16/08/2023

Deep learning

Advanced learning algorithms:

1). Neural Network

- inference (prediction)
- training

2). Practical advice for building machine learning systems

3). Decision Tree

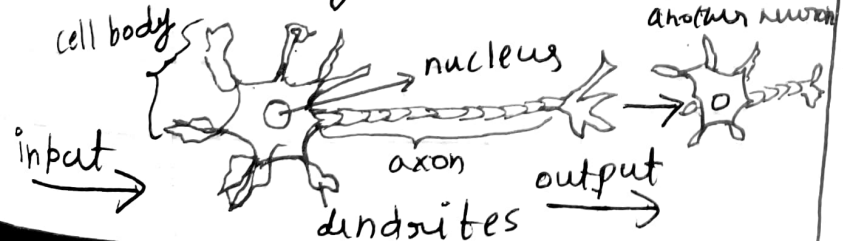
(i) Neural networks intuition

Neurons and the brain

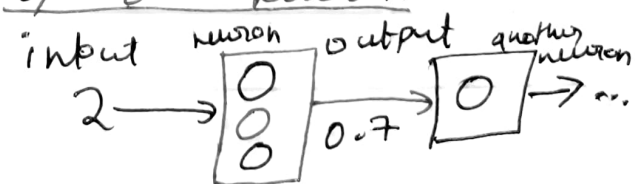
Neural Network

- origins: Algorithms that try to mimic the human brain.
- Used in the 1980's & early 1990's
  Fell out of favor in the late 1990's
- Resurgence from around 2005
- Speech → Images → text (NLP) → ...

Biological neuron



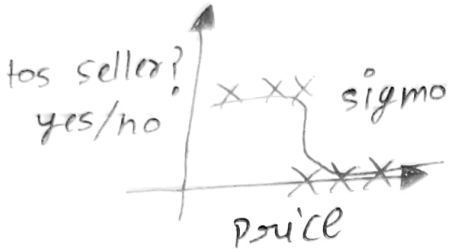
simplified mathematical model of a neuron



(ii) Demand Prediction

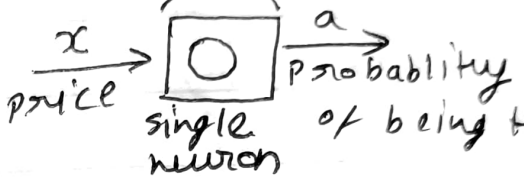
Example: 

x = price (Input)



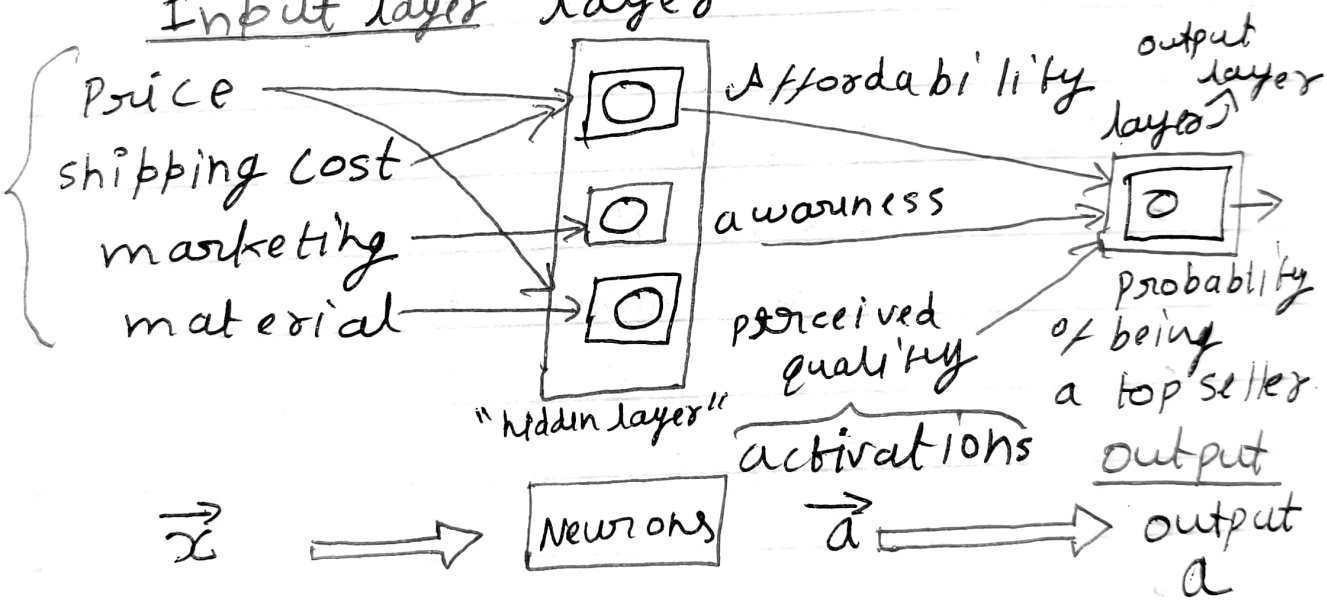
$a = f(x) = \frac{1}{1+e^{-(wx+b)}}$ (Output)

