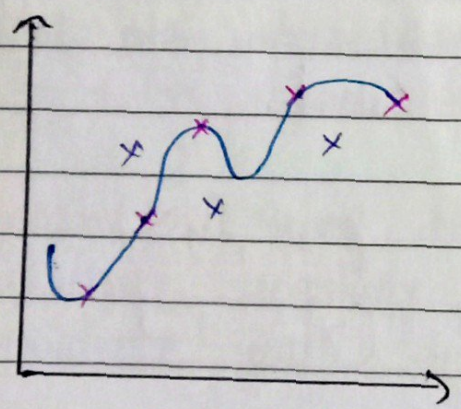
$$J(\vec{w},b) = \left[\frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2\right]$$

→ Compute test error :

$$J_{test} = \frac{1}{2m_{test}} \left[\sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}(\vec{x}^{(i)}_{test}) - y^{(i)}_{test}\right)^2\right]$$

→ Compute training error :

$$J_{train} = \frac{1}{2m_{train}} \left[\sum_{i=0}^{m_{train}} \left(f_{\vec{w},b}(x^{(i)}_{train}) - y^{(i)}_{train}\right)^2\right]$$



x = train

x = test

here, J_{train} will be low but J_{test} will be very high. Which means your ~~d~~ model is now predicting ~~two~~ well with new data.

Train / Test procedure for classification problem :-

→ Fit parameters by minimizing $J(\vec{w},b)$ to find $\vec{w}, b$

$$J(\vec{w},b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} \left[y^{(i)} \log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)}) \log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)\right] + \frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2$$

→ Compute ~~test~~ test error :

$$J_{test} = \frac{1}{-m_{test}} \sum_{i=1}^{m_{test}} \left[y^{(i)}_{test} \log\left(f_{\vec{w},b}(x^{(i)}_{test})\right) + (1-y^{(i)}_{test}) \log\left(1-f_{\vec{w},b}(x^{(i)}_{test})\right)\right]$$

→ Compute training error :

$$J_{train} = \frac{1}{-m_{train}} \sum_{i=1}^{m_{train}} \left[y^{(i)}_{train} \log\left(f_{\vec{w},b}(x^{(i)}_{train})\right) + (1-y^{(i)}_{train}) \log\left(1-f_{\vec{w},b}(x^{(i)}_{train})\right)\right]$$

Or

fraction of test set and train set that the algo. has misclassified.

$$\hat{y} = \begin{cases} 1 \text{ if } f_{\vec{w},b}(\vec{x}^{(i)}) \geq 0.5 \\ 0 \text{ if } f_{\vec{w},b}(\vec{x}^{(i)}) < 0.5 \end{cases}$$

~~count~~ count $\hat{y} \neq y$

→ J_{test} is the fraction of the test set that has been misclassified.

→ J_{train} —"— train set —"— .

⇒ Model Selection and Training / Cross Validation / Test Sets :-

Once parameters $\vec{w}, b$ are fit to the training set, the training error $J_{train}(\vec{w}, b)$ is likely lower than the actual generalization error.

$J_{test}(\vec{w}, b)$ is better estimate of how well the model will generalize to new data compared to $J_{train}(\vec{w}, b)$.

If you want to automatically choose a model, such as decide what degree polynomial to use. follow the procedure given below :-

size	price		
2104	400	training set	$(x^{(1)}, y^{(1)})$
1600	330		⋮
2400	369		
1416	232		$(x^{(m_{train})}, y^{(m_{train})})$
3000	540		
1985	300		
1534	315	cross validation	$(x_{cv}^{(1)}, y_{cv}^{(1)})$
1427	199		$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
1380	212	test set	$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$
1494	243		

Training / Cross Validation / Test set :-

Training Error:

$$J_{train} = \frac{1}{2m_{train}} \left[\sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 \right]$$

Cross Validation Error:

$$J_{cv} = \frac{1}{2m_{cv}} \left[\sum_{i=1}^{m_{cv}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 \right]$$ (validation / dev error)

Test Error:

$$J_{test} = \frac{1}{2m_{test}} \left[\sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)} \right)^2 \right]$$

Model Selection :-

$d=1$ 1. $f_{\vec{w},b}(\vec{x}) = w_1 x + b$ $\quad w^{(1)}, b^{(1)} \rightarrow J_{cv}(w^{(1)}, b^{(1)})$

$d=2$ 2. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + b$ $\quad \rightarrow J_{cv}(w^{(2)}, b^{(2)})$

$\vdots$

$d=10$ 10. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + \cdots + w_{10} x^{10} + b \rightarrow J_{cv}(w^{(10)}, b^{(10)})$

Pick the model for which cross validation error is least.

Estimate generalization error using test the set: $J_{test}(w, b)$

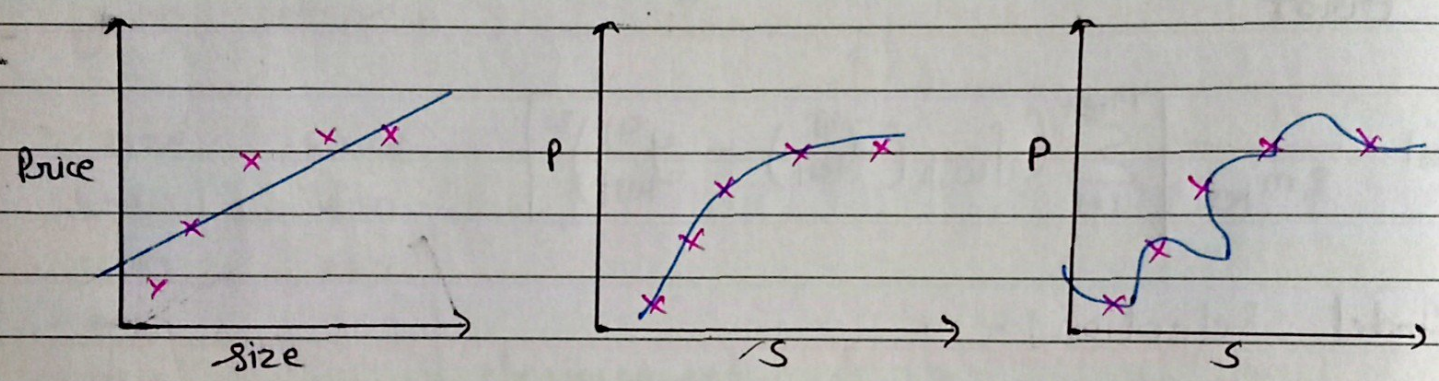
@) This procedure also works for choosing among other types of models including neural networks.

Bias and Variance :-

⇒ Diagnosing bias and variance :-

The typical workflow of developing a ML system is that you have an idea and you train the model, and you almost always find that it doesn't work as well as you wish yet.

It is found that looking at the bias and variance of learning algo. gives you very good guidance on what to try next.



$f = w_1 x + b$

High Bias
(underfit)
J_{train}, J_{cv} is high.

$f = w_1 x + w_2 x^2 + b$

"just right"
J_{train} is low
J_{cv} is low

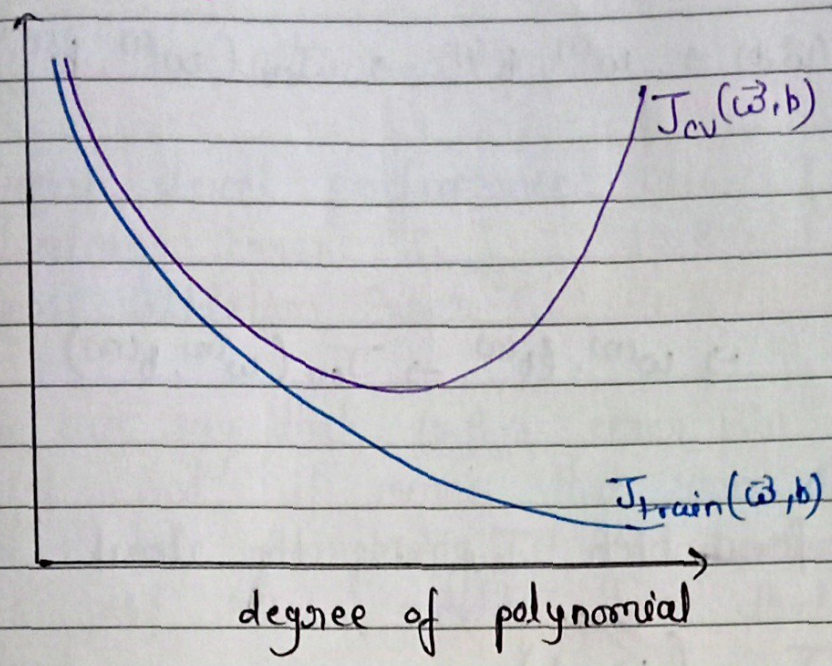
$f = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$

High Variance
(overfit)
J_{train} is low, J_{cv} is high

But, If you have many features, you can't plot f and visualize whether it's doing well as easily.

So, to find out if your algo has high bias or high variance, look at the performance of your algo on training and cross validation set.

Understanding bias and variance



High bias (underfit)
J_{train} will be high
($J_{train} \approx J_{cv}$)

High variance (overfit)
$J_{cv} >> J_{train}$
(J_{train} may be low)

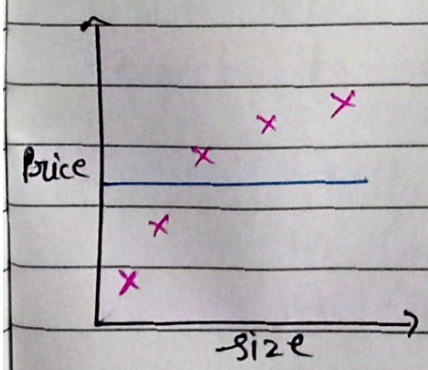
High bias and high variance
J_{train} will be high
and $J_{cv} >> J_{train}$

⇒ Regularization and bias / variance :-

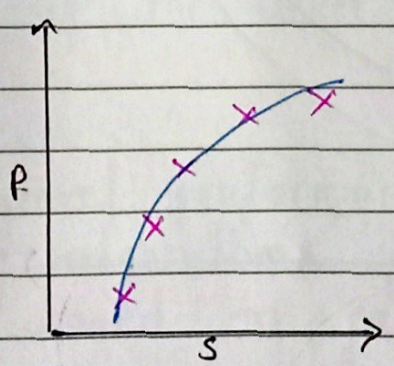
Let's take a look at how regularization affects the bias and variance.

model : $f(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$

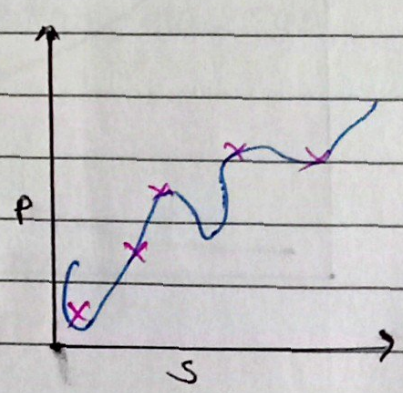
$$J = \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(\bar{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$



large λ
high bias
($\lambda = 10,000$, $w_1 \approx 0$, $w_2 \approx 0$)
$f = b$
J_{train} is large



intermediate λ
J_{train}, J_{cv} are small



small λ
high variance
($\lambda = 0$)
J_{train} is small
$J_{cv} >> J_{train}$

Choosing the regularization parameter λ

1. Try $\lambda = 0$ $\rightarrow \min J(\vec{w}, b) \rightarrow w^{(1)}, b^{(1)} \rightarrow J_{cv}(w^{(1)}, b^{(1)})$
2. Try $\lambda = 0.01$
3. Try $\lambda = 0.02$
4. Try $\lambda = 0.04$

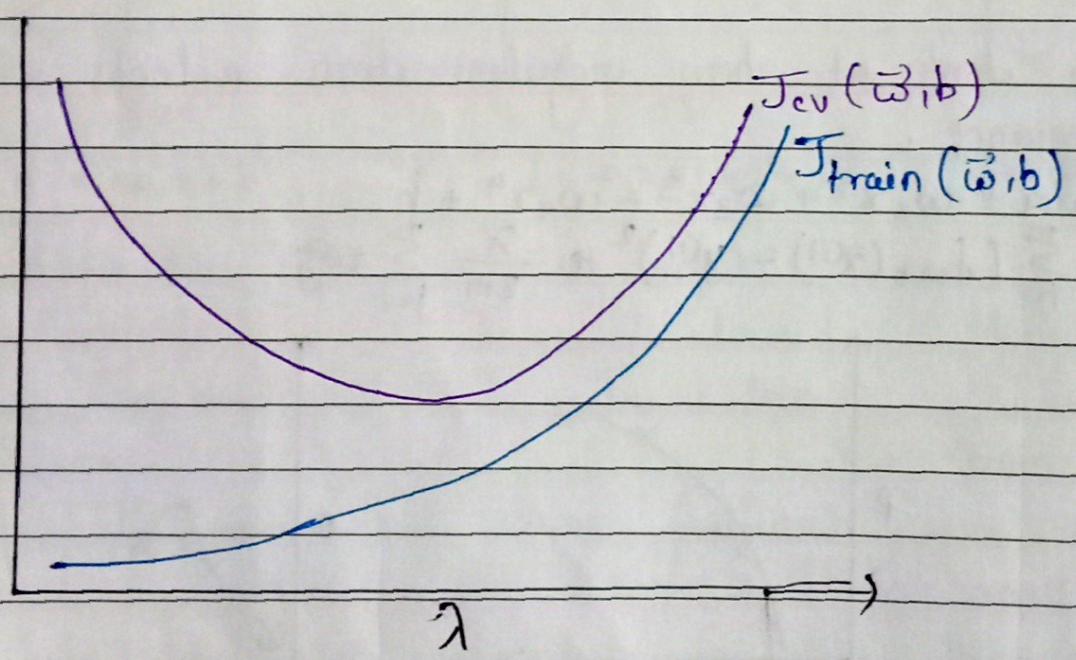
⋮

12. Try $\lambda \approx 10$ $\rightarrow w^{(12)}, b^{(12)} \rightarrow J_{cv}(w^{(12)}, b^{(12)})$

You can pick the λ for which J_{cv} is the least.

Report test error: $J_{test}(w, b)$

Bias and Variance as a function of λ



Date
Page

⇒ Establishing a baseline level of performance :-

Speech Recognition Example

Human level performance: 10.6% } 0.2%
Training Error J_{train} : 10.8% } 4.0%
Cross Validation Error J_{cv}: 14.8%

→ You can say that 10.8% error in training set is too high and it means that your model has high bias. But, while comparing it to human level performance it's just 0.2% greater than that which is pretty good.

→ So, the algorithm is doing well in training set, but cross validation error is much higher than training set, which leads us to the conclusion that your ~~algorithm~~ algorithm has high variance problem.

