Week 01

01] Neural network

*] A neural network (NN) is a type of machine learning model inspired by how the human brain works. It's made up of layers of small computing units called neurons, which process data and learn complex relationships.

*) A typical neural network has 3 parts:

1] Input layer = takes in features (like price, day of week, weather)

2] Hidden layers = process the input through mathematical operations and learn patterns

3] Output layer = gives the final prediction (like predicted demand, price or classification)

*) Each neuron performs:

$$z = w_1 x_1 + w_2 x_2 + \dots w_n x_n + b$$

and passes it through a activation function like (sigmoid) to introduce non-linearity

*] During the training, the model learns weights and biases that minimize prediction error.

Ex : Demand prediction

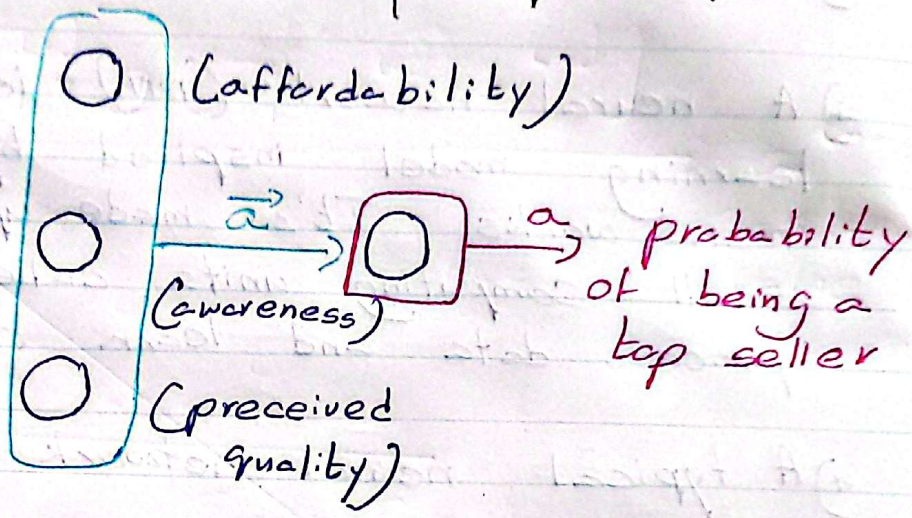
Input layer } Hidden layer } output layer