a ← activation



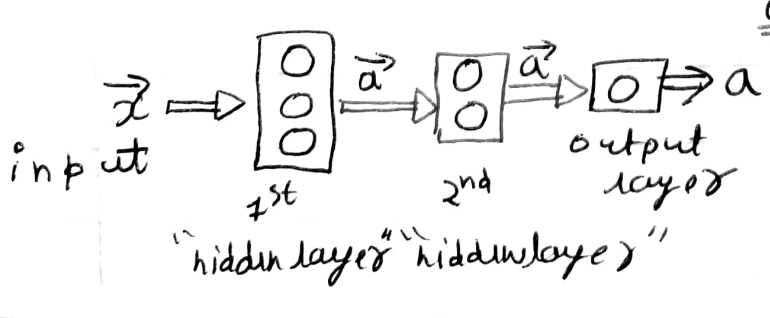
Example: if we have multiple features

Input layer — layer — output layer

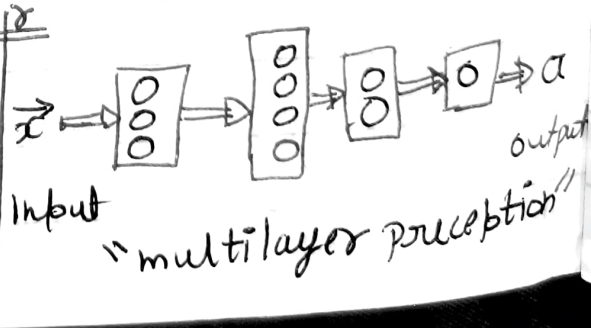


$\vec{x}$ → Neurons → $\vec{a}$ → output a

multiple Hidden layers

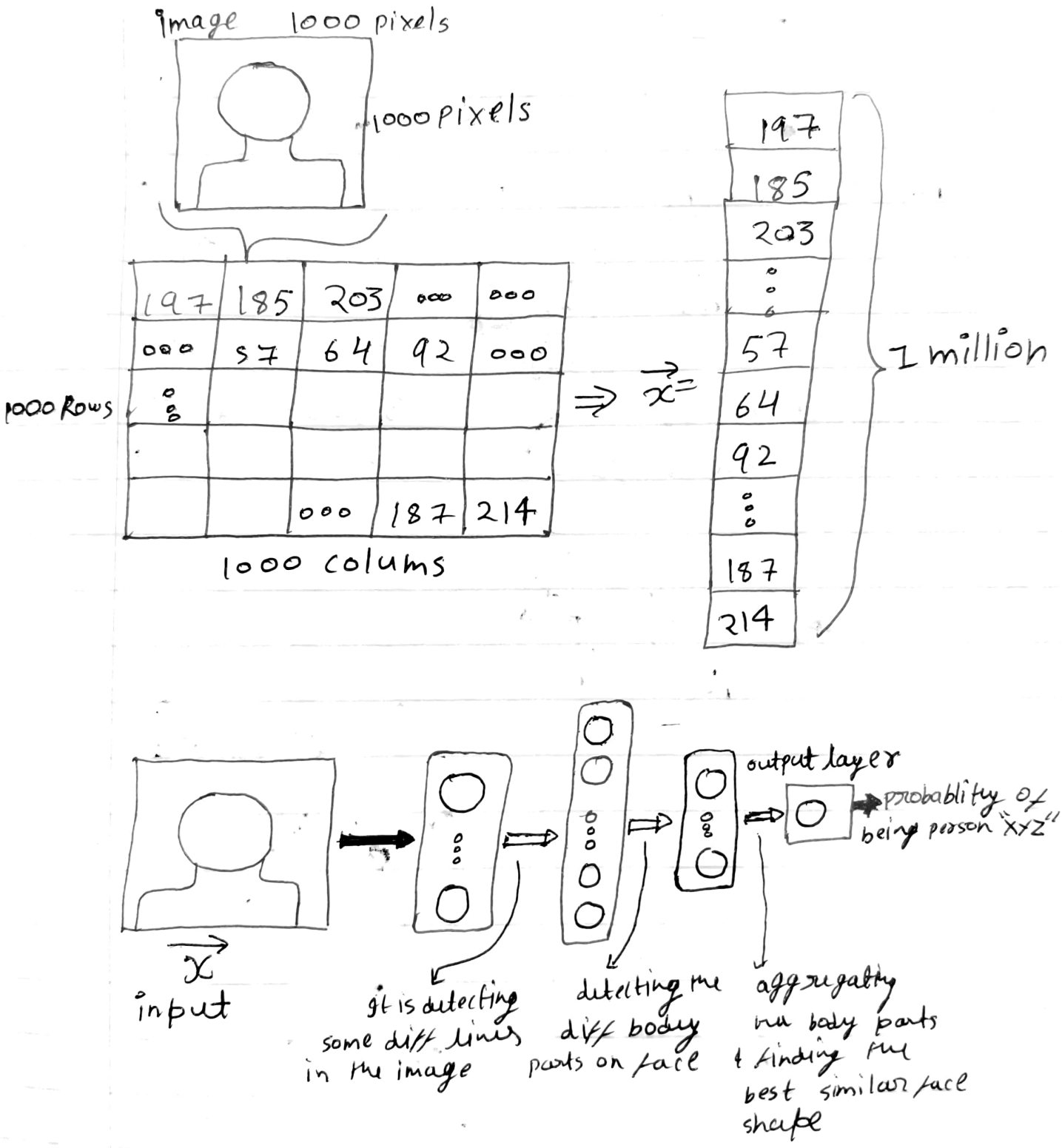


or



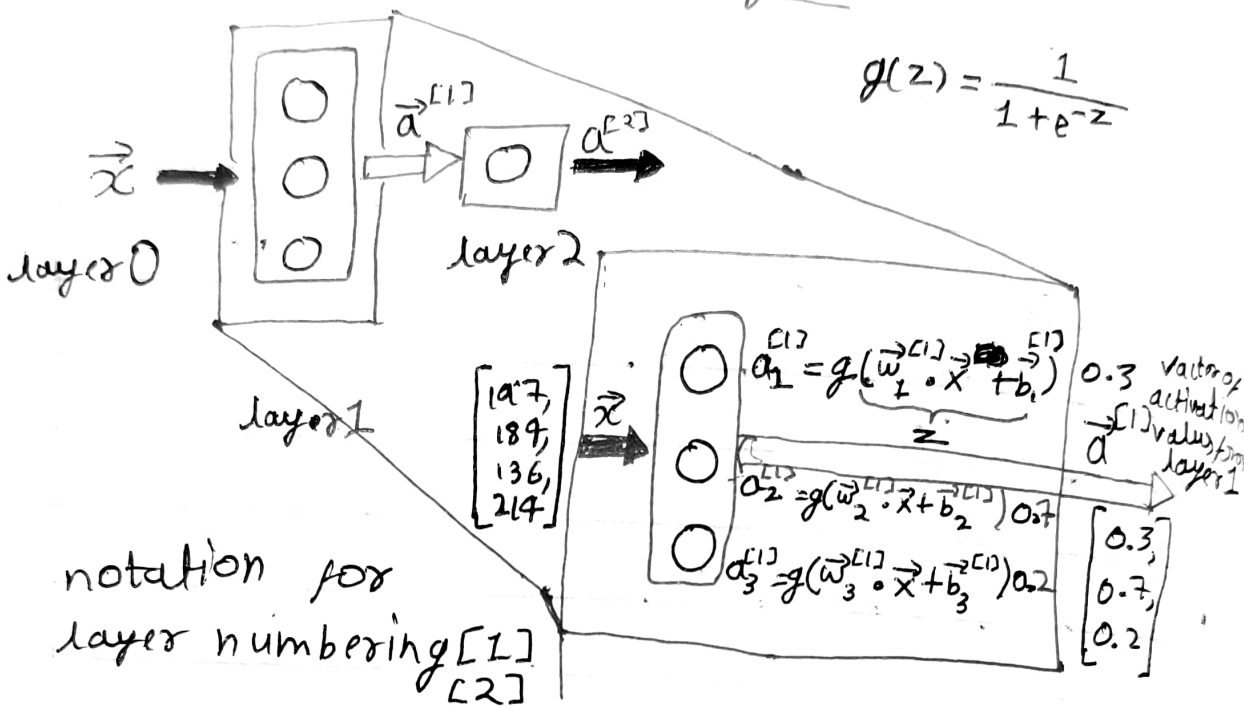
(iii) # Example: Recognizing Images

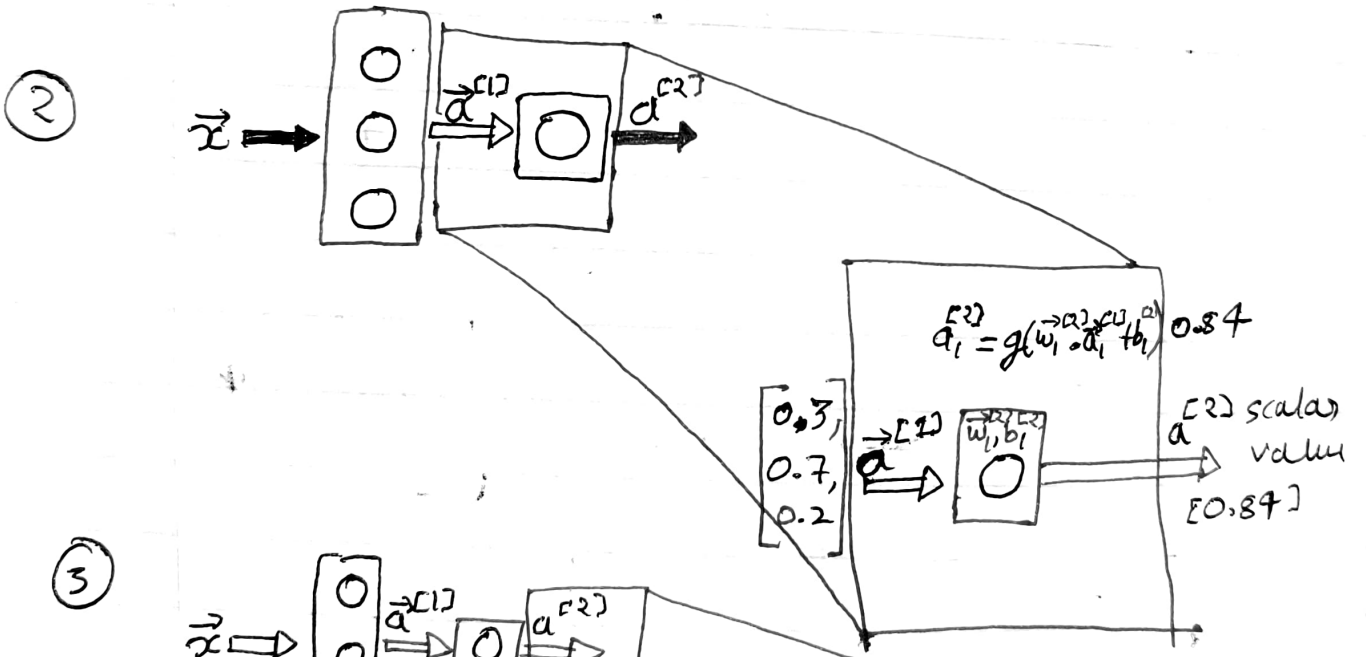
Face recognition

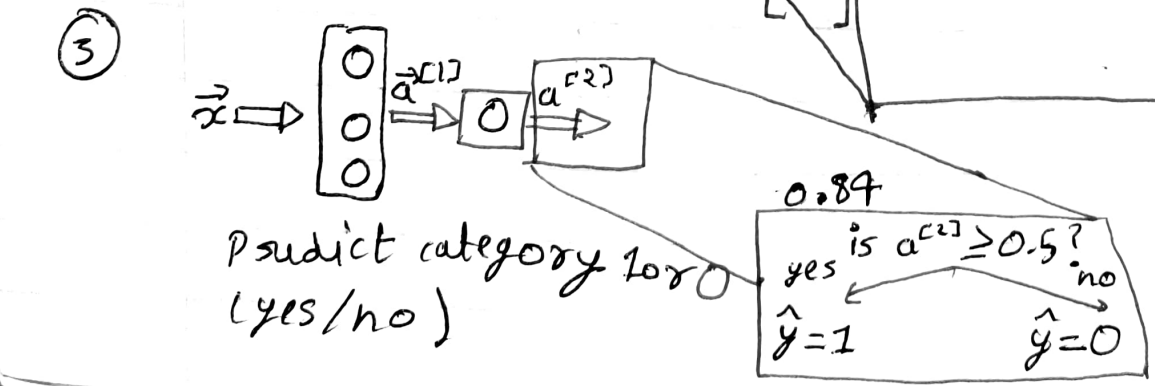


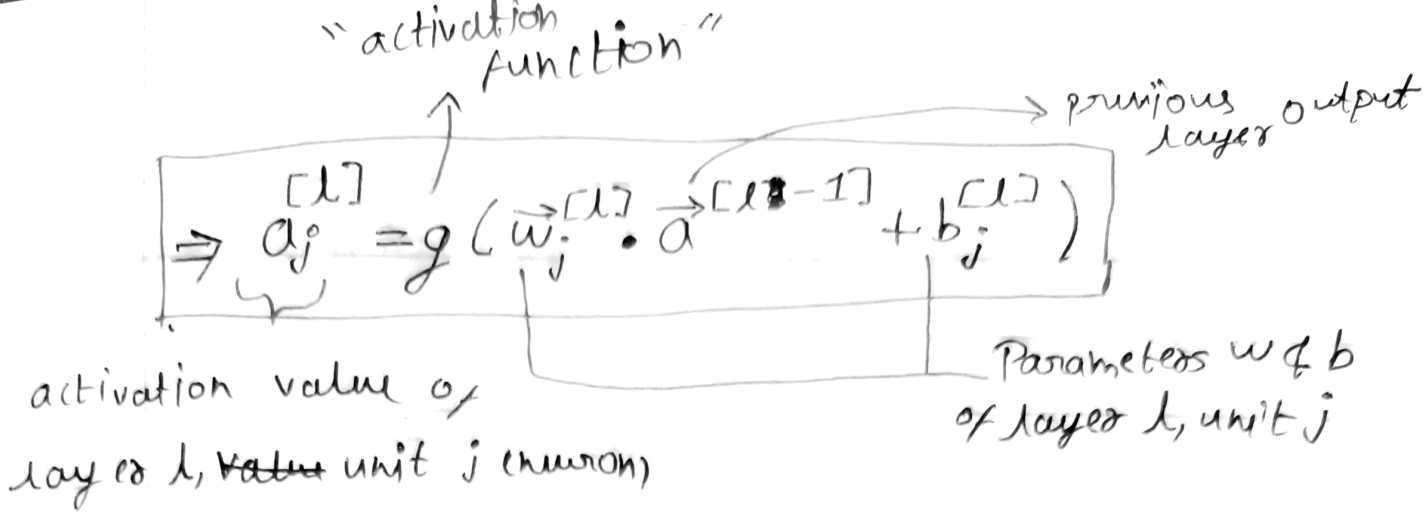
Neural Network Model

(i) Neural network layer



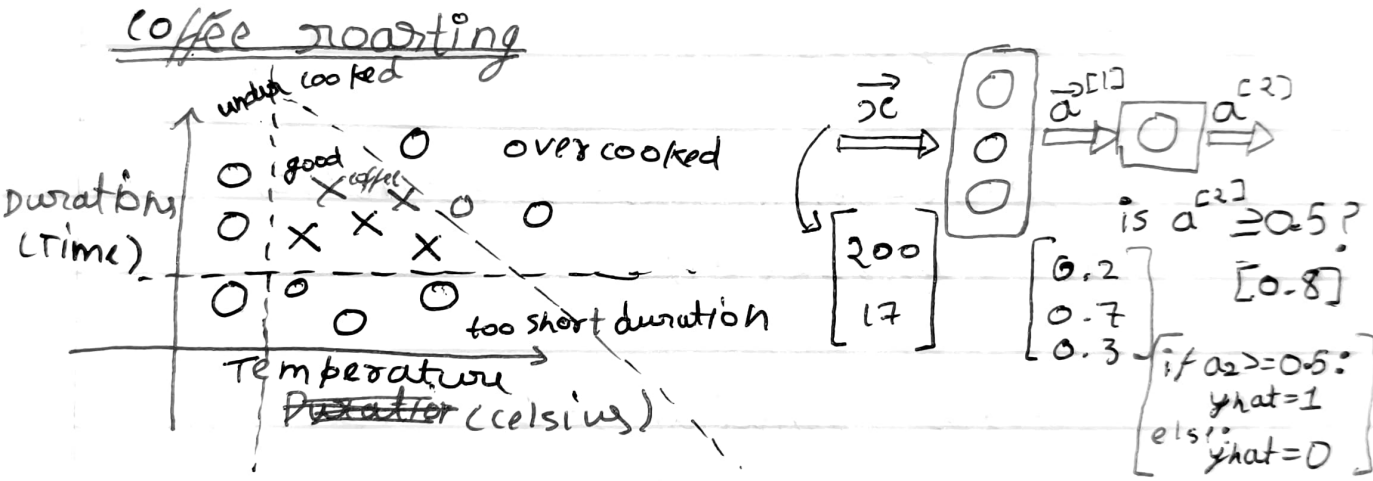






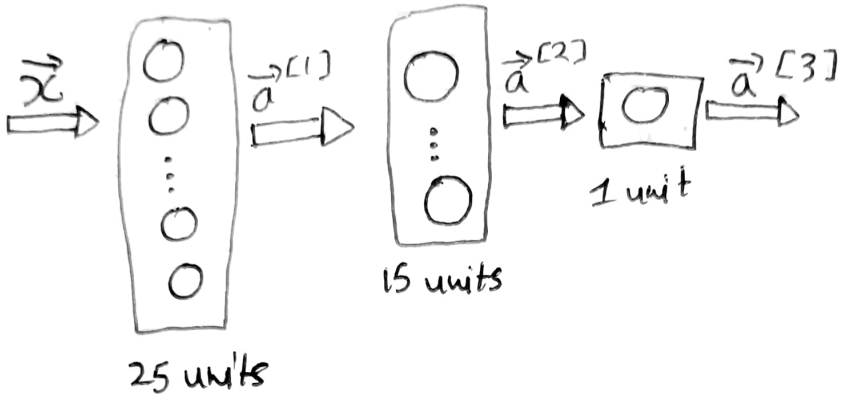
TensorFlow implementation

(i) Inference in Code