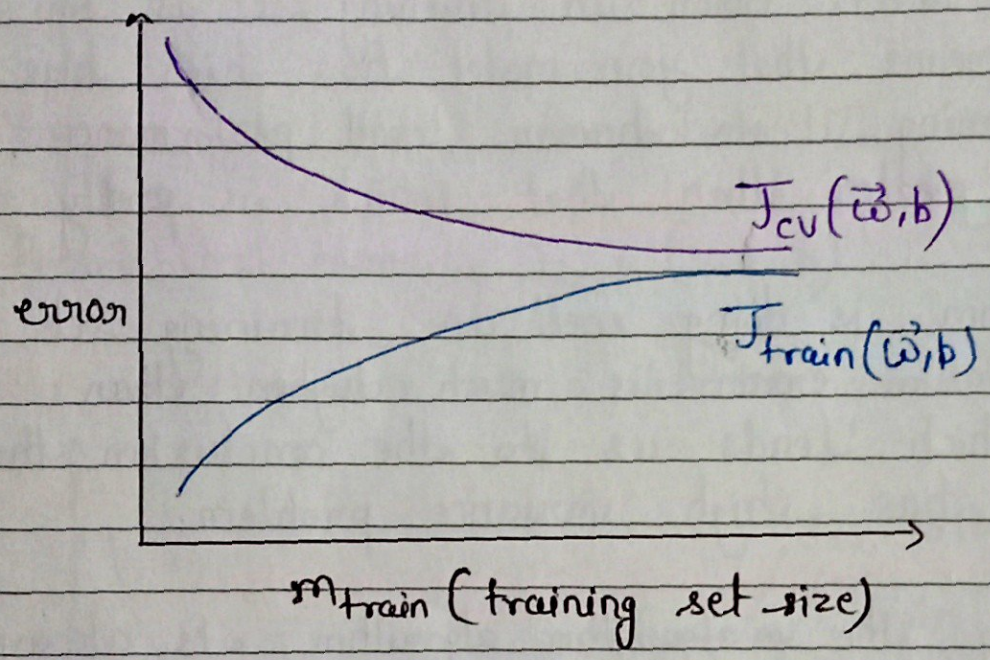
→ So, while judging the ~~algorythm~~ algorithm, it's always good to establish a baseline level of performance. It means, what is the level of error you can expect to get to?

↳ Human level performance
↳ Competing algorithms performance
↳ Guess based on experience

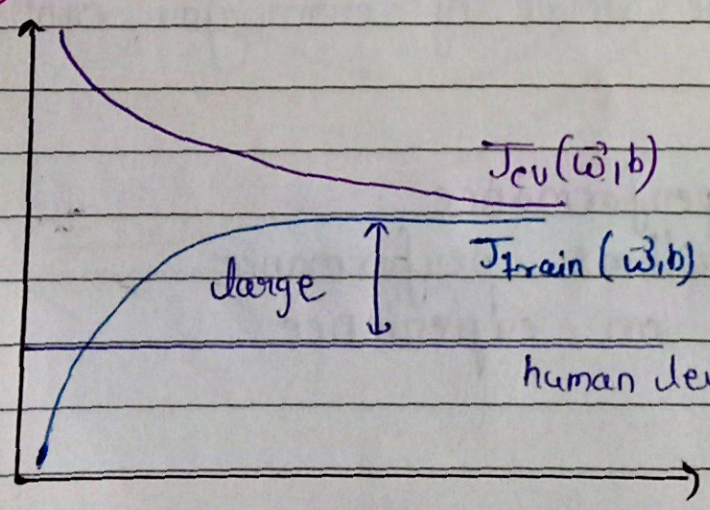
⇒ Learning Curves :-

↳ Learning curves are a way to help understand how your learning algorithm is doing as a function of the amount of experience it has, whereby experience, I mean the for example, the number of training examples it has.

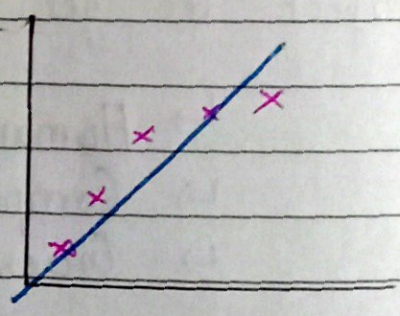
For model = $w_1 x + w_2 x^2 + b$



→ High Bias



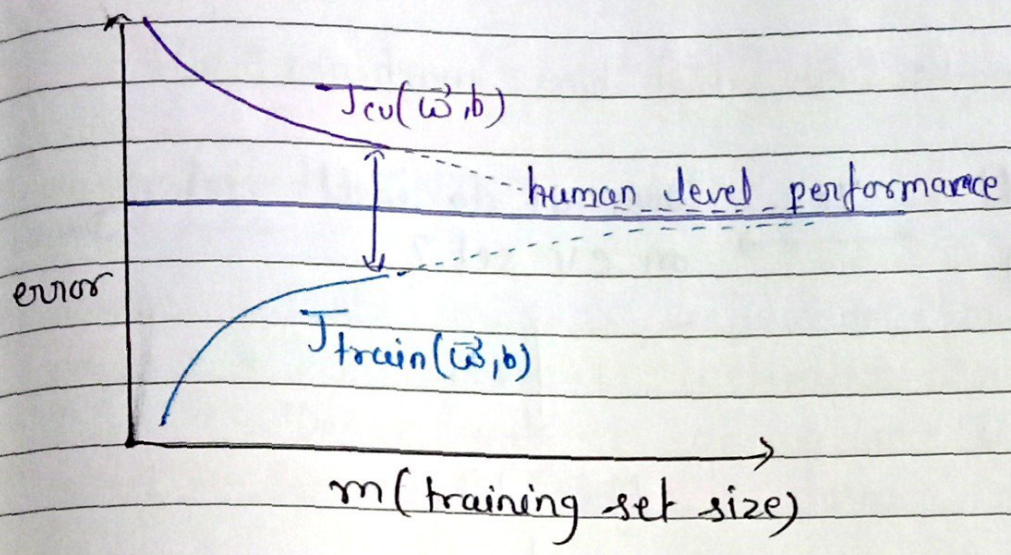
$f(x) = w_1 x + b$

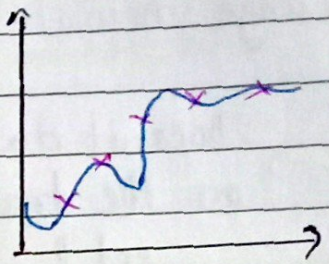


↳ If algo. has high bias, get more data is a waste of time.

High Variance

$f(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$ (with small λ)





↳ in case of high variance, increasing the training data might help.

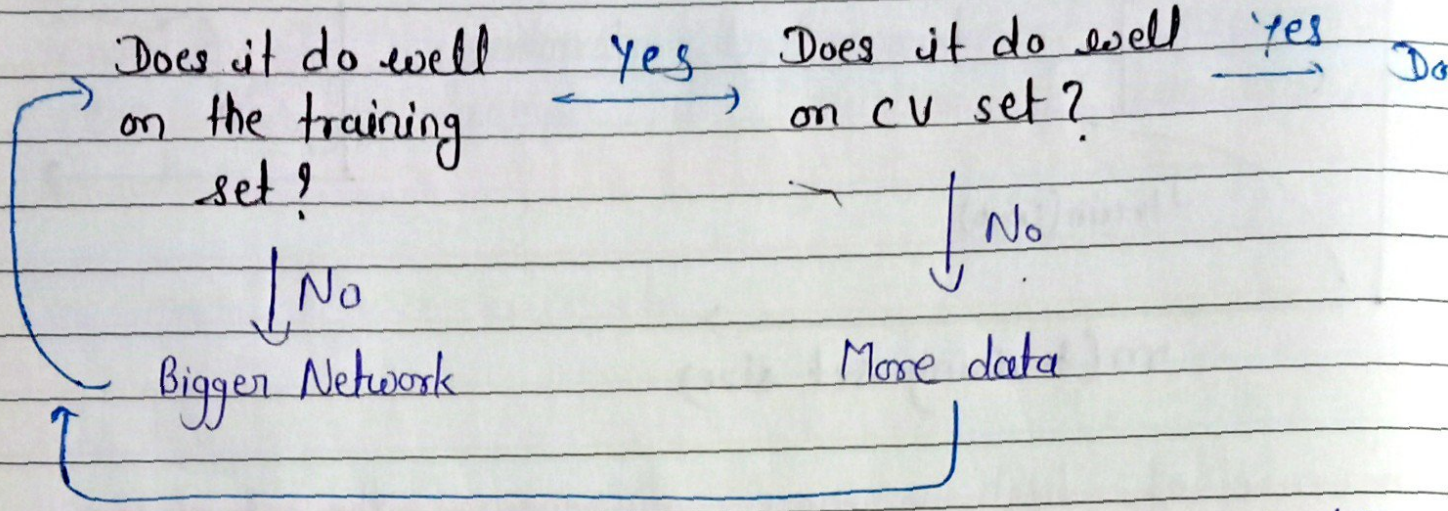
⇒ Deciding What to Try Next :-

↳ Get more traing examples	fixes high variance
↳ Try smaller set of features.	fixes high variance
↳ Try getting additional features	fixes high bias
↳ Try adding polynomial features	fixes high bias
↳ Try increasing λ.	fixes high variance
↳ Try decreasing λ	fixes high bias

"Bias and Variance are the concept that takes a short time to learn, but takes lifetime to master".

⇒ Bias / Variance and Neural Networks :-

"Large neural networks are low bias machines"



↳ It is a simple recipe that isn't always applicable.

↳ There are two limitations in this technique.

- ↳ Bigger network - Much larger neural networks are very complex and takes a lot of time in computation, which, at some point, makes them infeasible.
- ↳ More data - In this era, you have a access to large datasets. But, for some problems, you can not collect more data after some extent.

"A large neural network will usually do as well or better than a smaller one so long as regularization is choosen correctly."

But!!! How to regularize a ~~netwo~~ neural network.

Date
Page

Neural Network Regularization

$$J(W,B) = \frac{1}{m}\sum_{i=1}^{m} L(f(\vec{x}^{(i)}), y^{(i)}) + \frac{\lambda}{2m}\sum_{\text{all weights } W}(W^2)$$

unregularized MNIST model -

```
layer_1 = Dense(units=25, activation="relu")
layer_2 = Dense(units=15, activation="relu")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
```

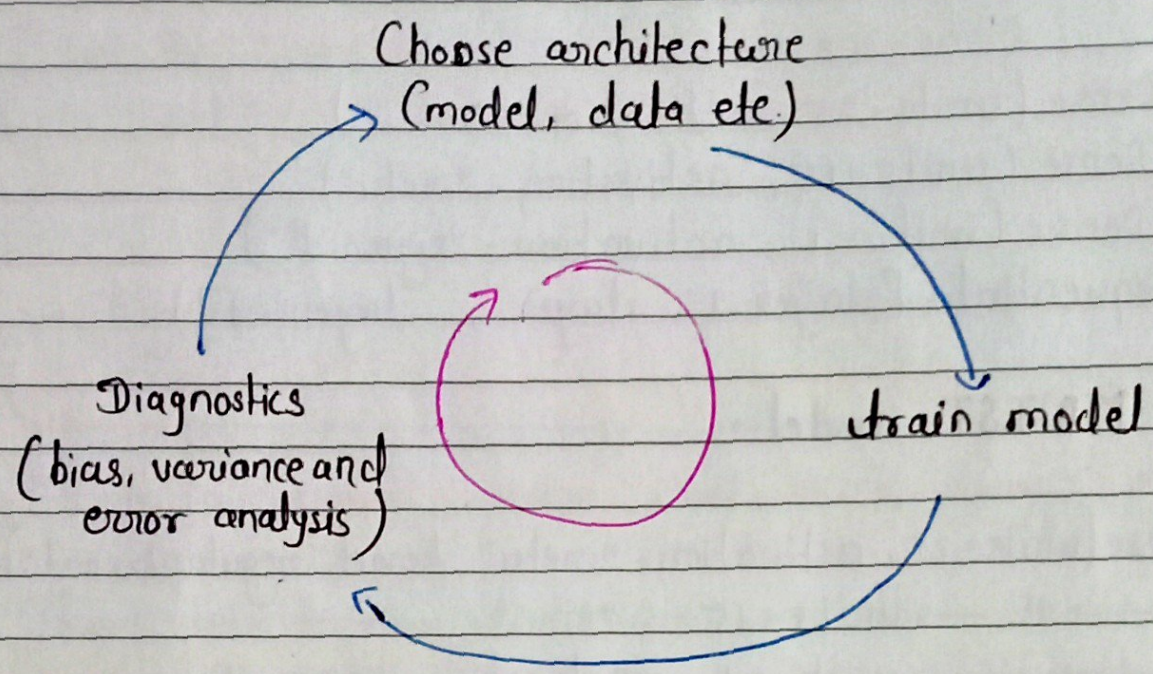
regularized MNIST model -

```
layer_1 = Dense(units=25, activation="relu", kernel_regularizer=L2(0.01))
layer_2 =  —— " —— units=15  —— " ——
layer_3 =  —— " —— units=1, activation="sigmoid" —— " ——
model = Sequential([layer_1, layer_2, layer_3])
```

λ

Machines Learning Development Process :-

⇒ Iterative loop of ML development :-



⇒ Error Analysis :-

↳ second after bias and variance for improving algo.

m_{cv} = 500 examples in cross validation set.
Algorithm misclassified 100 of them.

→ Manually examine 100 examples and categorize them based on common traits.

pharma : 21
Deliberate mis-spellings : 3
Unusual email routing : 7
Steal Password : 18
Spam message in embedded image : 5

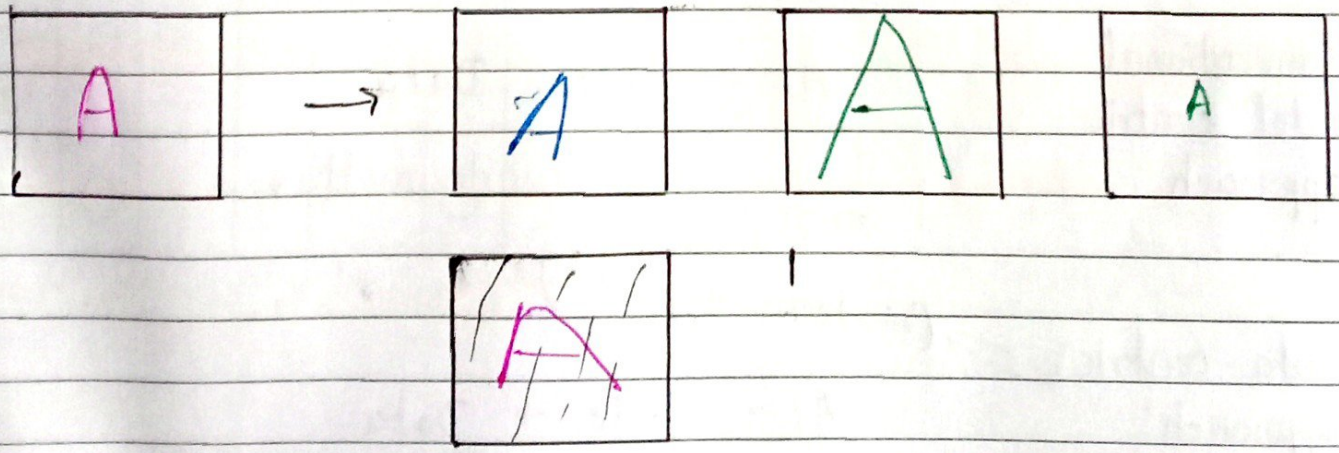
↳ after this analysis, based on the conclusion you can choose what to try next and where to put your focus on.

⇒ Adding Data :-

While creating a machine learning program, you may want to get more data of everything which is fine. But, if error analysis suggest you a subset where the problem is occuring the most, it's better to add more data of that specific domain to boost the performance of your machine learning model.

Beyond getting brand new training examples (x, y), another technique : Data Augmentation
↳ generally used for images and audio.

Augmentation = modifying an existing training examples to create a net new training example.



↳ all the above images are of 'A' but d looks different (by distortion of image)

for audio :

↳ original audio ("~~Gotel~~ what is today's weather?")

↳ original + noisy bg (crowd)

↳ original + noisy bg (car)

↳ original + audio on bad cellphone connection

↳ There is one another technique called Data Synthesis to enlarge your training set.

Data Synthesis = Using ~~artifiti~~ artificial data inputs to create a new training example.

↳ You can use text editor on your computer to create raw data by writing random letter with diff. fonts and contrast.