price

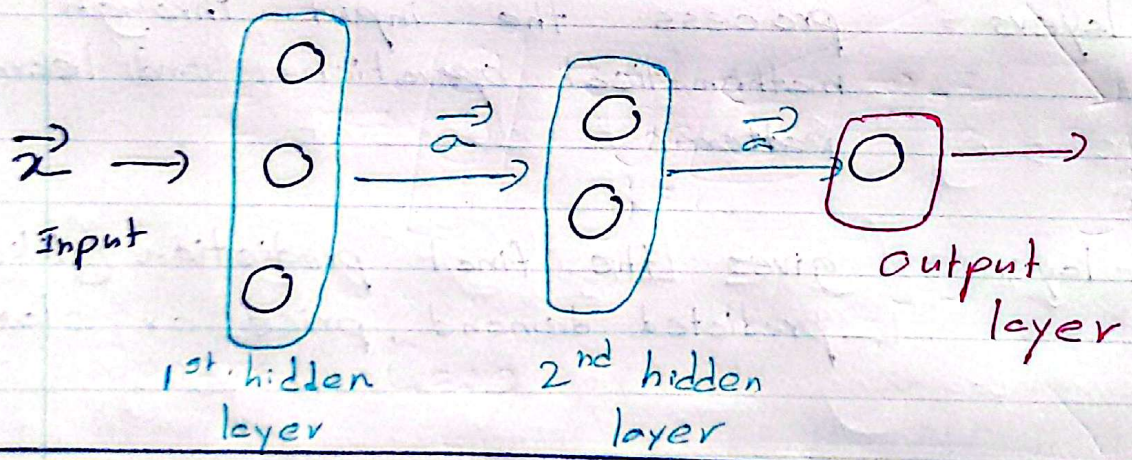
shipping cost

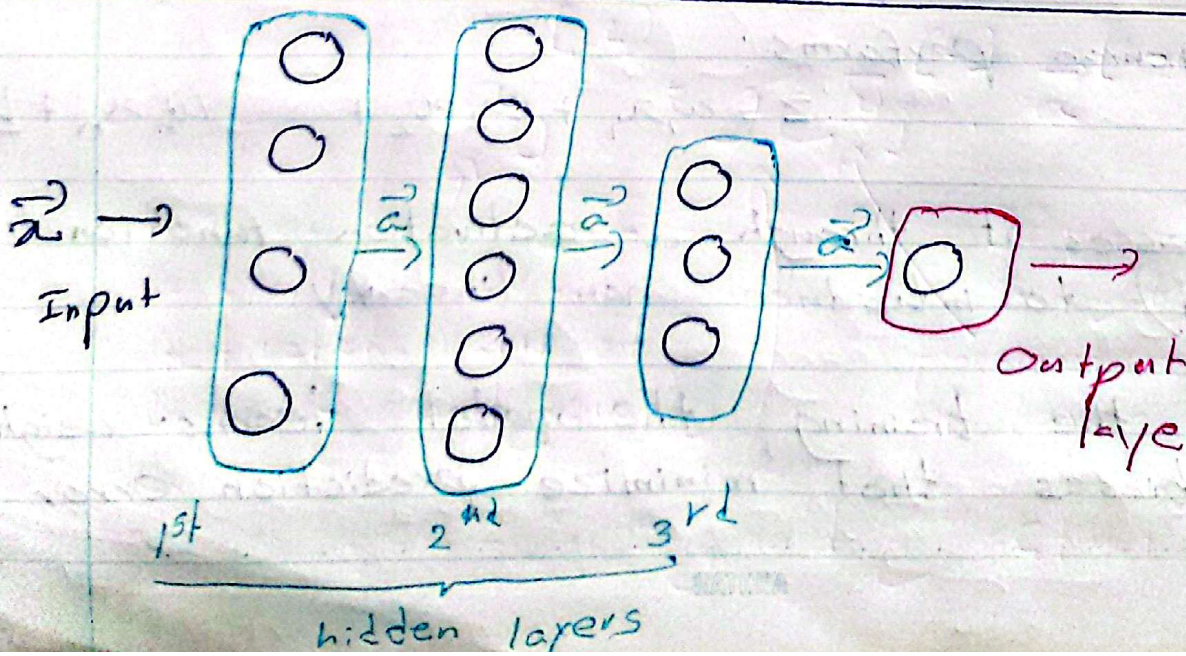
marketing

material

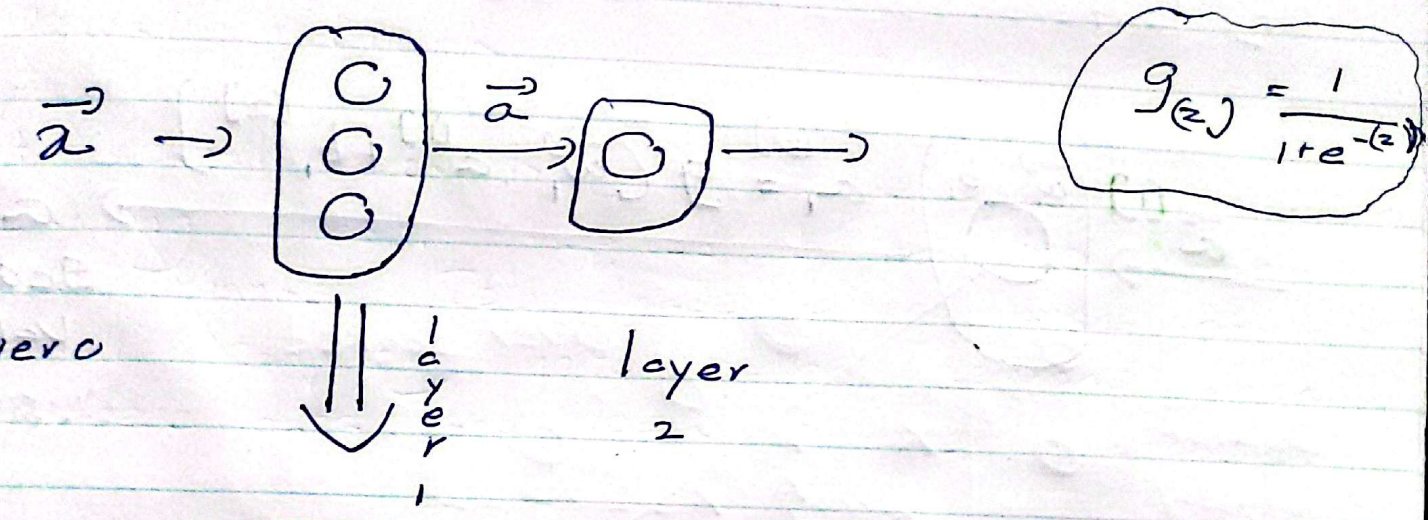


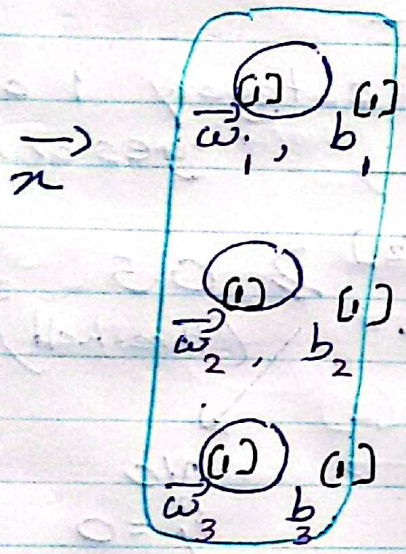
Multiple hidden layers





02] Neural Network layer





$$a_1^{[1]} = g\left(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]}\right) \quad 0.3$$