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

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

model for digit classification



inference code

```
x = np.array([[0.0, ...245, ...250, ...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
```

(ii) Data in TensorFlow

Feature vectors

temperature (celsius)	duration (minutes)	Good coffee? (1/0)
200.0	17.0	1
425.0	18.5	0
...	...	...

```
x = np.array([[200.0, 17.0]])
```

`[[200.0, 17.0]]` → why double braces?

$\Rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times3}$ if we want to implement this in code then we have to use above formate with double braces

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

`[[1,2,3], [4,5,6]]`

* Now when we used [[200.0, 17.0]] by duble braces because we have to pass a 2-D array instead of 1-D array.

2-D vector

$x = np.array([[200.0, 17.0]]) \rightarrow [200, 17]$ 1x2

$x = np.array([[200.0], [17.0]]) \rightarrow \begin{bmatrix} 200 \\ 17 \end{bmatrix}$ 2x1

1-D vector

$x = np.array([200.0, 17.0])$

no rowse & no colums

Going back to the code..,

```
x = np.array([[200, 17]])
layer_1 = Dense(units = 3, activation='sigmoid')
a1 = layer_1(x)
```

data in tensorFlow

⇒ print(a1)

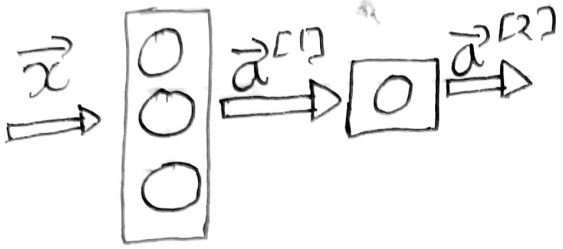
```
[tf.Tensor([[0.2, 0.7 0.3]], shape=(1,3), dtype=float
```

data in Numpy

⇒ a.numpy()

```
[array([[0.2, 0.7, 0.3]], dtype=float32)]
```

(111) Building a neural Network



		y
200	17	1
120	5	0
...		...