Approaches :-

↳ Conventional model-centric approach :

AI = Code + Data

work on this

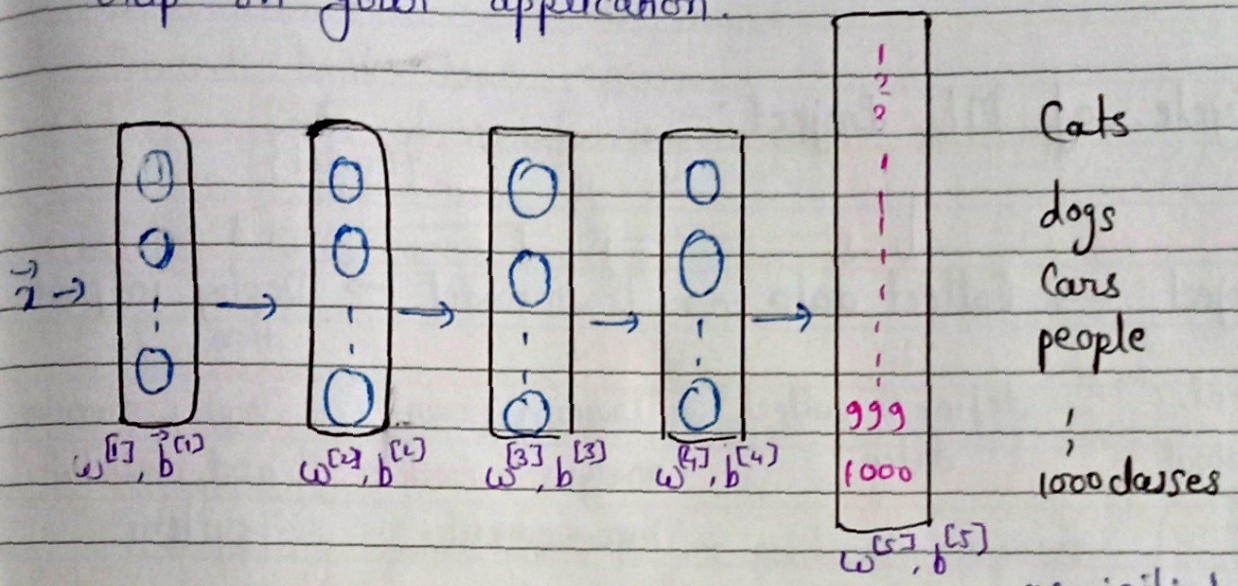
↳ Data-centric approach : (modern approach)

AI = Code + Data

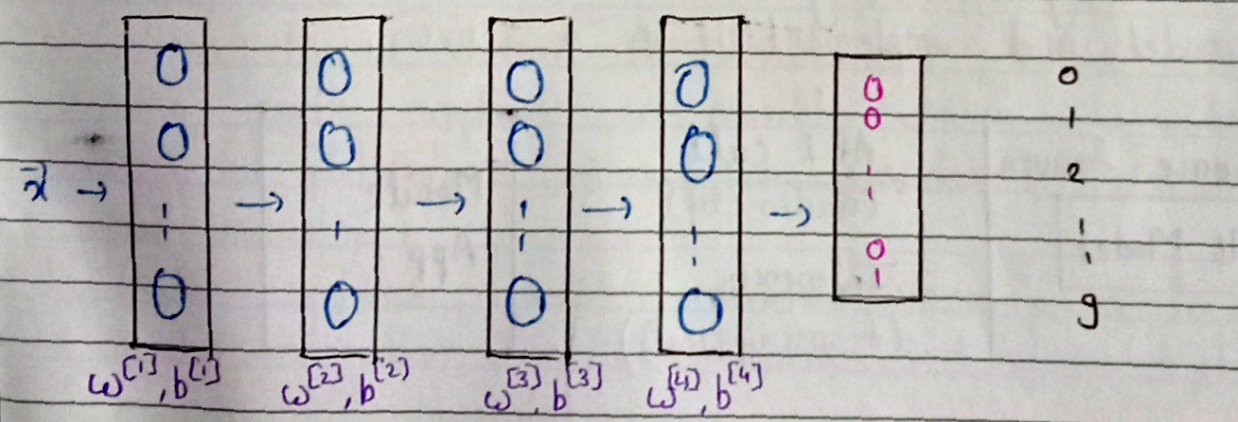
work on this

⇒ Transfer Learning : Using Data from a different task :-

→ For an application where you don't have that much data, transfer learning is wonderful technique that lets you use data from a different task to help on your application.



now these same parameters can be used as initial parameters for other recognition algorithm to train it.



option 1 : only train output layers ~~p~~ parameters.
↳ for very small training set

option 2 : train all parameters
↳ for a bit larger training set

⇒ Full Cycle of ML Project :-

{Audit}

Scope Project → Collect data → Train model → Deploy in production

↓ Define project, what is it about (overview)

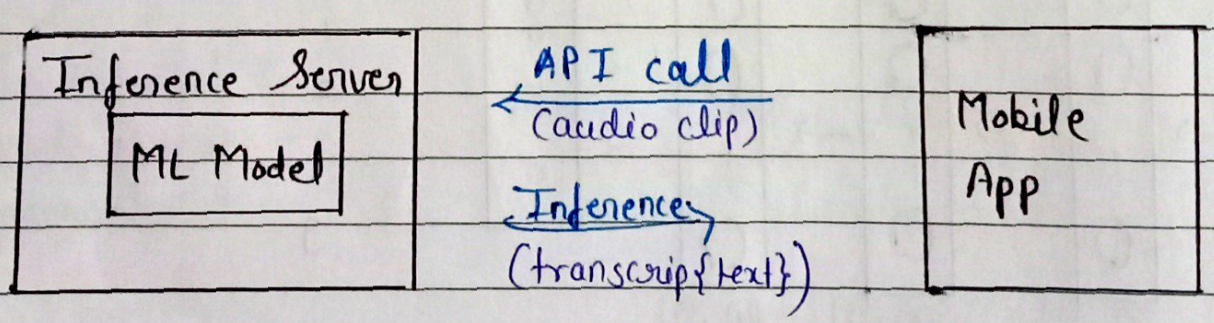
↓ define & collect data

↓ Training, error analysis & iterative improvement

↓ Deploy, monitor and maintain system

→ Deployment :-

↳ A common way to deploy the model would be to take your ML model and implement it in a server.



↳ Software engineering may be needed for :
↳ Ensure reliable and efficient prediction
↳ Scaling
↳ Logging
↳ System Monitoring
↳ Model Updates

} MLOps : machine learning operations

Date
Page

⇒ Fairness, Bias, and Ethics :-

→ Bias

↳ Hiring tool that discriminates against women.
↳ Facial recognition system matching dark skinned individuals to criminal mugshots.
↳ Biased bank loan approvals.
↳ Toxic effects of reinforcing negative stereotypes.

} examples

→ Adverse use cases

↳ Deepfakes
↳ Spreading toxic speech through optimizing for engagement.
↳ Generating fake content for political purposes.
↳ Using ML to build harmful products

} examples

→ Guidlines

↳ Get a diverse team to brainstorm things that might go wrong, with emphasis on possible harm to vulnerable groups.
↳ Carry out literature search on standards / guidelines for your industry.
↳ Audit systems against possible harm prior to deployment.
↳ Develop mitigation plan (if applicable), and after deployment, monitor for possible harm.

Decision Trees :-

⇒ Decision Tree Model :-

↳ To understand how decision trees work, lets take an example :-

Cat Classification Example :-

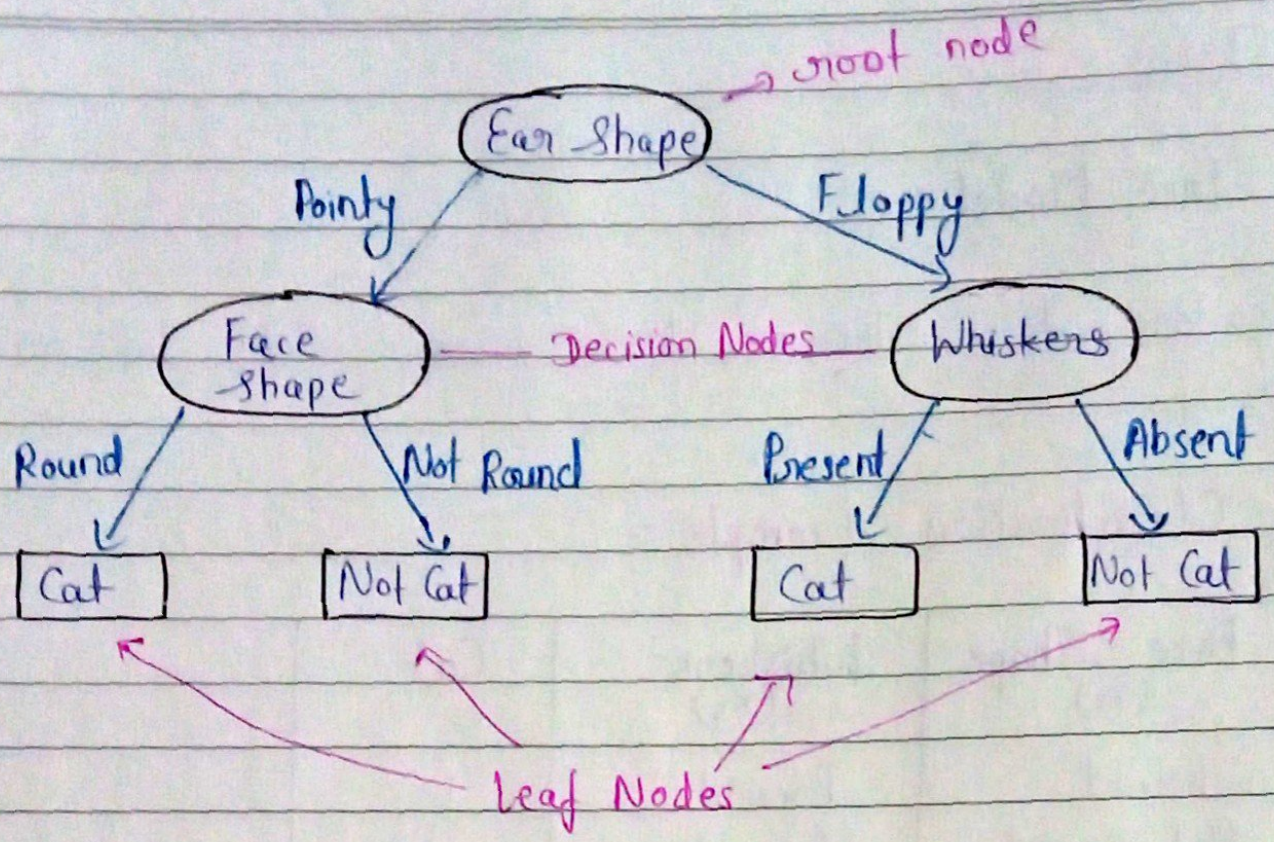
Ear Shape (x_1)	Face Shape (x_2)	Whiskers (x_3)	Cat
Pointy	Round	Present	1
Floppy	Not Round	Present	1
F	Round	Absent	0
P	NR	P	0
P	R	P	1
P	R	A	1
F	NR	A	0
P	R	A	1
F	R	A	0
F	R	A	0

X ↳ categorical values (discrete)

Y ↳ Binary

↳ Here is an example of a model that you might get after training a decision tree learning algorithm on the dataset you just saw.

⟶



⇒ Learning Process :-

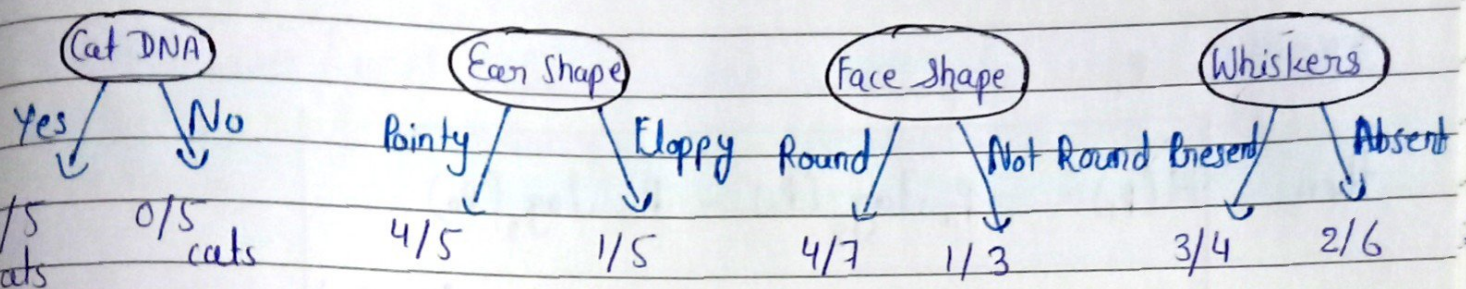
↳ In the decision tree, first we have to choose a feature for root node, according to which we splits the training examples by their value of that specific feature.

↳ Then, the same process is repeated for ~~for~~ the decision nodes until we get our answer.

↳ There are some key decisions we need to make in order to create the algorithm.

↳ Decision 1 : How to choose what feature to split on at each node?

↳ (we need max purity)



↳ Decision 2: When do you stop splitting?

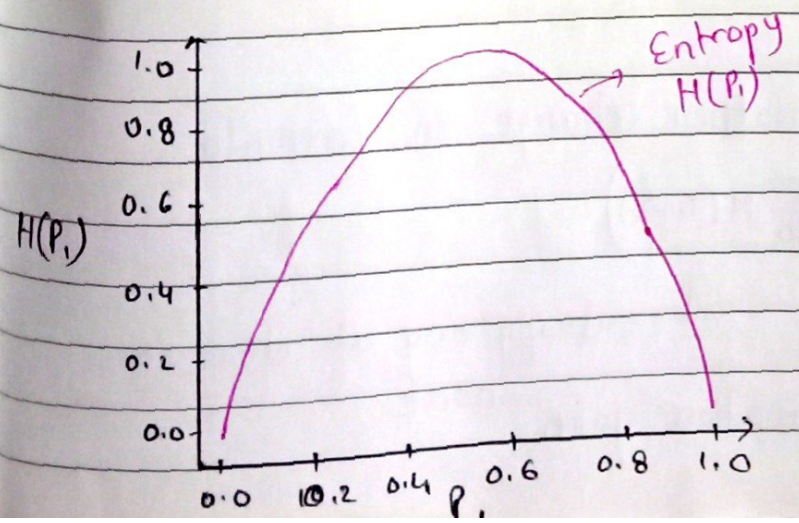
- ↳ When a node is 100% one class.
- ↳ When splitting a node will result in the tree exceeding a maximum depth.
- ↳ When improvements in purity score are below a threshold
- ↳ When number of examples in a node is below a threshold

Decision Tree Learning :-

⇒ Measuring Purity :-

→ Entropy as a measure of impurity

P_1 = fraction of examples that cats (desired category)



→ Entropy is max when $P_1 = 0.5$

→ Entropy is min for $P_1 = 0$ or 1

$P_0 = 1 - P_1$

then,
$$\boxed{\begin{aligned} H(P_1) &= -P_1 \log_2(P_1) - P_0 \log_2(P_0) \\ &= -P_1 \log_2(P_1) - (1-P_1)\log_2(1-P_1) \end{aligned}}$$

↳ $\log_2$ makes the peak of the function 1.
↳ Note : "$0 \log(0)$" = 0

⇒ Choosing a split : Information Gain :-
↳ reduction of impurity

root
$P_1 = \frac{5}{10} = 0.5$, $H(P_1) = 1$

Ear Shape
- Pointy: $P_1 = \frac{4}{5} = 0.8$, $H(0.8) = 0.72$
- Floppy: $P_1 = \frac{1}{5} = 0.2$, $H(0.2) = 0.72$

Face Shape
- Round: $P_1 = 4/7 = 0.57$, $H(0.57) = 0.99$
- Not Round: $P_1 = \frac{1}{3} = 0.33$, $H(0.33) = 0.92$

Whiskers
- Pre: $P_1 = 3/4 = 0.75$, $H(0.75) = 0.81$
- Absent: $P_1 = \frac{2}{6} = 0.33$, $H(0.33) = 0.92$

$\underbrace{\left(\frac{5}{10} H(0.8) + \frac{5}{10} H(0.2)\right)}_{\text{average}}$ $\left(\underbrace{\frac{7}{10} H(0.57)}_{w_{left}} + \underbrace{\frac{3}{10} H(0.33)}_{w_{right}}\right)$ $\left(\frac{4}{10} H(0.75) + \frac{6}{10} H(0.33)\right)$

↳ We will choose the feature by calculating weighted average and picking up the ~~minimum~~ one with minimum average.