$$a_2^{[1]} = g\left(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}\right) \quad 0.7$$

$$a_3^{[1]} = g\left(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]}\right) \quad 0.2$$

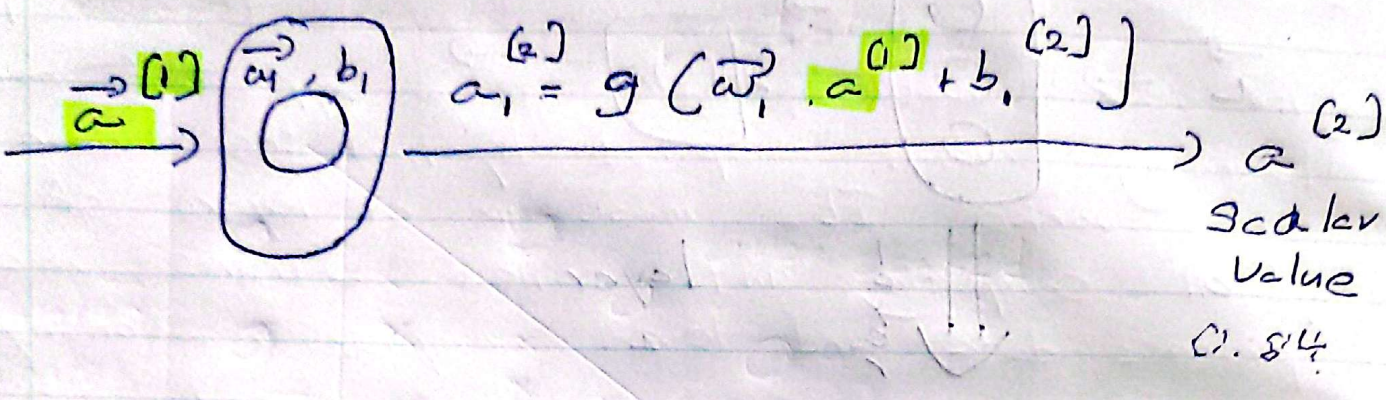
[1], [2] ... => notation for layer numbering