neural network model

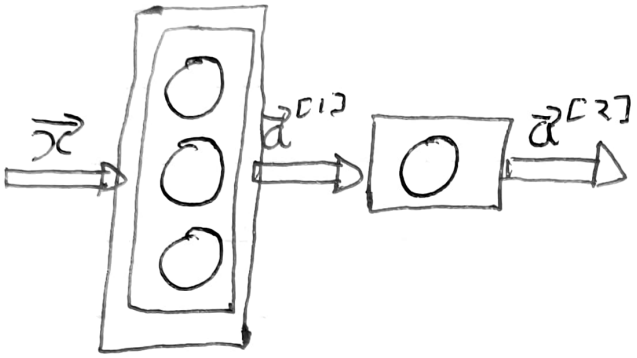
```
layer_1 = Dense(units=3, activation='sigmoid')
layer_2 = Dense(units=1, activation='sigmoid')
model = sequential([layer_1, layer_2])

x = np.array([ [200.0, 17.0],
               [120.0, 5.0],      4X2
               [425.0, 20.0],
               [212.0, 18.0] ])

y = np.array([1,0,0,1])

model.compile(...)  ← more about this's next week!
model.fit(x,y)

model.predict(x_new)
```

we can do this also

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

Neural network implementation in python

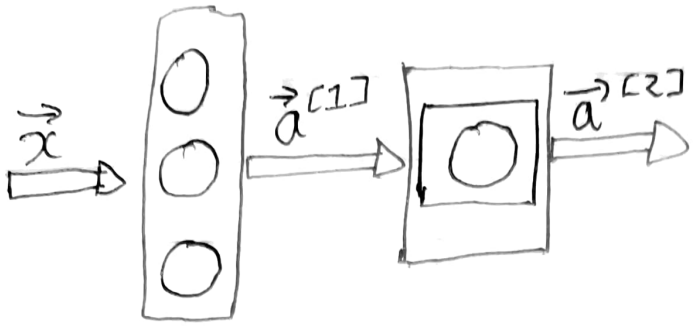
(i) Forward prop in a single layer




```
x = np.array([200, 17])        1 D Array
# a_1^[1] = g(w_1^[1] · x + b_1^[1])      # a_2^[1]      # a_3
```

```
w1_1 = np.array([1,2])          w1_2 = [-3,4]
b1_1 = np.array([-1])           b1_2 = [1]
z1_1 = np.dot(w1_1, x)+b1_1     z1_2 = np.dot(....)
a1_1 = sigmoid(z1_1)            a1_2 = sigmoid(z1_2)
```

```
a1 = np.array([a1_1, a1_2, a1_3])
```



$a_1 = np.array([a1\_1, a1\_2, a1\_3])$

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

$w2\_1 = np.array([-7, 8, 9])$
$b2\_1 = np.array([3])$
$z2\_1 = np.dot(w2\_1 \cdot a_1) + b2\_1$
$a2\_1 = sigmoid(z2\_1)$

$\Downarrow$

$a2 = np.array([a2\_1])$

(ii) General implementation of forward propagation

forward prop in Numpy

activation vector

```
def dense(a_in, W, b):
    units = W.shape[1]  # no. of columns
    a_out = np.zeros(units)  # [0,0...,0]
    for j in range(units):
        w = W[:, j]  # columns 1 by 1
        z = np.dot(W, a_in) + b[j]
        a_out[j] = g(z)  # sigmoid fn
    return a_out
```

~~def sequential(a, W, b, n):~~
~~for i in range(n):~~
~~a = dense(x, w)~~

```
def sequential(X):
    a1 = dense(x, w1, b1)
    a2 = dense(a1, w2, b2)
    a3 = dense(a2, w3, b3)
    a4 = dense(a3, w4, b4)
    return a4
```

$\vec{w}_1^{[1]} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $\vec{w}_2^{[1]} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}$, $\vec{w}_3^{[1]} = \begin{bmatrix} 5 \\ -6 \end{bmatrix}$

⇒ W = np.array([
[1, -3, 5],
[2, 4, -6]]) 2 by 3

$b_1^{[1]} = -1$, $b_2^{[1]} = 1$, $b_3^{[1]} = 2$

⇒ b = np.array([-1, 1, 2])

$\vec{a}^{[0]} = \vec{x}$

⇒ a_in = np.array([-2, 4])

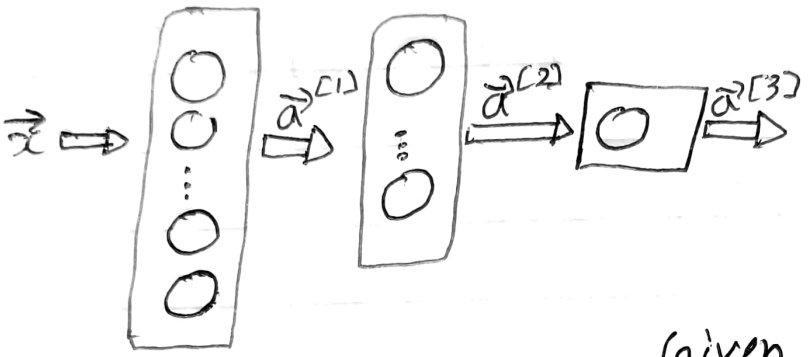
⇒ sequntial(a_in) ↵

Now we know how to implemeat forward prop from scrach.

Neural Network Training

week=2

(i) Train a Neural Network in Tensorflow



Given set of (X, Y) examples.

1 is 1 or not?