↳ Here, we can make another change in formula.

$$\rightarrow H(0.5) - \left(\frac{5}{10} H(0.8) + \frac{5}{10} H(0.2)\right)$$

↳ entropy at root node
↳ entropy avg at decision node
↳ This is called information gain

↳ Now, we will choose the spit for which the reduction of impurity or information gain is maximum.

$$\boxed{IG = H(p_1^{root}) - \left(\omega^{left} H(p_1^{left}) + \omega^{right} H(p_1^{right})\right)}$$

⇒ Putting it together :-

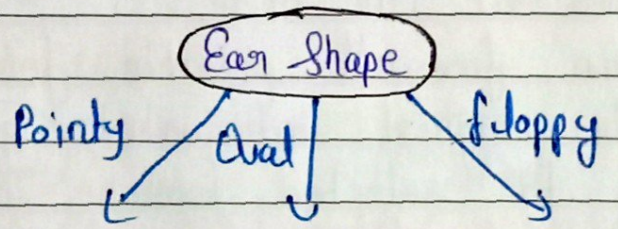
↳ Start with all examples at the node.
↳ Calculate information gain for all possible features, and pick the one with the highest information gain.
↳ Split dataset according to selected feature, and create left and right branches of the tree.
↳ keep repeating ~~spitt~~ splitting process until ~~st~~ stoping criteria is met:
  ↳ when a node is 100% one class.
  ↳ when ~~exceeding~~ splitting a node will result in the tree exceeding a max depth.
  ↳ IG from additional splits is less than threshold
  ↳ When no. of ex. in a node is below threshold.

⇒ Using One-Hot Encoding of Categorical Features:-

↳ What if we have features that can take more than two discrete values?? Use One-Hot Encoding.

↳ Let's take the previous example with a change. The "Ear Shape" feature can now take three possible values: Pointy, Floppy and Oval.

↳ This means that when you split these features, you have a ~~three like~~ tree like this:



↳ But, rather than using features which can have 3 possible values, we are going to create 3 new feature with binary values.

Pointy ear	Floppy ear	Oval ear	Face Shape	Whiskers	Cat
1	0	0	Round	Present	1
0	0	1	Not Round	Present	1
0	0	1	Round	Absent	0
1	0	0	Not round	Present	0
0	0	1	Round	Present	1

Date _____
Page _____

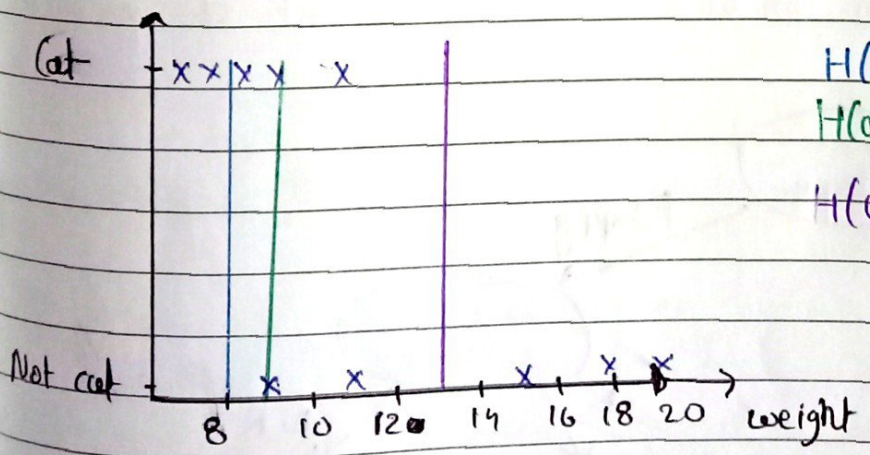
One-Hot Encoding : If a categorical feature can take on k values, create k binary features (0 or 1 values)

↳ You can also use one-hot encoding for neural networks or any other models.

⇒ Continuous Valued Features :-

Ear Shape	Face Shape	Whiskers	Weight (lbs)	Cat
Pointy	Round	Present	7.2	1
Floppy	Not round	Present	8.8	1
Floppy	Round	Absent	15	0
⋮	⋮	⋮	⋮	⋮
Floppy	Round	Absent	20	0

↳ Here for weight, we would choose a threshold value x for which the information gain is highest.

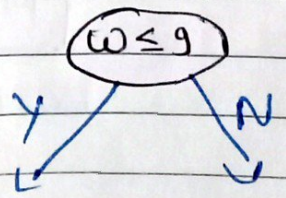


$$H(0.5) - \left(\frac{2}{10} H\left(\frac{2}{2}\right) + \frac{8}{10} H\left(\frac{3}{8}\right)\right) = 0.24$$

$$H(0.5) - \left(\frac{4}{10} H\left(\frac{4}{4}\right) + \frac{6}{10} H\left(\frac{1}{6}\right)\right) = 0.61$$

$$H(0.5) - \left(\frac{7}{10} H\left(\frac{5}{7}\right) + \frac{3}{10} H\left(\frac{0}{3}\right)\right) = 0.40$$

↳ here, $x = 9$



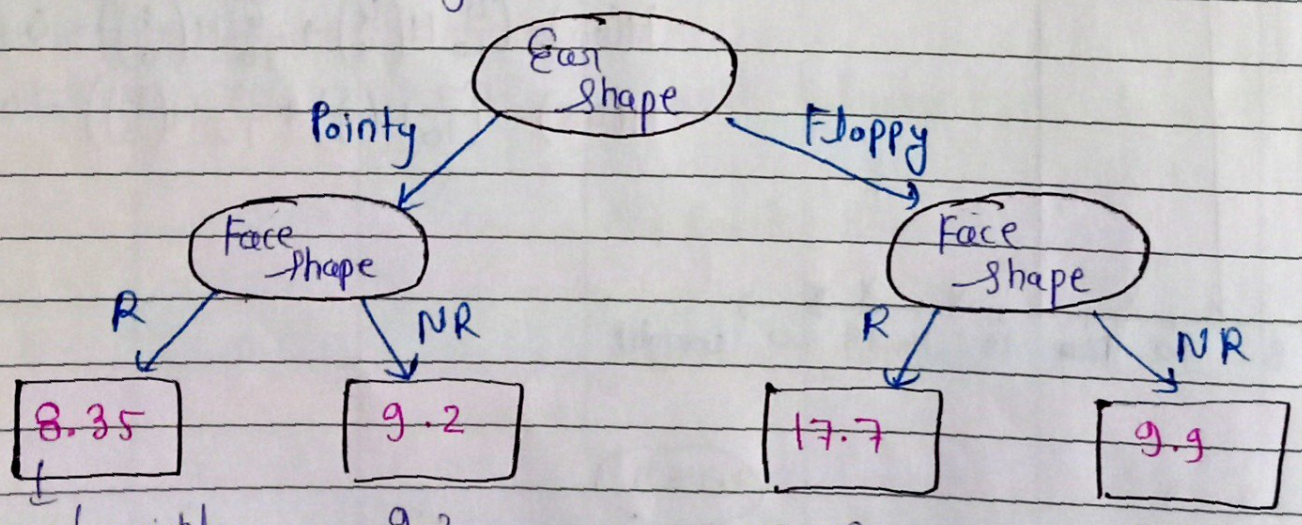
⇒ Regression Trees :-

↳ A generalization of decision trees which can predict ~~no~~ numbers rather than just categorical values.

Ear Shape	Face Shape	Whiskers	Weight (lbs.)
Pointy	Round	Present	7.2
Floppy	Not Round	P	8.8
F	R	Absent	15
P	N	P	9.2
P	R	P	8.4
P	R	A	7.6
F	N	A	11
P	R	A	10.2
F	R	A	18
F	R	A	20

X Y

↳ This is a regression problem because we want to predict a number (weight).



avg of weights 7.2, 8.4, 7.6, 10.2

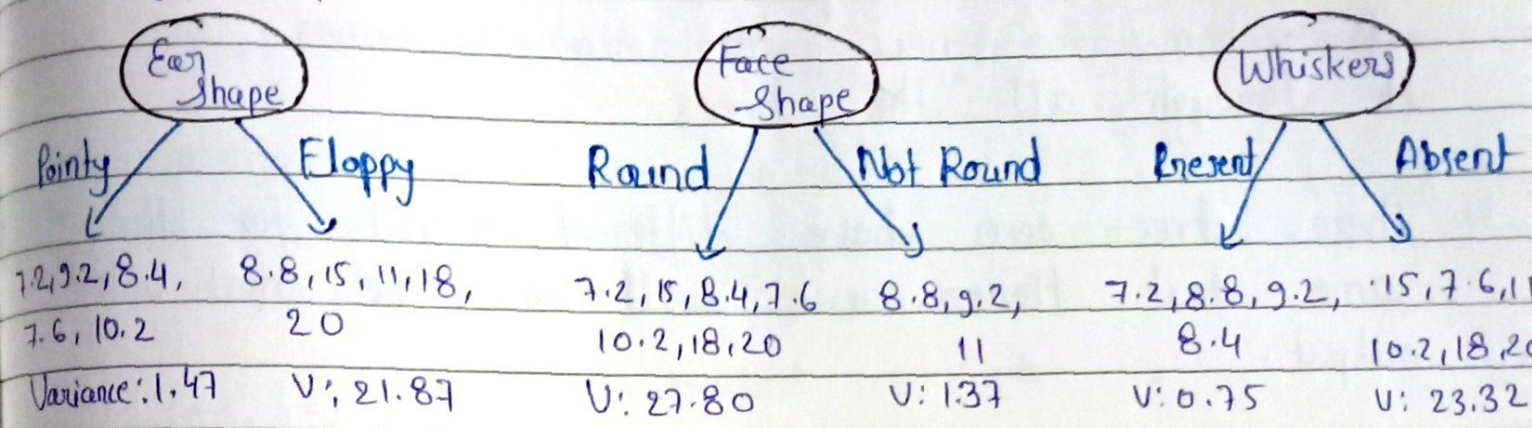
9.2

15, 18, 20

8.8, 11

→ Choosing A Spit

V: 20.51



↳ Here we don't calculate information gain. Instead, we would calculate variance and pick the feature with minimum value of ~~variance~~ reduction in variance.

$$20.51 - \left(\underbrace{\frac{5}{10}}_{w_{left}} \times 1.47 + \underbrace{\frac{5}{10}}_{w_{righ}} \times 21.87\right) = 8.84 \quad \text{(for ear shape)}$$

↳ This technique is followed recursively until we no longer need to split further.

Tree Ensembles :-

⇒ Using Multiple Decision Trees :-

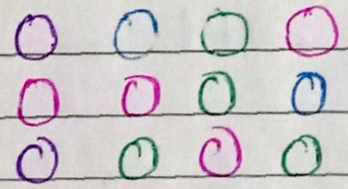
↳ One of the weaknesses of using a single decision tree is that, that decision tree can be highly sensitive to small changes in the data. One solution to make the ~~arr~~ tree less sensitive or more robust is to build not one decision tree, but to build a lot of decision trees, and we call that a tree ensemble.

↳ We will create multiple trees and train them. So when, a new test example comes, we run it through all the trees.

↳ These trees can have different outputs for the same test. Hence, we will pick the most voted output.

⇒ Sampling with replacement :-

↳ Imagine having a bag with 4 balls of different colors inside it. If we take out one ball at a time and put it back into the bag, doing the same action will lead us to some patterns of colors.



} here, some of the colors are repeated.

↳ Following the same procedure with training set of decision tree will give us a new training set with random and probably some repeated values.

Date
Page

⇒ Random Forest Algorithm :-

↳ works better than single decision tree algorithm.

Generating a tree sample :-

↳ Given training set of size m

↳ For b = 1 to B : → around 100 (64, 128 ...)

→ Use sampling with replacement to create a new training set of size m.

→ Train a decision tree on the new dataset.

↳ Doing so will give you a decision tree.

↳ Having done with this once, we would then go and repeat this again.

↳ This will give us 'B' different decision trees.

↳ With these different decision tree, you get an ~~op~~ output ~~with~~ of most voted value.

↳ This is also called Bagged Decision tree.

↳ But, sometimes we get very similar tree which is not very desirable thing to happen. To overcome this, we use Random Forest Trees.

→ Randomizing the feature choice :-

At each node, when choosing a feature to use to split, if n features are available, pick a random subset of $k < n$ features and allow the algorithm to only choose from that subset of features.

$\boxed{k = \sqrt{n}}$ → for larger n

⇒ XGBoost :-

↳ most commonly used.

→ Boosted Trees Intuition :

↳ Given training set of size m

↳ For b = 1 to B :

↳ Use sampling with replacement to create a new training set of size m. But instead of picking from all examples with equal (1/m) probability, make it more likely to pick misclassified examples from previously trained trees

↳ Train a decision tree on new dataset.