1, 2, 3 => notation for neuron numbering

layer 1 → output $\vec{a}$

$\begin{bmatrix} 0.3 \\ 0.7 \\ 0.2 \end{bmatrix}$ vector of activation values from layer 1

② layer 2 ⟹



③ prediction = (predict category 1 or 0 / yes or no)

$a^{[2]}$

is $a^{[2]} \geq 0.5$ (threshold)

yes $\hat{y} = 1$

No $\hat{y} = 0$

General Case

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

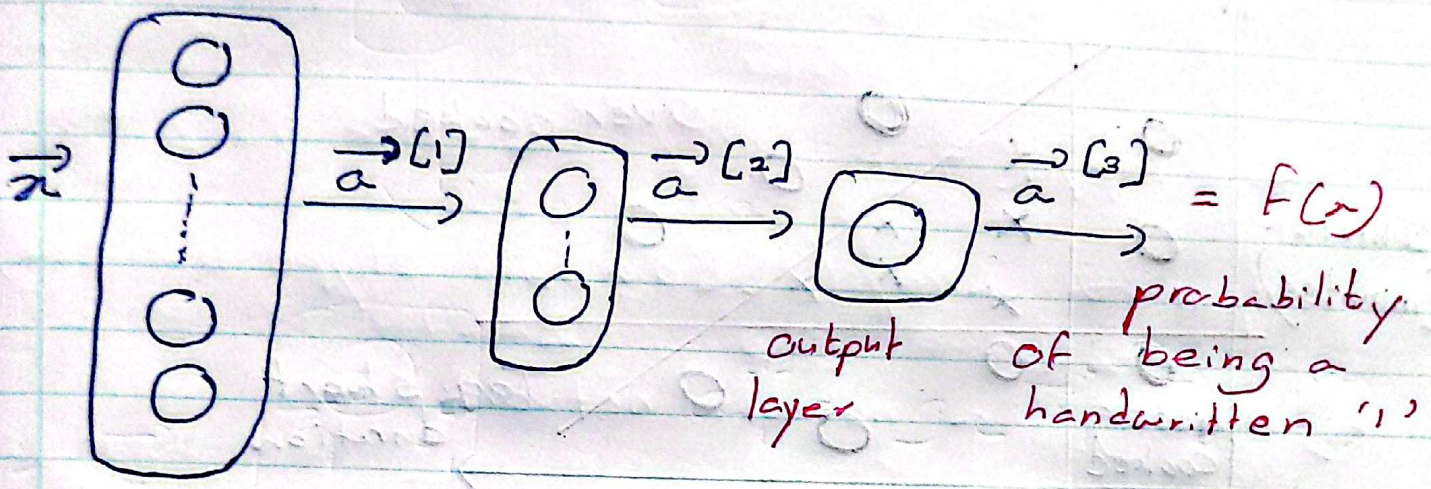
Activation value of layer l, unit (neuron) j

parameters w and b of layer l, unit j

Sigmoid ("activation function")

03] Inference (Making predictions)

Forward propagation



25 units layer 1

15 units layer 2

1 unit layer 3

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

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

yes

$\hat{y} = 1$

(image is digit 1)