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequencial ([
        layers.Dense(units=25, activation="sigmoid"),
        layers.Dense(units=15, activation="sigmoid"),
        layers.Dense(units=1, activation ="sigmoid")
    ])
model.compile(loss=tf.keras.losses.BinaryCrossentropy())
model.fit(X, Y, epochs=100)
```

(ii) Model Training Steps

	Logistic Regression	Neural Network

① specify how to compute output given input x & parameter w, b (define model) $f_{\vec{w},b}(\vec{x}) = ?$	`z = np.dot(x, w) + b`
	`f_x = 1/(1+np.exp(-z))`
		`model = Sequential([`
		`Dense(...),`
		`Dense(...)`
		`Dense(...)])`

② specify loss & cost
Loss = $L(f_{\vec{w},b}(\vec{x}), y)$
cost = $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} L(f(\vec{x}^{(i)}), y^{(i)})$
	`loss = -y * np.log(f_x)`
	`-(1-y) * np.log(`
	`1-f_x)`
		`model.compile(`
		`loss = ...()`
		`)`

③ Train on data to minimize the cost function, $J(\vec{w}, b)$	`w = w - alpha * dj_dw`
	`b = b - alpha * dj_db`
		`model.fit(`
		`x, y, epochs=100`
		`)`

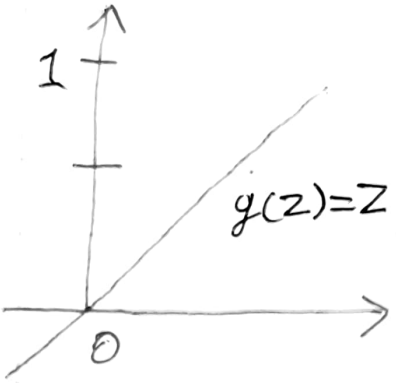
$$L = -y\log(f(\vec{x})) - (1-y)\log(1-f(\vec{x}))$$

same ↕ `BinaryCrossEntropy()` loss for neural network.

Activation Functions

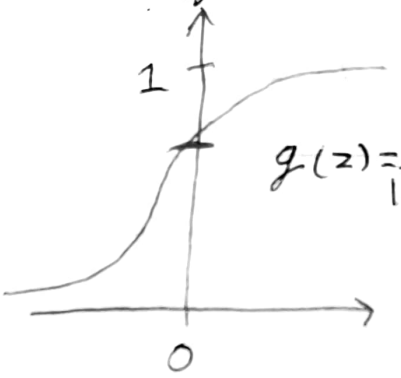
(i) Alternatives to the sigmoid activation

Linear Activation Function



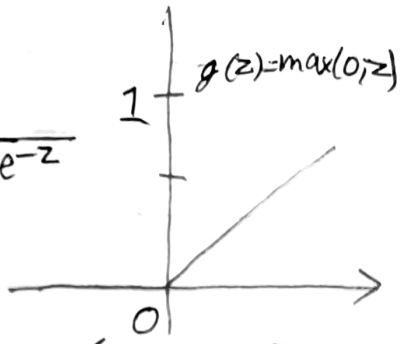
$a = g(z) = \vec{w} \cdot \vec{x} + b$

sigmoid



$0 < g(z) < 1$

ReLU



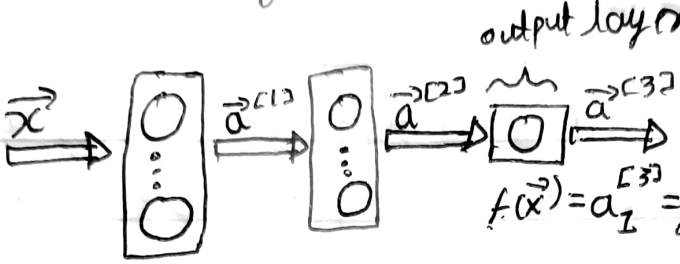
if $z < 0$ $g(z) = 0$

if $z \geq 0$ $g(z) = z$

Note:- ReLU activation function is widely used as it has a much lower run time, & run much faster than any sigmoid function. but "ReLU function should only be used in the hidden layers."

(ii) choosing activation functions

Output Layer



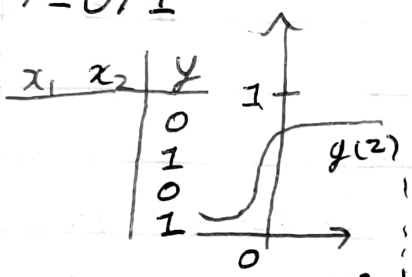
Choosing $g(z)$ for output Layer?

Binary classification

⇒ sigmoid

$Y = 0/1$

x_1	x_2	y
		0
		1
		0
		1



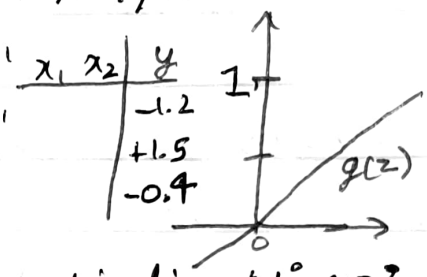
activation = 'sigmoid'

Regression

Linear activation f^n

$Y = +/-$

x_1	x_2	y
		-1.2
		+1.5
		-0.4



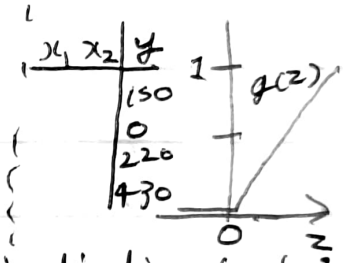
activation = 'linear'

Regression

ReLU

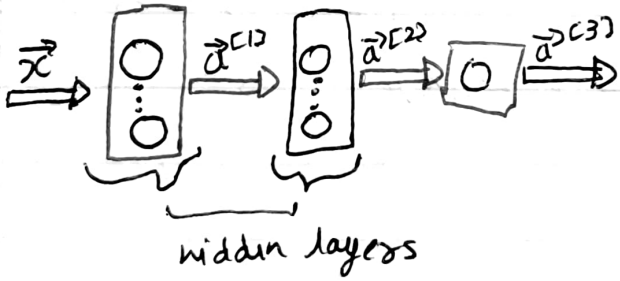
$Y = 0$ or $+$

x_1	x_2	y
		150
		0
		220
		430



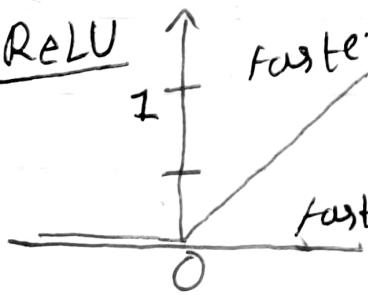
activation = 'relu'

Hidden Layer



Choosing $g(z)$ for hidden Layers?

⇒ **ReLU**



$g(z) = \max(0, z)$

faster learning

faster

more commonly choice rather than other activation function

Multiclass Classification