Date
Page

XGBoost (eXtreme Gradient Boosting):

- Open Source implementation of boosted trees.
- Fast efficient implementation
- Good choice of default splitting criteria and criteria for when to stop splitting
- Built in regularization to prevent overfitting
- Highly competitive algorithm for machine learning

Using XGBoost

```
from xgboost import XGBClassifier/XGBRegressor

model = XGBClassifier()/XGBRegressor()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

When to use decision trees :-

Decision Trees and Tree Ensembles :

- works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast
- Small decision trees may be human iterable

Neural Networks :

- works well on all types of data, including tabular and unstructured data.
- Maybe slower than decision trees
- works with transfer learning
- when building a system of multiple models working together, it might be ~~easing~~ easier to string together multiple neural networks

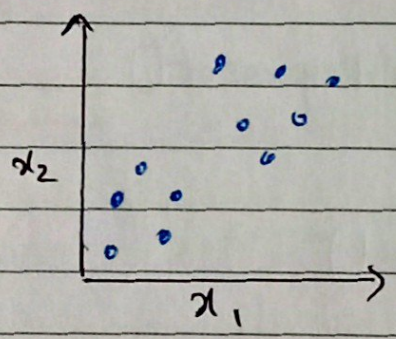
Unsupervised Learning, Recommenders, Reinforcement Learning

Clustering :-

↳ A clustering algorithm looks at a number of data points and automatically finds data points that are related or similar to each other.

↳ In unsupervised learning, you are given a dataset with just x but not the labels or target labels y.

↳ Since, there is no 'y', we can not tell the algorithm what is the right answer we want to predict. Here, we want the algorithm to find similarities in data.



→ Applications of clustering :-

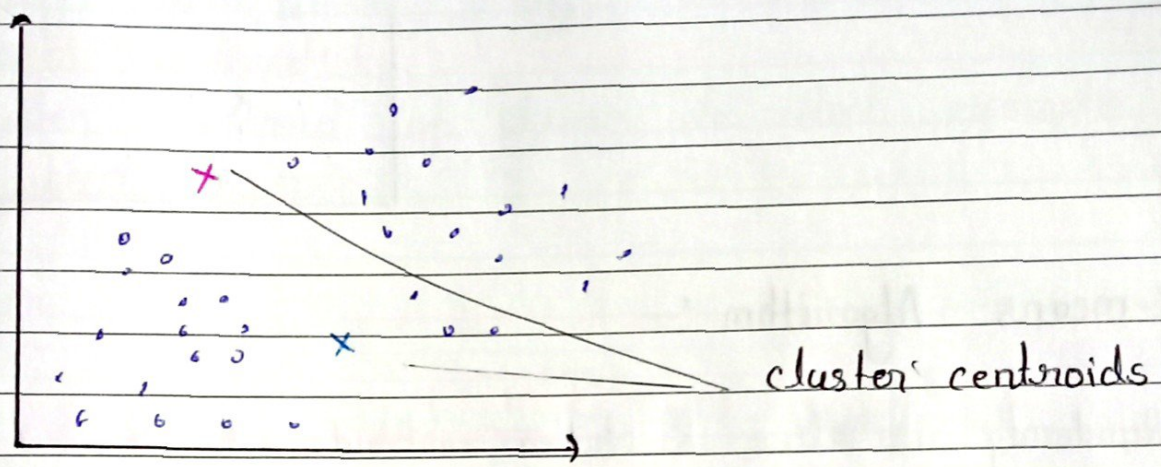
↳ Grouping similar news

↳ DNA analysis

↳ Astronomical data analysis

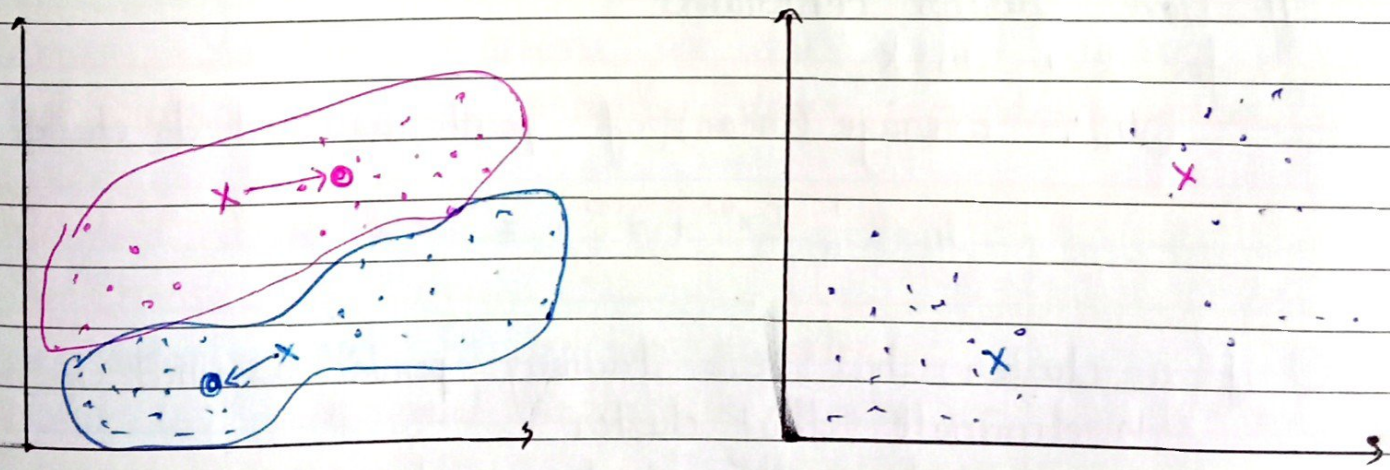
⇒ K-means intuition :-

↳ Lets take a look at what the k-means algorithm does.



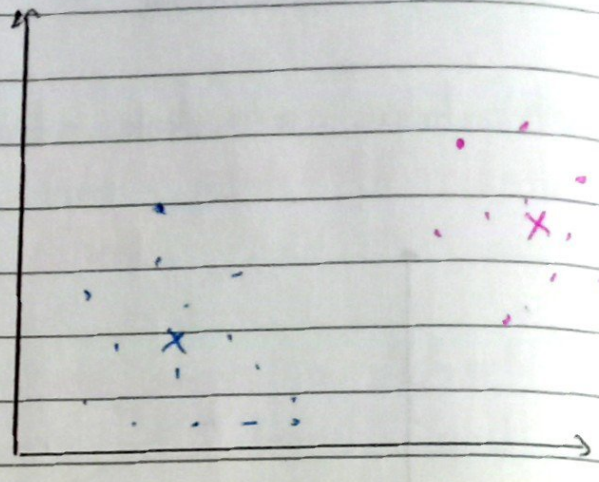
↳ Here, we have an unlabeled dataset. To apply the algorithm, we will first assume two points on the dataset set and call them "cluster centroids".

↳ Then, we will see which datapoint is close to which cluster centroids.



↳ Now, we will find the mean of all the datapoints that are close to a cluster centroid and then change the position of that centroid to that mean position.

↳ After repeating same procedure for several times, we get two separated clusters



⇒ K-means Algorithm :-

↳ Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$.
↳ these are vectors with the same dimentions as datapoints $x^{(1)}, x^{(2)} \ldots$

↳ Repeat {

Assign points to cluster centroids

for $i = 1$ to m

$c^{(i)} := \text{index (from 1 to k) of cluster centroid closest to } x^{(i)}$

$\min_k \| x^{(i)} - \mu_k \|^2$

Move cluster centroids

for $k = 1$ to K

$\mu_k := \text{average (mean) of points assigned to cluster } k$

$$\mu_1 = \frac{1}{4} [x^{(1)} + x^{(5)} + x^{(6)} + x^{(12)}]$$

@ if a cluster has zero training points assigned to it.
↳ eliminate that cluster
↳ reinitialize the clusters centroids

}

↳ K-means works fine and give useful results even if the data is not well-separated.

Date
Page

⇒ Optimization Objective :-

notations :-

$c^{(i)}$ = index of cluster (1, 2, ..., K) to which example $x^{(i)}$ is currently assigned

μ_k = cluster centroid k

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

- Cost Function :-

$$\boxed{J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2}$$

↳ it is called distortion function.

↳ we try to minimize this cost function.

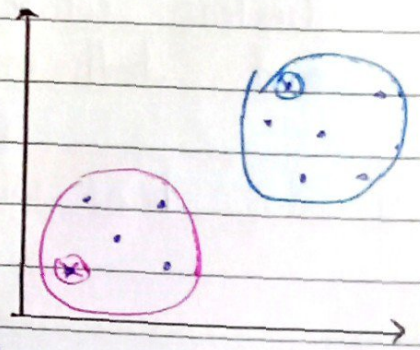
⇒ Initializing K-means :-

↳ The very ~~fist~~ first step of the K-means algo. was to choose random locations as the initial guesses. But how do you actually take random guess?

Random initialization :-

↳ Choose $K < m$

↳ Randomly pick K training examples. Set $\mu_1, \ldots, \mu_K$ equal to these K examples.



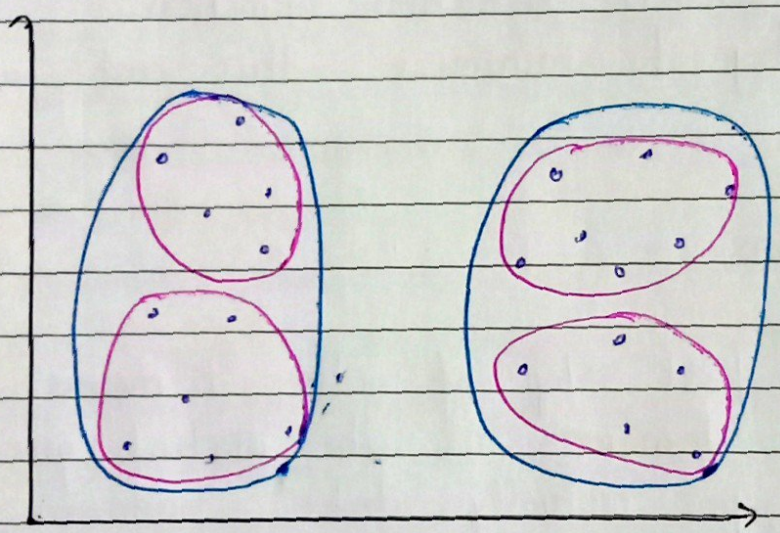
↳ But, in this way you can end up getting different clusters with different initialization. One way to tackle this problem is to run the algo. multiple times and ~~to~~ choose the best clusters with minimum cost function

for i = 1 to 100 {
    Randomly initialize K-means.
    Run K-means. Get $c^{(1)}, \ldots c^{(m)}, \mu_1, \ldots, \mu_K$
    Compute cost function (distortion).
}

↳ Pick set of clusters that gave lowest cost J

⇒ Choosing the number of clusters :-

↳ It's ambiguous to decide the value of K or how ~~may~~ many clusters are there in the dataset.

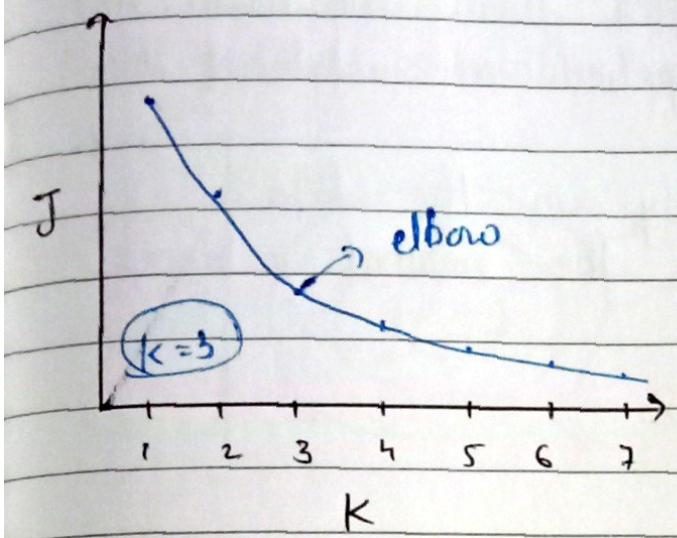


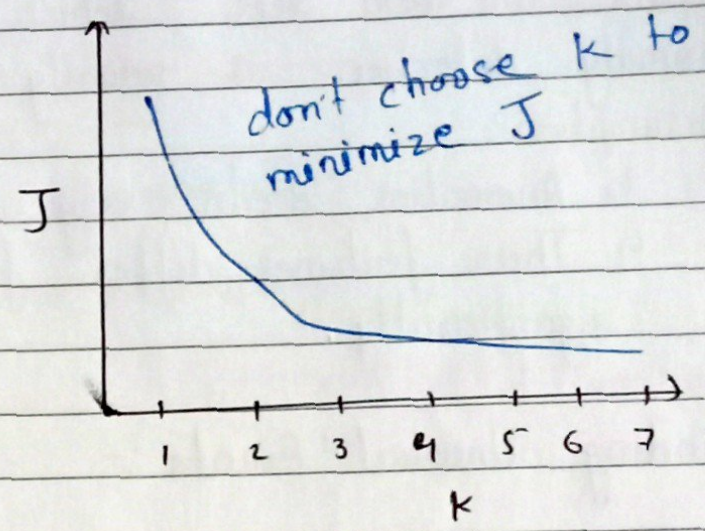
↳ For the above example, one can say, there are 2 clusters while other can say there are 4 clusters and both are right.

↳ So, how will you choose the number of clusters.

Elbow Method :-

- In this method, you run the algo. with different values of K and plot the cost function.





- Although this method is not very recommended.

- Often, you want to get clusters for some later (downstream) purpose.
- Evaluate K-means based on how well it performs on that later purpose.

Anomaly Detection :-

↳ It is a process of identifying unexpected items or events in data sets, which differ from the norm. And anomaly detection is usually applied on unlabeled data.

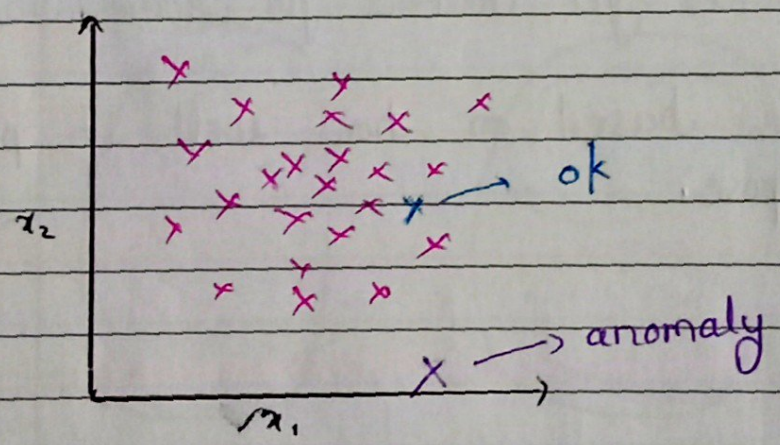
Assumption:

↳ Anomalies occur very rarely in the data.

↳ Their features differ from the normal instances significantly.

⇒ Finding Unusual Events :-

↳ At first we will give the training data to our model to learn what value of input features are normal.



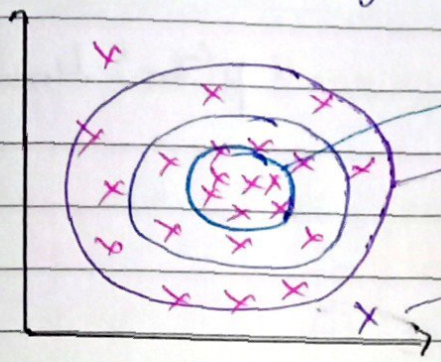
↳ Then, if our test data is close to normal features, it's not an anomaly otherwise it is.

↳ ~~We calculate~~ The most common way to carry out anomaly detection is through density estimation.

Density Estimation :-

Dataset: $\{x^{(1)}, x^{(2)}, \ldots x^{(m)}\}$

Model $p(x)$ = probability of x being seen in the dataset



high probability

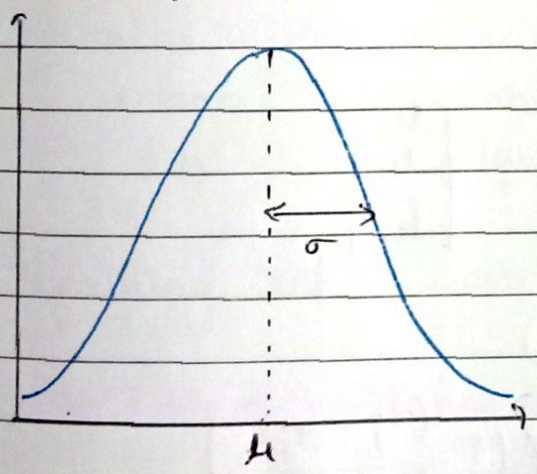
low probability

if $p(x) < \varepsilon$ (small number)
↳ it's an anomaly.

⇒ Gaussian (Normal) Distribution :-
↳ also called bell shaped distribution

↳ say x is a number.

↳ Probability is of x is determined by a gaussian with mean μ, variance σ^2,



$$\boxed{p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}}$$

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2$$

↳ some people suggest to use $\left(\frac{1}{m-1}\right)$ instead of $\left(\frac{1}{m}\right)$

Date
Page

⇒ Anomaly Detection Algorithm :-

Training set: $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(m)}\}$

Each example $\vec{x}^{(i)}$ has n features

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

$$\boxed{p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)}$$

$$\hookrightarrow = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

Algo.

↳ Choose n features x_i that you think you might be indicative of anomalus examples.

↳ Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

i) $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$ $\qquad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$

ii) vectorized :

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^{m} \vec{x}^{(i)} \implies \vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}$$

↳ Given new example x, compute p(x)

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\,\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

↳ Anomaly if $p(x) < \varepsilon$

Date
Page

⇒ Developing and Evaluting an Anomaly Detection System:-

→ real number evaluation :-

↳ When developing a learning algorithm (choosing features, etc), making decision is much easier if we have a way of evaluating our learning algorithm.

↳ Assume we have some labeled data,

$y = 1$, for anomalous examples

$y = 0$, for non-anomalous examples

↳ Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ {unlabeled} (assume non-anomalous)

$y = 0$, for all training examples.

↳ CV set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

Test set: $(x_{test}^{(1)}, y_{test}^{(1)}), \ldots, (x_{test}^{(m_t)}, y_{test}^{(m_t)})$

} few $y = 1$, mostly $y = 0$

→ Aircraft engines ~~motin~~ monitoring examples:-

10000 = good engines

20 = flawed engines

Training set: 6000 good engines {train algo. on this set}

CV: 2000 good engines ($y=0$) , 10 anomalous ($y=1$)

↳ tune ε , x_j

Test: 2000 good engines ($y=0$), 10 anomalous ($y=1$)

↳ If you don't have enough data, eliminate the test data.

→ Algorithm Evaluation :-

↳ Fit model $p(x)$ on training set $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

↳ On CV / test examples x, predict

$$y = \begin{cases} 1 & ; p(x) < \varepsilon \\ 0 & ; p(x) \geq \varepsilon \end{cases}$$

↳ Possible evaluation metrices:

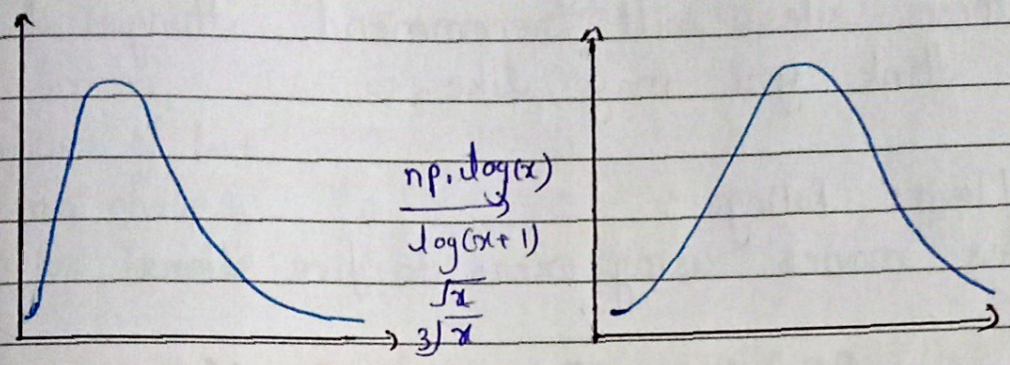
- True positive, false positive, false negative, true negative
- Precision / Recall
- F_1-score

⇒ Anamoly Detection vs Supervised Learning :-

Anamoly Detection	Supervised Learning
↳ very small pos. ex. ($y=1$), Large neg. ex. ($y=0$)	→ Large no. of pos. and neg. examples.
↳ many different types of anomalies. Hard for any algo. to learn from pos. ex. what anomalies look like; future anomalies may be completely different.	→ Enough pos. ex. for algo. to get a sense of what pos. ex. are like, future pos. ex. likely to be similar to ones in training set.
→ Fraud detection → Manufacturing: Finding new previously unseen defects → Monitoring machines in data center	→ Email spam classification → Manufacturing: Finding known previously seen defects → weather prediction → disease classification

⇒ Choosing what features to use :-

↳ ~~Do~~ Always choose gaussian features. If any feature is non-gaussian, try to ~~change~~ convert it to gaussian.



plt.hist(x) (histogram)

↳ Try adding new features to remove the errors.

Collaborative Filtering :-

⇒ Making Recommendations :-

↳ Everytime you go to an online shopping website like Amazon or a movie streaming sites like netflix or other. Many of these sites will recommend things to you that they think you may like.

→ Predicting Movies Ratings :-

users rates movies using zero to five stars.

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	? never watched	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

notation

n_u = no. of users

n_m = no. of movies

$r(i,j) = 1$ if user j has rated movie i

$y^{(i,j)}$ = rating given by user j to movie i

$n_u = 4$

$n_m = 5$

$r(1,1) = 1$

$y^{(3,2)} = 4$

↳ With this framework for recommend systems, one possible way to approach the problem is to look at the movies that user have not ~~watched~~ rated and try to predict how users would rate those movies because then we can try to recommend things to users that they are more likely to rate as five stars.

Date
Page

⇒ Using Per-Item Features :-

what if we have features of the movies?

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

$n_u = 4$, $n_m = 5$, $\{n = 2$ (no. of features)$\}$

$x^{(1)} = \begin{bmatrix} 0.9 \\ 0 \end{bmatrix}$ $\quad x^{3} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$

For user i: Predict rating for movies i as: $w^{(i)} \cdot x^{(i)} + b^{(i)}$ (Just linear regression)

$w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$ $\quad b^{(1)} = 0$ $\quad x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix} \Rightarrow w^{(1)} \cdot x^{(3)} + b^{(1)} = 4.95$

For user j: Predict user j's rating for movie i as: $w^{(j)} \cdot x^{(i)} + b^{(j)}$

↳ This is a lot like linear regression except that we are fitting different linear regression model for each user.

→ Cost Function :-

Notation :-

$r(i,j) = 1$ if user j has rated movie i (0 otherwise)

$y^{(i,j)}$ = rating given by user j to movie i (if defined)

$w^{(j)}, b^{(j)}$ = parameters for user j

$x^{(i)}$ = feature vector for movie i

For user j and movie i, predict rating : $w^{(j)} \cdot x^{(i)} + b^{(j)}$

$m^{(j)}$ = no. of movies rated by user j

To learn $w^{(j)}, b^{(j)}$

$$\boxed{\frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2} + \underbrace{\frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2}_{\text{to prevent overfitting regularization}}$$

↳ cost function $J(w^{(j)}, b^{(j)})$

→ sum of all the movies user has actually rated.

↳ By minimizing the cost function, you get pretty good value of parameters $w^{(j)}$ & $b^{(j)}$.

↳ we can eliminate the term "$m^{(j)}$" as it a constant and creates no such difference.

To learn $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \ldots, w^{(n_u)}, b^{(n_u)}$ for all users:

$$\boxed{\frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y(i,j)\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2}$$

↳ Cost Function $J\left(\begin{matrix} w^{(1)} & \ldots & w^{(n_u)} \\ b^{(1)} & \ldots & b^{(n_u)} \end{matrix}\right)$

↳ We can use gradient descent or any other optimization to minimize this cost function.

Date
Page

⇒ Collaborative Filtering Algorithm :-

What if we don't have the features x_1 & x_2 ?

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
1) LaL	5	5	0	0	?	?
2) RF	5	?	?	0	?	?
3) CPoL	?	4	0	?	?	?
4) NCC	0	0	5	4	?	?
5) SvK	0	0	5	?	?	?

↳ Suppose we somehow know the parameters $w^{(1)}, w^{(2)}, \ldots$

$$w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}, \quad w^{(2)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}, \quad w^{(3)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}, \quad w^{(4)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$$

$$b^{(1)} = b^{(2)} = b^{(3)} = b^{(4)} = 0$$

using $w^{(j)} x^{(i)} + b^{(j)} = 0$

$$\left.\begin{aligned} w^{(1)} \cdot x^{(1)} &\approx 5 \\ w^{(2)} \cdot x^{(1)} &\approx 5 \\ w^{(3)} \cdot x^{(1)} &\approx 0 \\ w^{(4)} \cdot x^{(1)} &\approx 0 \end{aligned}\right\} \quad x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

In this way, we can guess the values of features.

→ **Cost Function :-**

Given $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \ldots, (w^{n_u}), b^{(n_u)}$

to learn $x^{(i)}$

$$\boxed{J(x^{(i)}) = \frac{1}{2} \sum_{j:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2}$$

to learn $x^{(1)}, x^{(2)}, \ldots, x^{(n_m)}$:

$$\boxed{J(x^{(1)}, x^{(2)} \ldots x^{(n_m)}) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2}$$

But remember, we assumed here that we already have the parameters w & b which is not true practically. So how do we get those parameters?
Here comes collaborative filtering.

→ <u>Collaborative Filtering</u> :-

To learn $w^{(1)}, b^{(1)}, \ldots, w^{(n_u)}, b^{(n_u)}$.

$$\min_{w^{(1)}, b^{(1)} \ldots w^{(n_u)}, b^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

To learn $x^{(1)}, x^{(2)}, \ldots, x^{(n_m)}$:

$$\min_{x^{(1)} \ldots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

putting them together:

$$\boxed{J(\vec{w}, b, x) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2}$$

↳ $\min_{\substack{w^{(1)}, \ldots, w^{(n_u)} \\ b^{(1)}, \ldots, b^{(n_u)} \\ x^{(1)}, \ldots, x^{(n_m)}}}$

Date
Page

→ Gradient Descent :

repeat {

$$w_i^{(j)} = w_i^{(j)} - \alpha \frac{\partial}{\partial w_i^{(j)}} J(w, b, x)$$

$$b^{(j)} = b^{(j)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w, b, x)$$

$$x_k^{(i)} = x_k^{(i)} - \alpha \frac{\partial}{\partial x_k^{(i)}} J(w, b, x)$$

}

⇒ Binary Labels : favs, likes and clicks :-

↳ To predict whether the user will like an item or not.
↳ To predict whether the user will ~~by~~ buy the item after being shown (used in ads).

predict that the probability of $y^{(i,j)} = 1$
it is given by,

$$g(w^{(j)} \cdot x^{(i)} + b^{(j)})$$

~~wch~~ where $\boxed{g = \frac{1}{1+e^{-z}}}$

→ Cost function for binary application :-

$$L(f_{(w,b,x)}(x), y^{(i,j)}) = -y^{(i,j)} \log(f(x)) - (1 - y^{(i,j)}) \log(1 - f(x))$$

$$J(w, b, x) = \sum_{(i,j):r(i,j)=1} L(f_{(w,b,x)}(x), y^{(i,j)})$$

Recommeder systems implementation details :-

⇒ Mean Normalization :-

↳ help the algorithm run faster.

movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	Eve(5)
LaL	5	5	0	0	?
RF	5	?	?	0	?
CPoL	?	4	0	?	?
NCC	0	0	5	4	?
SvK	0	0	5	?	?

↳ Suppose we have a new user Eve, who haven't rated any movie yet.

↳ If we run algorithm on Eve and try to minimise to cost function,

$$\underbrace{\frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(w^{(j)}\cdot x^{(i)}+b^{(j)}-y^{(i,j)}\right)^2}_{\text{this term means nothing for eve as he hasn't rated any movie.}}+\underbrace{\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(w_k^{(j)}\right)^2}_{\text{this term would minimize the } w\text{, which will be } 0.}+\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2$$

~~so~~ we end up getting the parameters as,

$$w^{(5)}=\begin{bmatrix}0\\0\end{bmatrix}, \quad b^{(5)}=0$$

↳ If we predict the ratings for Eve, ~~he will~~ with these parameters,

$$w^{(5)}\cdot x^{(i)}+b^{(5)}=0$$

we get zero rating for all the movies.

↳ This problem can be tackeled by Mean Normalization.

↳ Let's create a matrix ~~for~~ of each rating and take the mean of every row. (for some applications, take mean of columns)

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & ? & ? \end{bmatrix} \longrightarrow \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & ? & ? \end{bmatrix}$$

$$y(i,j) = y(i,j) - \mu$$

↳ Now, For user j, on movie i predict:

$$w^{(j)} \cdot x^{(i)} + b^{(j)} + \mu_i$$ {μ_i is added because rating can not be negative}

↳ For user 5 (Eve):

$$\underbrace{w^{(5)} \cdot x^{(1)} + b^{(5)}}_{0} + \underbrace{\mu_1}_{2.5} = 2.5$$

↳ This means, Eve is likely to rate a movie as its mean instead of zero, which is much more reasonable.

⇒ Tensorflow Implementation :-

↳ You don't need to calculate derivatives.

↳ For some functions, calculating its derivatives can be a very hard task. Tensorflow can help you in that.

i) Autodiff

Date
Page

→ Custom Training Loop