Softmax

Logistic Regression, $y=0$ or 1 (2 possible output values)

$$a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x}) = 0.71$$

$$\bigcirc = 1 - a_1 = P(y=0|\vec{x}) = 0.29$$

Softmax Regression, $y=1,2,3,4$ (4 possible output values)

$\times$ $z_1 = \vec{w}_1 \cdot \vec{x} + b_1$,

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} \quad (\times\ \bigcirc\ \square\ \triangle)$$

$$= P(y=1|\vec{x}) = 0.30$$

softmax Regression, $y=1,2,\ldots,N$ (N possible outputs)

$$z_j = \vec{w}_j \cdot \vec{x} + b_j \quad , j=1,2,\ldots,N$$

parameters $w_1, w_2, \ldots, w_N$
$b_1, b_2, \ldots, b_N$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y=j|\vec{x})$$

note: $a_1 + a_2 + a_3 + \cdots + a_N = 1$

$\bigcirc$ $z_2 = \vec{w}_2 \cdot \vec{x} + b_2$,

$$a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

$$= P(y=2|\vec{x}) = 0.20$$

$\square$ $z_3 = \vec{w}_3 \cdot \vec{x} + b_3$,

$$a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

$$= P(y=3|\vec{x}) = 0.15$$

$\triangle$ $z_4 = \vec{w}_4 \cdot \vec{x} + b_4$,

$$a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

$$= P(y=4|\vec{x}) = 0.35$$

Cost

Logistic regression

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

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

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

$loss = -y \log a_1 - (1-y) \log(1-a_1)$

a_2

if $y=1$: $-y \log a_1$; if $y=0$: $-(1-y) \log(1-a_1)$

$J(\vec{w}, b)$ = average loss

softmax regression

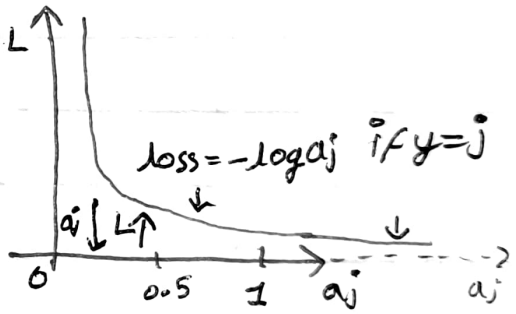
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = p(y=1|\vec{x})$$

$$\vdots$$

$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = p(y=N|\vec{x})$$

Crossentropy Loss

$$loss(a_1, \cdots, a_N, y) = \begin{cases} -\log a_1, & \text{if } y=1 \\ -\log a_2, & \text{if } y=2 \\ \vdots \\ -\log a_N, & \text{if } y=N \end{cases}$$



Decision Trees

week=3

(9) Decision Tree Model

Cat classification example

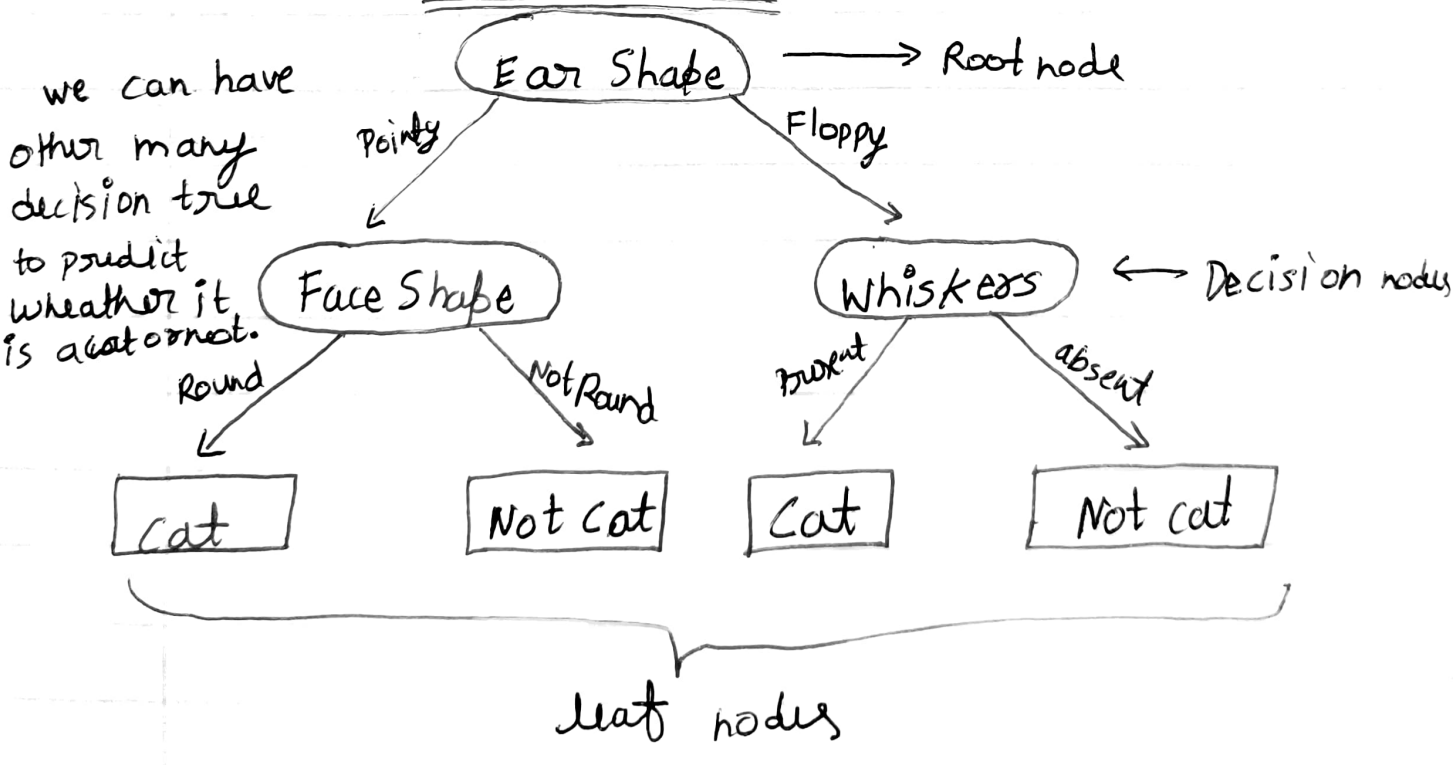
images	Ear shape (x_1)	Face Shape (x_2)	whiskers (x_3)	Cat
1img	Pointy	Round	Present	1
2img	Floppy	not Round	Present	1
3img	Floppy	Round	Absent	0
⋮	⋮	⋮	⋮	⋮

X

Y

categorical (discrete Value).

Decision Tree

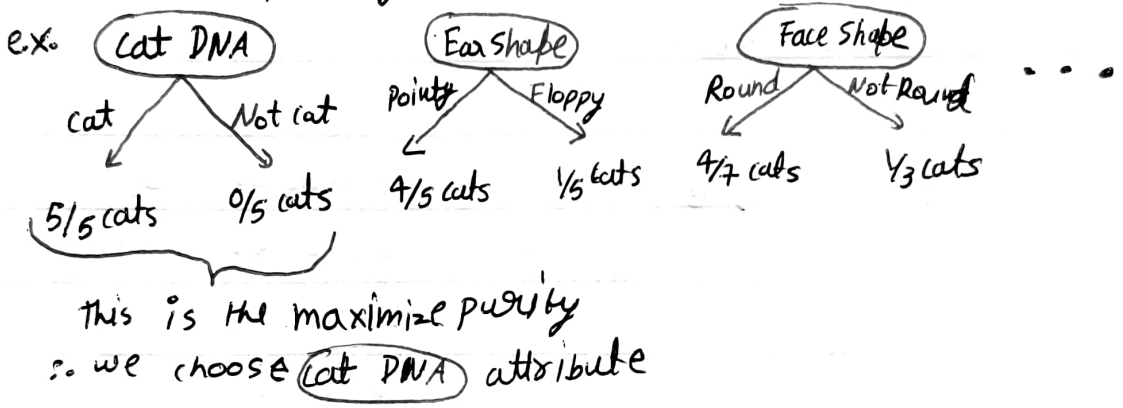


(ii) Learning Process

Decision Tree Learning

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

⇒ Maximize purity (or minimize impurity)



This is the maximize purity
∴ we choose (Cat DNA) attribute

Decision 2: When do you stop splitting?

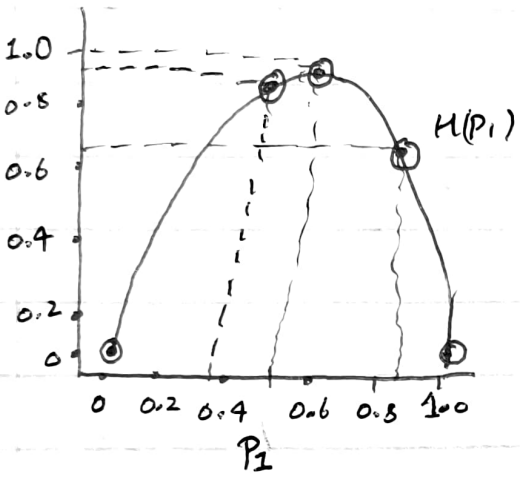
- when a node is 100% one class.
- when splitting a node will results in the exceeding a maximum depth.
- when improvements in purity score are below a threshold.
- when number of examples in a node is below a threshold.

Decision Tree Learning

(i) Measuring Purity

Entropy as a measure of impurity

P_1 = fraction of examples that are cats



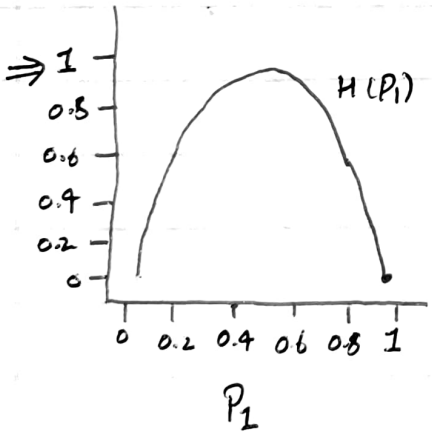
Example 1: $P_1 = 0$, $H(P_1) = 0.0$

Example 2: $P_1 = 2/6$, $H(P_1) = 0.92$

Example 3: $P_1 = 3/6$, $H(P_1) = 1.0$

Example 4: $P_1 = 5/6$, $H(P_1) = 0.65$

Example 5: $P_1 = 6/6$, $H(P_1) = 0.0$



$\Rightarrow P_0 = 1 - P_1$

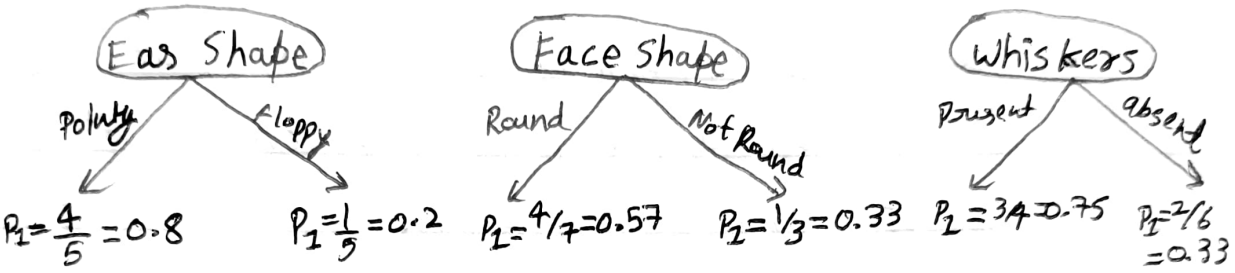
$$\Rightarrow H(P_1) = -P_1 \log_2(P_1) - P_0 \log_2(P_0)$$

$$= -P_1 \log_2(P_1) - (1-P_1) \log_2(1-P_1)$$

Note: "$0 \log(0)$" $= 0$

(ii) Choosing a split: Information Gain

Choosing a split



$P_1 = \frac{4}{5} = 0.8$ $P_1 = \frac{1}{5} = 0.2$ $P_1 = 4/7 = 0.57$ $P_1 = 1/3 = 0.33$ $P_1 = 3/4 = 0.75$ $P_1 = 2/6 = 0.33$

$H(0.5) = 1$, $H(0.8) = 0.72$ $H(0.2) = 0.72$ $H(0.57) = 0.99$ $H(0.33) = 0.92$ $H(0.75) = 0.81$ $H(0.33) = 0.92$

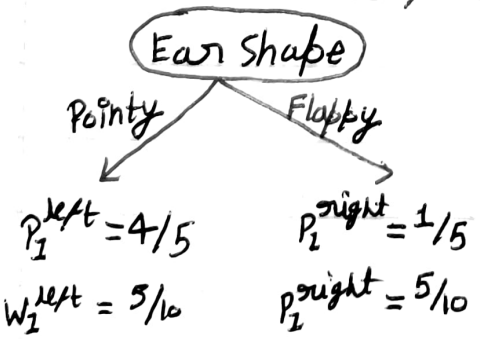
$\Rightarrow$ $H(0.5) - \left(\frac{5}{10} H(0.8) + \frac{5}{10} H(0.2)\right)$ $H(0.5) - \left(\frac{7}{10} H(0.57) + \frac{3}{10} H(0.33)\right)$ $H(0.5) - \left(\frac{4}{10} H(0.75) + \frac{6}{10} H(0.33)\right)$

$= 0.25$ (Reduction Rate of Entropy) $= 0.03$ $= 0.12$

Information Gain

we choose this attribute as it has large entropy reduction rate.

Information Gain



$P_1^{left} = 4/5$ $P_1^{right} = 1/5$

$W_1^{left} = 5/10$ $P_1^{right} = 5/10$

Information Gain

$= H(P_1^{root}) - \left(W^{left} H(P_1^{left}) + W^{right} H(P_1^{right})\right)$

(iii) Putting it together

Decision Tree Learning

- start with all examples at the root node.
- Calculate "information Gain" for all possible features, & pick the one with the highest information gain.
- Split dataset according to selected feature, & create left & right branches of the tree.
- keep repeating splitting process until ~~spl~~ stopping criteria is met:
  - when a node is 100% one class.
  - when splitting a node will result in the tree exceeding a maximum depth.
  - Information gain from additional splits is less than threshold.
  - when number of examples in a node is below a threshold.

⇒ So, to do this we uses recursive splitting technique by which we first create left sub tree & right sub tree recursively.