```
w = tf.Variable(3.0)
x = 1.0
y = 1.0  # target value
alpha = 0.1
```

tf.Variables are parameters we wanna optimize

$J = (wx - 1)^2$

```
iterations = 30
for iter in range(iterations):
    with tf.GradientTape as tape:
        fwb = w*x
        costJ = (fwb - y)**2
```

these steps are recorded in 'tape'.

```
    [dJdw] = tape.gradient(costJ, [w])
    w.assign_add(-alpha * dJdw)
```

tf.Variables requires special functions to modify

→ Collaborative Filtering implementation:-

```
optimizer = keras.optimizers.Adam(learning_rate = 1e^-1)
iterations = 200
for iter in range(iterations):
    with tf.GradientTape as tape:
        cost_value = — —
        —

    grads = tape.gradient(cost_value, [x, w, b])
    optimizer.apply_gradients(zip(grads, [x, w, b]))
```

⇒ Finding Related Items :-

↳ Suppose, you have a movie or item i with feature $x^{(i)}$.

↳ To find other related items,
find item k with $x^{(k)}$ similar to $x^{(i)}$.

$$\sum_{l=1}^{n} (x_l^{(k)} - x_l^{(i)})^2 \text{ or } ||x^{(k)} - x^{(i)}||^2$$ → with this you get item k similar to item i.

→ Limitations of Collaborative Filtering :-

↳ Cold start problem. How to
↳ rank new items that few users have rated.
↳ show something reasonable to new users who have rated few items.

↳ Use side information about items or users:
↳ Item: Genre, movie stars, studio, ...
↳ User: Demographics (age, gender, location), expressed preferences

Content Based Filtering :-

⇒ Collaborative Filtering vs Content Based ~~Ft~~ Filtering :-

Collaborative Filtering	Content Based Filtering
recommend items to you based on ratings of users who gave similar ratings as you.	recommend items to you based on features of user and item to find good match.

user features:
- ↳ Age
- ↳ Gender (1 hot)
- ↳ Country (1 hot, 200)
- ↳ Movies watched
- ↳ Average rating per genre

} $x_u^{(j)}$ for user j

movie features:
- ↳ Year
- ↳ Genres
- ↳ Reviews
- ↳ Average rating

} $x_m^{(i)}$ for movie i

vector size could be different

↳ Try to figure out if $x_m^{(i)}$ is good match for $x_u^{(j)}$.

Predict rating of user j on movie i as:

$$\boxed{v_u^{(j)} \cdot v_m^{(i)}}$$

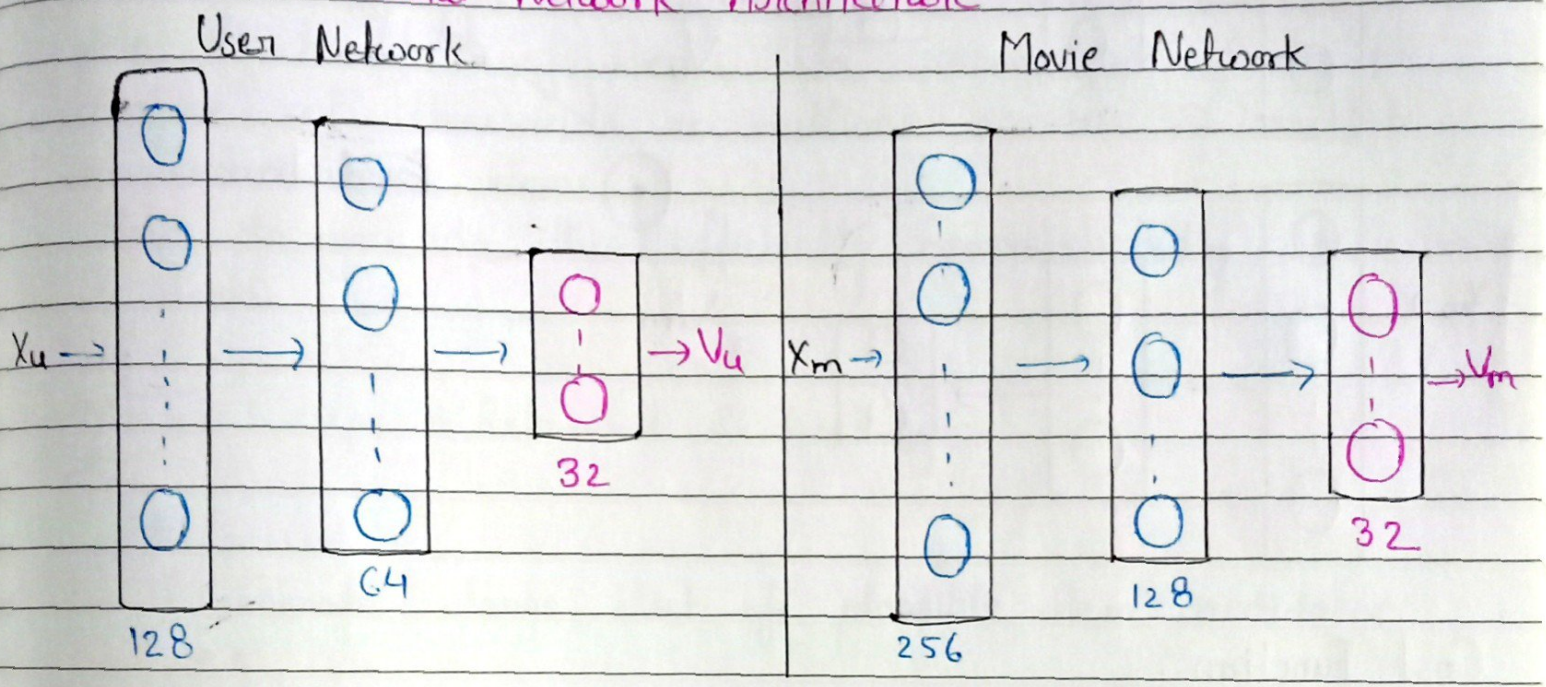
derived from $x_u^{(j)}$ — derived from $x_m^{(i)}$

↳ Now, the task is to compute $v_u^{(j)}$ & $v_m^{(i)}$ out of/from $x_u^{(j)}$ & $x_m^{(i)}$.

↳ One thing to keep in mind ~~of~~ is that ~~do~~ size of $v_u^{(j)}$ & $v_m^{(i)}$ should be same in order to calculate the dot product.

⇒ Deep Learning for Content Based Filtering :-

Neural Network Architecture



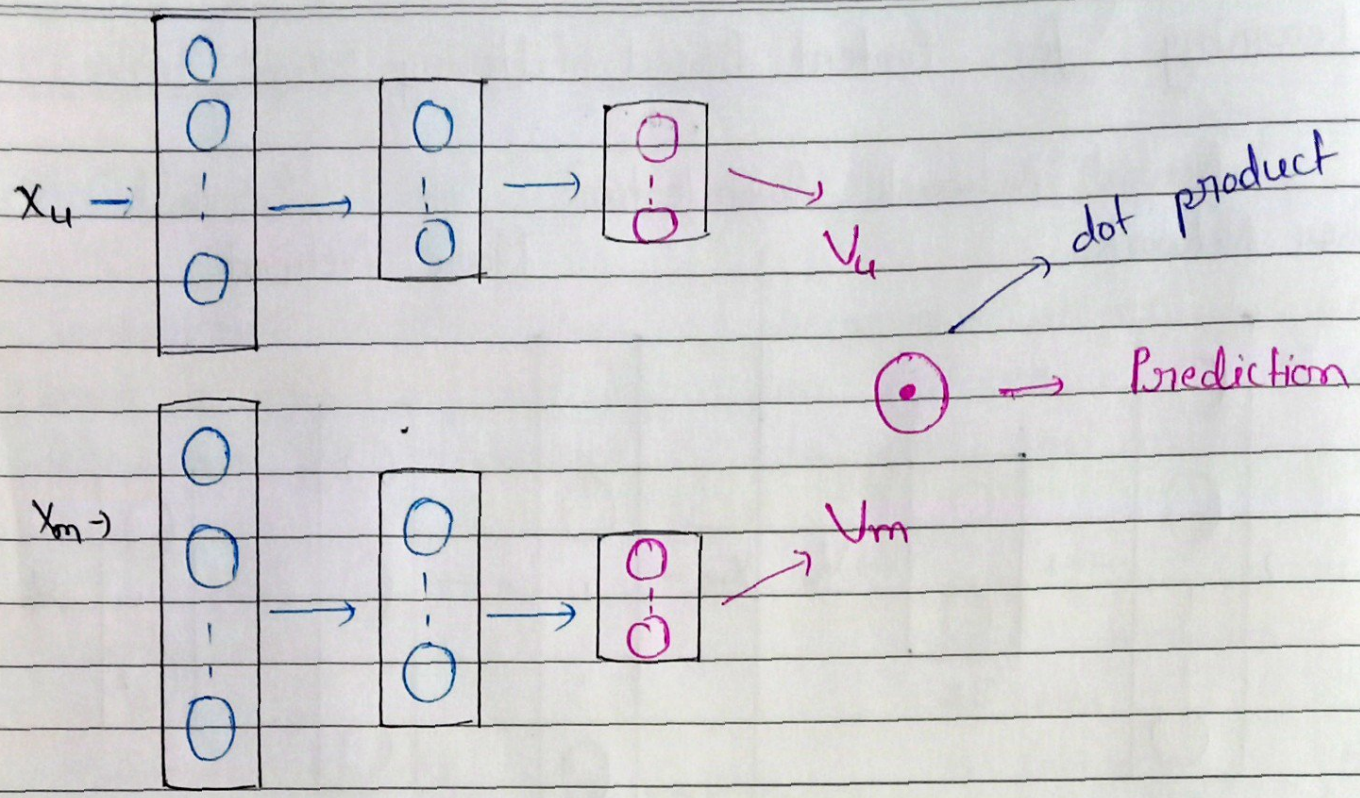
prediction: $V_u \cdot V_m$

$g(V_u \cdot V_m)$ for binary features such as likes.

↳ Notice that user network and movie network can hypothetically have different number of layers and different number of units per hidden layer. Only the output layer needs to have same size.

↳ Here, we have created two neural network for user and movie separately.

↳ Lets see how we can represent these two networks together.



Cost Function:

$$\boxed{J = \sum_{(u,j):r(i,j)=1} \left(V_u^{(j)} \cdot V_m^{(i)} - y^{(i,j)}\right)^2 + \text{NN regularization term}}$$

↳ This ~~it~~ is the cost function used to find the features for both user and movie.

Learned user and item vectors:

↳ $V_u^{(j)}$ is a vector of length 32 that describes user j with features $x_u^{(j)}$.

↳ $V_m^{(i)}$ is a vector of length 32 that describes movie i with features $x_m^{(i)}$.

To find movies similar to i:

↳ as we have vector $V_m^{(i)}$ for movie i. To find a movie similar to i, we find vector $V_m^{(k)}$ similar to $V_m^{(i)}$.

$$\| V_m^{(k)} - V_m^{(i)} \|^2 \rightarrow \text{small}$$

note: This can be pre-computed ahead of time.

Date
Page

⇒ Recommending from a large catalogue :-

↳ Today's recommended systems will sometimes need to pick a handful of items to recommend from a catalogue of thousands or millions or 10s of millions or even more items.
How do you do this efficiently computationally, let's take a look.

→ Two steps : Retrieval & Ranking :

Retrieval :
↳ Generate large list of plausible item candidates
e.g.:
1) For each of the last 10 movies watched by the user, find 10 most similar movies.

$$\|V_m^{(k)} - V_m^{(1)}\|^2$$

2) For most viewed 3 genres, find the top 10 movies.
3) Top 20 movies in the country

↳ Combine retrieved items into list, removing duplicates and items already watched / purchased.

Ranking :
↳ Take list retrieved and rank using learned model.
↳ Display ranked items to user.

note: - if V_m is already calculated, just calculate V_u.

Date
Page

→ Retrieval Step:

↳ Retrieving more items results in better performance, but slower recommendations.

↳ To analyse/optimize the trade-off, carry out offline experiments to see if retrieving additional items results in more relevant recommendations. (i.e., $p(y^{(i,j)}) = 1$ of items displayed to user are higher).

⇒ TensorFlow implementation of Content-Based Filtering :-

```
user_NN = tf.keras.models.Sequential([
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(32)
    ])

item_NN = tf.keras.models.Sequential([
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(32)
    ])

input_user = tf.keras.layers.Input(shape=(num_user_features))
vu = user_NN(input_user)
vu = tf.linalg.l2_normalize(vu, axis=1)

input_item = tf.keras.layers.Input(shape=(num_item_features))
vm = item_NN(input_item)
vm = tf.linalg.l2_normalize(vm, axis=1)

output = tf.keras.layers.Dot(axes=1)([vu, vm])

model = Model([input_user, input_item], output)

cost_fn = tf.keras.losses.MeanSquaredError()
```

Reinforcement Learning Introduction :-

⇒ What is reinforcement learning? :-

↳ RL enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences.
↳ It uses rewards and punishments as signals for positive and negative behaviour.

→ Autonomous Helicopter Example:

position of helicopter ⟶ how to move control sticks

state s ⟶ action a

↳ Reward Function :
↳ It is an incentive mechanism that tells the agent what is correct and what is wrong using reward and punishment.

positive reward: +1 if helicopter flying well
negative reward: -1000 if helicopter flying poorly

→ Application :
↳ Controlling Robots
↳ Factory Optimization
↳ Financial (stock) Trading
↳ Playing Games (including video games)

Date
Page

⇒ Mars Rover Example :-

↳ We'll develop reinforcement learning using a simplified example inspired by the Mars rover.

↳ Here, the rover can be any of six positions, as shown by six boxes here.

pic 1 (100 reward)	(0)	(0)	Rover 'R' (0)	(0)	pic 2 (40 reward)
1	2	3	4	5	6

state

← Left Right →

↳ The rover can either go to left or right.

↳ Possible actions :-

state ← →

i). 4 3 2 1
0 0 0 100

ii) 4 5 1
0 0 40

iii) 4 5 4 3 2 1
0 0 0 0 0 100

state action reward new state

(S, a, R(S), S')

↓

(4, ←, 0, 3)

↳ Thats the formulism of how RL works.

⇒ The return in RL :-

↳ Now, you know what are the states of reinforcement learning application and how depending on the actions you take, you go through different state, and also get to enjoy different rewards.
But how do you know if a particular set of rewards is better or worse than a different set of rewards?

↳ The concept of a return captures that rewards you can get quicker are maybe more attractive than rewards that take you a long time to get to.

→ mars rover example :-

Pic 1 (100)	(0)	(0)	Rover 'R' (0)	(0)	Pic 2 (40)
1	2	3	4	5	6

state

$$\text{Return} = 0 + \underset{\substack{\downarrow \\ \text{discount factor } (\gamma < 0)}}{(0.9)}\, 0 + (0.9)^2\, 0 + (0.9)^3\, 100 = 0.729 \times 100 = 72.9$$

$$\boxed{\text{Return} = R_1 + \gamma R_2 + \gamma^2 R_3 + \text{------ (until terminal state)}}$$

Discount Factor $\gamma = 0.9,\ 0.99,\ 0.999$ } in many applications

↳ The γ in financial applications represents how much less the amount of money will value that it does today.

⇒ Making Decisions : Policies in RL :-

↳ A policy is a function $\pi(s) = a$ mapping from states to actions, that tells you what action a to take in a given state s.

$$\underset{s}{\text{state}} \xrightarrow{\text{policy}} \underset{a}{\text{action}}$$

π

100	←	←	←	→	40

$\pi(s) = a$

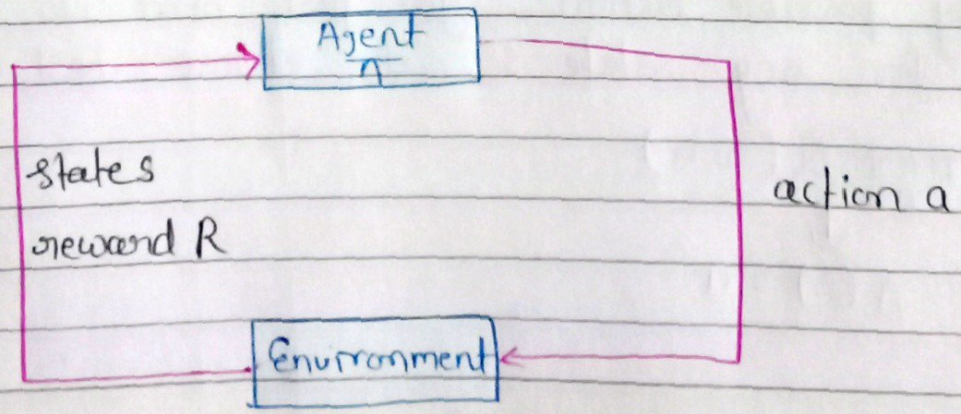
$\pi(2) = \leftarrow$
$\pi(3) = \leftarrow$
$\pi(4) = \leftarrow$
$\pi(5) = \rightarrow$

↳ The goal of reinforcement learning is to find a policy π that tells you what action $(a = \pi(s))$ to take in every state (s) so as to maximize the return.

⇒ Markov - Decision Process :-

↳ In MDP, the future depends only on where you are now, not on how you got there.

↳ One other way to think of the MDP is that the future only depends on the current state and not on how anything that might have occured to getting to current state.



State - Action Value Function :-

⇒ State - Action Value Function Definition :- (Q - function)

↳ It specifies how good it is for an agent to perform a particular action in a state with a policy π.

$Q(s,a)$ = Return if you
- ↳ start in state s.
- ↳ take action a (once)
- ↳ then behave optimally after that

100	50	25	12.5	20	40	← Return
	←	←	←	→		← action
100	0	0	0	0	40	← reward

$Q(2, \rightarrow) = 0 + (0.5)0 + (0.5)^2 + (0.5)^3 100 = 12.5$

$Q(2, \leftarrow) = 0 + (0.5)100 = 50$

100 100	50 12.5	25 6.25	12.5 10	6.25 20	40 40
100	0	0	0	0	40

↳ The best possible return from state s is $\max_a Q(s,a)$.

↳ The best possible action in state s is the action that gives $\max_a Q(s,a)$

↳ So, if you ~~co~~ have calculated $Q(s,a)$ for every state and every possible action, you just need to look at $Q(s,a)$ for any state s to choose best action a.

$$\max_a Q(s,a)$$
$$\downarrow$$
$$\pi(s) = a$$

Date
Page

Bellman Equations :-

$$\boxed{Q(s,a) = R(s) + \gamma \max_{a'} Q(s', a')}$$

s: current state
a: current action
s': state you get to after taking action a
a': action that you take in state s'

R(s): Reward of current state

if you are at terminal state :

$$\boxed{Q(s,a) = R(s)}$$

↳ In the bellman equation :
R(s) = Reward you get right away
$\max_{a'} Q(s', a')$ = Return from behaving optimally starting from state s'

↳ The bellman equation can also be written as:

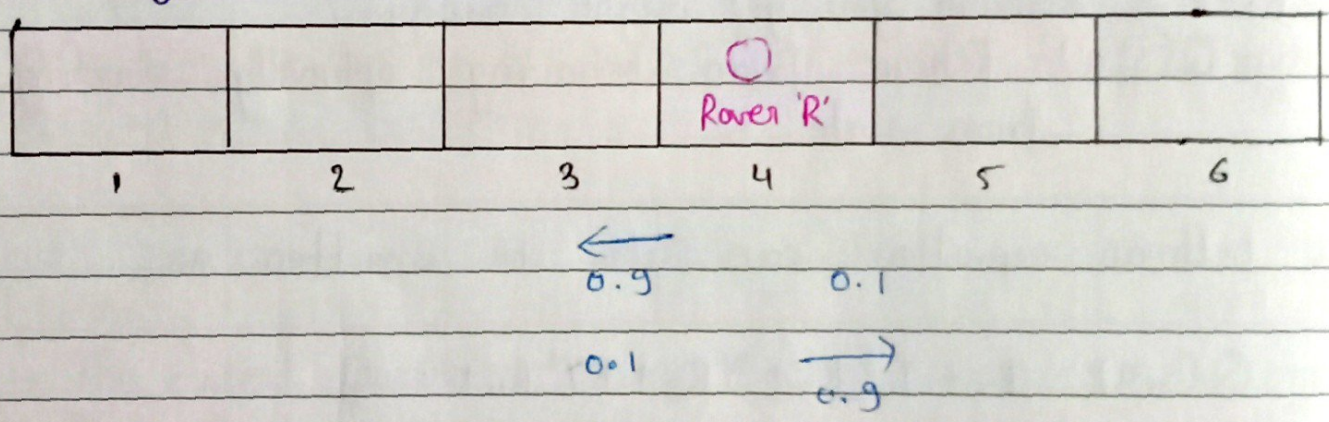
$$Q(s,a) = R_1 + \gamma [R_2 + \gamma R_3 + \gamma^2 R_4 + \dots]$$

⇒ Random (Stochastic) Environment :-

↳ In some applications, when you take an action, the outcome is not always completely reliable.

↳ For example, if you command your mars rover to go left, maybe there's a slippel little bit of rock slide, or maybe the floor is really slippery and so it slips and goes to the wrong direction.

↳ Let's say you take an action and command it to go left. Most of the time you'll succeed but what if 10% of the time, it actually ends up accidently slipping and going in opposite direction.



→ Expected Return:

100	← 0	← 0	← 0	→ 0	40
1	2	3	4	5	6

Suppose, you start at 4 and follow this policy. Then there can be many random sequences of visits.

i) 4 3 2 1
0 0 0 100

ii) 4 3 4 3 2 1
0 0 0 0 0 100

iii) 4 5 6
0 0 40

Date
Page

↳ When the RL problem is stochastic, there isn't one sequence of rewards instead you see diffrent sequence of rewards.

↳ Here, we are interested in maximizing the average of sum of rewards.

$$\boxed{\begin{aligned} \text{Expected Return} &= \text{Average}(R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots) \\ \hookrightarrow \quad ER &= E(R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots) \end{aligned}}$$

→ goal of RL: Choose a policy $\pi(s) = a$ that will tell us what action 'a' to ~~tack~~ take in state 's' so as to maximize the expected return.

→ Bellman Equation:

$$\boxed{Q(s,a) = R(s) + \gamma E\left[\max_{a'} Q(s', a')\right]}$$

as the next state s' is random, so we take average value of $\max_{a'} Q(s', a')$.

Continuous State Spaces :-

⇒ Discrete vs Continuous :-

↳ Discrete states represent count. In our mars rover example, there were 6 states the rover could be in. The rover could not be in a state between 1 and 2.

↳ Continuous states represents measurable amounts. Like, an actual mars rover can be anywhere between a ~~continue~~ continuous line from 0 to 6 km.

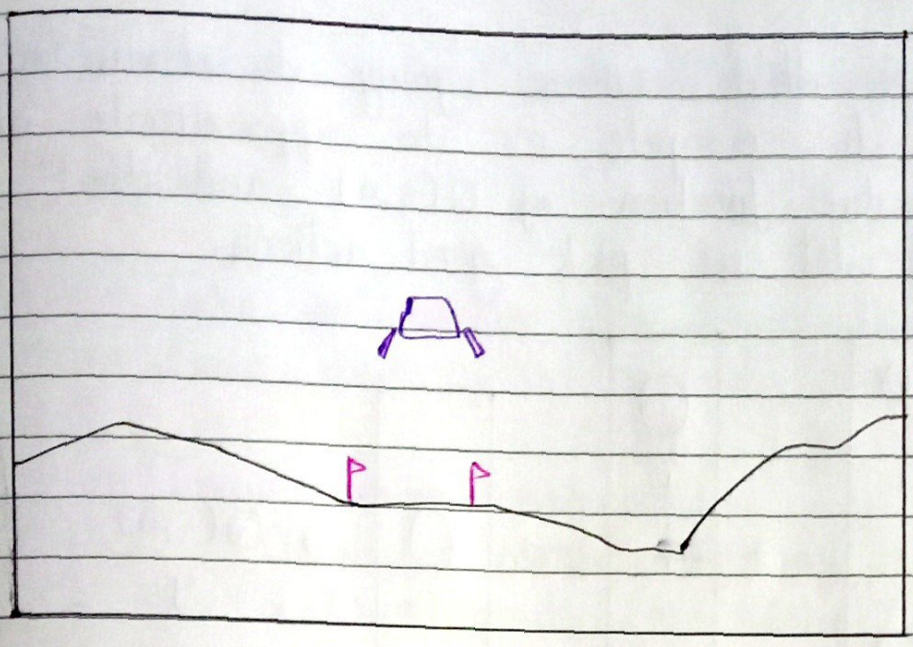
↳ Let's take an example of a car. In this case, a car can be anywhere in x-direction and y-direction, also it can have any of the angle between 0° to 360°.
Here, to represent a state of car, we need to use a vector.

$$S = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \begin{array}{l} \rightarrow x\text{-direction (position)} \\ \rightarrow \text{position in } y\text{-direction} \\ \rightarrow \text{angle} \\ \rightarrow \text{velocity in } x\text{-direction} \\ \rightarrow \text{velocity in } y\text{-direction} \\ \rightarrow \text{angular velocity} \end{array}$$

↳ For autonomous Helicopter,

$$S = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \omega \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix}$$

⇒ Lunar Lander :-



actions:
- ↳ do nothing
- ↳ left thruster
- ↳ main thruster
- ↳ right thruster

Reward Function :
- ↳ Getting to landing pad : 100 - 140
- ↳ Additional reward for moving towards / away from the pad.
- ↳ Crash : −100
- ↳ Soft landing : +100
- ↳ Leg grounded : +10
- ↳ Fire main engine : −0.3 } to avoid unnecessary
- ↳ Fire side thruster : −0.03 } fuel burning

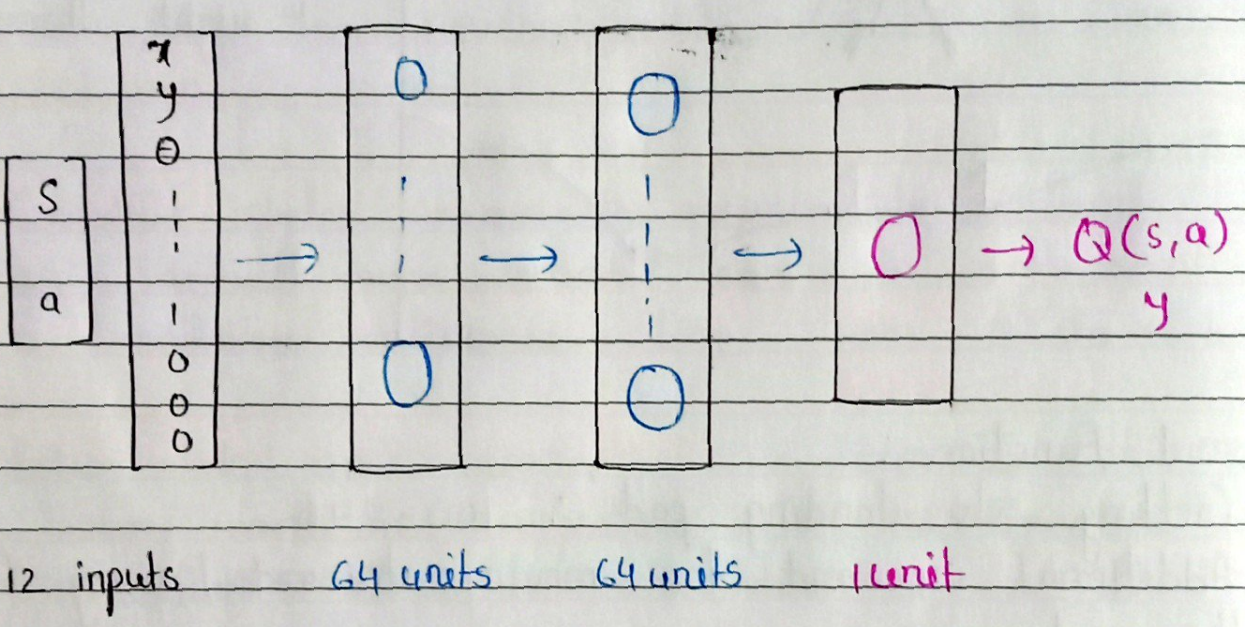
↳ Learn a policy π that, given

$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$$

picks action $a = \pi(s)$ so as to maximize the return. $\{\gamma = 0.985\}$

⇒ Learning the State-Value Function :-

↳ The key idea is that we're going to train a neural network to compute or to approximate the state action value function of $Q(s,a)$, and that in turn will let us pick good actions.



In a state 's', use neural network to compute $Q(s, \text{nothing})$, $Q(s, \text{left})$, $Q(s, \text{main})$, $Q(s, \text{right})$ and pick the action 'a' that maximizes $Q(s,a)$.

↳ The question is, how do you train a neural network to output $Q(s,a)$?

↳ We will use bellman equation to create a training set and train the network.

$$\underbrace{Q(s,a)}_{x} = \underbrace{R(s) + \gamma \max_a Q(s',a')}_{y}$$

at first $Q(s',a')$ is any random value.

{collect $(s, a, R(s), s')$}

↳ We use the lunar lander and take actions randomly. By doing so we collect a lot of examples when we were in a state 's', took action 'a' and get a result

Date
Page

→ Learning Algorithm :- (DQN = Deep Q - Network)

↳ Initialize neural network randomly as guess of $Q(s,a)$

↳ Repeat {

→ Take actions in the lunar lander. Get $(s, a, R(s), s')$.

→ Store 10,000 most recent $(s, a, R(s), s')$ tuples.

↳ {Replay Buffer}

→ Train neural network:

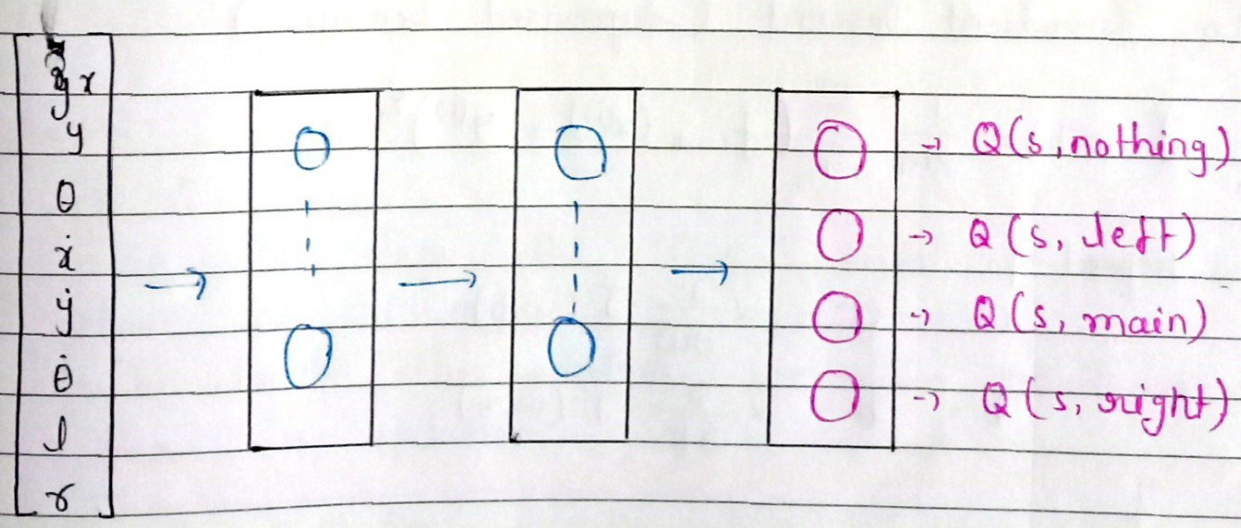
Create training set of 10000 examples using $x = (s, a)$ and $y = R(s) + \gamma \max_a Q(s', a')$

Train Q_{new} such that $Q_{new}(s,a) \approx y$.

→ Set $Q = Q_{new}$.

}

⇒ Algorithm Refinement : Improved NN Architecture :-



pick the action 'a' that maximizes $Q(s,a)$.

Date
Page

⇒ Algorithm Refinement : ϵ - Greedy Policy :-

↳ To pick actions while you're still learning.

→ How to choose actions:

In some state 's'

option 1:

↳ Pick the action 'a' that maximizes $Q(s,a)$. → works okay

option 2: { ϵ - greedy policy ($\epsilon = 0.05$) }

↳ with probability 0.95, pick the action 'a' that maximizes $Q(s,a)$. (Greedy, 'Exploitation')

↳ with probability 0.05, pick the action 'a' randomly. ("Exploration") # To discover new actions

@ Start ϵ high, then gradually decrease ($1.0 \rightarrow 0.01$)

⇒ Algorithm Refinement : Mini - batch and Soft updates :-

→ Mini - Batch :-

In Gradient Descent (Supervised Learning):

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)} \right)^2$$

Repeat {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

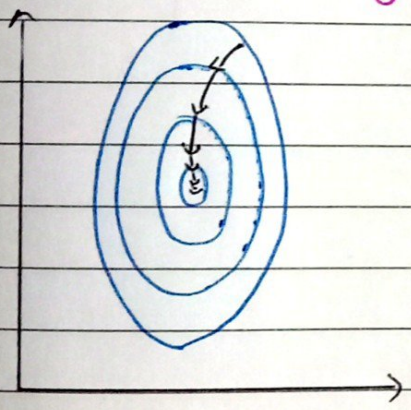
}

↳ here if m is very very large, the descent will be very slow.

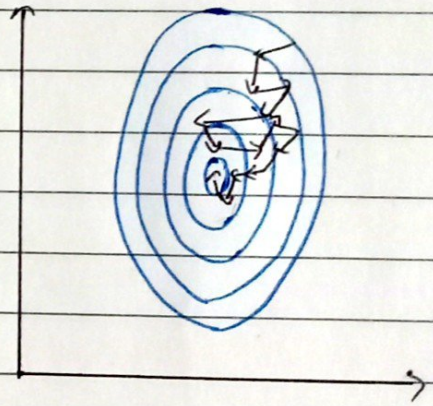
↳ In that case, we can take just a small portion(m) of total population (m) and apply gradient descent to it. In each iteration, we replace the set of m' with new portion of m. In It is called 'mini-batch' and gradient descent is then called 'mini-batch gradient descent.

↳ The Mini-batch technique can also be applied in reinforcement learning. Here, we won't take all 10,000 example but a set of 1000 examples from it.

Batch Learning



Mini-Batch Learning



→ Soft Update :-

→ Set $Q = Q_{new}$

↳ in this step, the 'Q_{new}' might be even worse choice of $Q(s, a)$.

↳ To tackle this problem, we take small part of Q_{new} in updation.

$$\boxed{Q = 0.01\, Q_{new} + 0.99\, Q}$$

Date
Page

⇒ Limitations of RL :-

↳ Much easier to get to work in a simulation than a real robot!

↳ Far fewer applications than supervised and unsupervised learning.