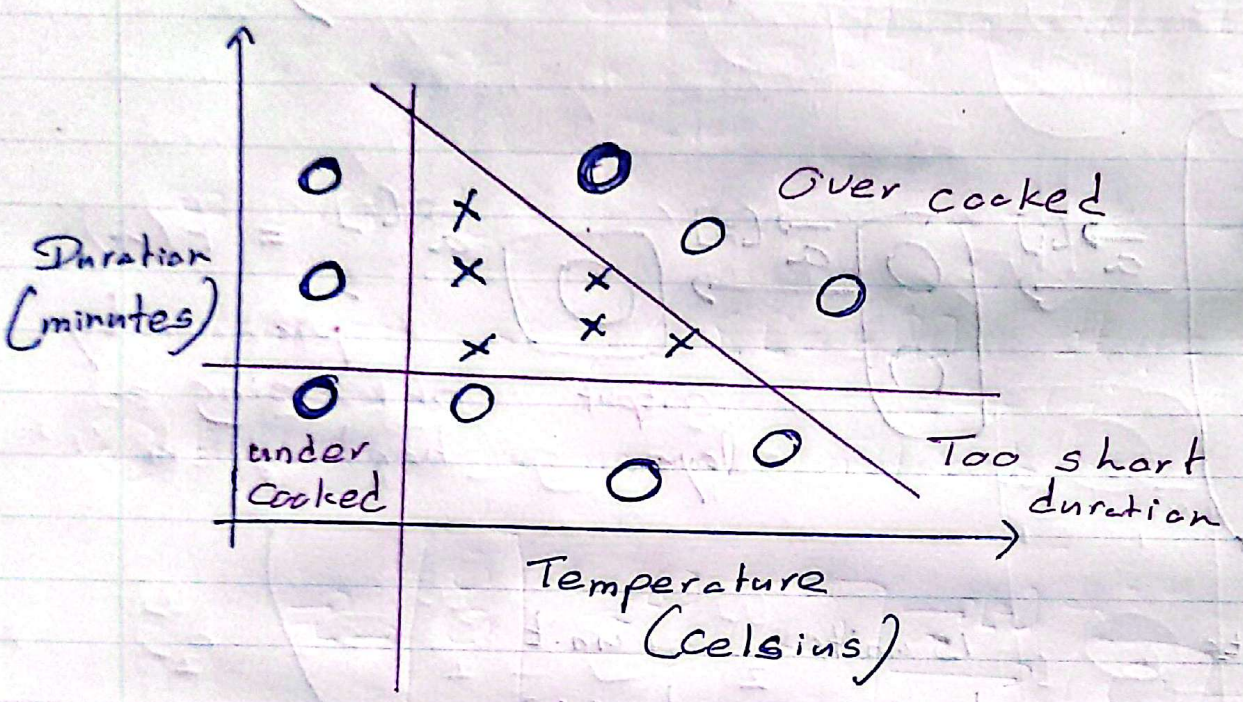
no

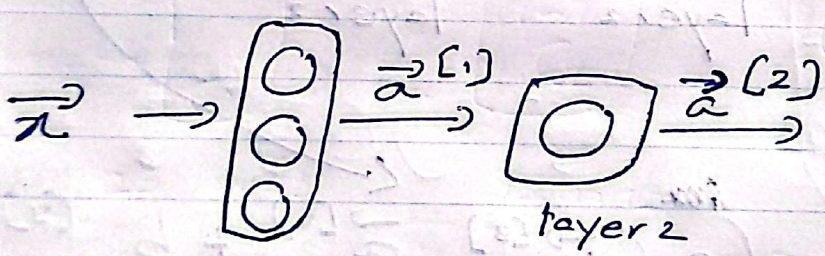
$\hat{y} = 0$

(image isn't digit 1)

04] Inference in Code

Coffe roasting prediction





is $a_1^{[2]} \geq 0.5$?

$\hat{y} = 1$ (X)

$\hat{y} = 0$ (O)

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

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

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

05] Data in Tensorflow

Numpy arrays

x = np.array([[200, 17]]) ⇒ [200, 17] 1x2

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

"2D Arrays"

x = np.array([200, 17]) } "1D vector"

Activation vector

Feature vector

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

output → tf.Tensor([[0.2 0.7 0.3]], shape(1, 3), dtype = float32)

(1x3 matrix)

a1.numpy

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

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

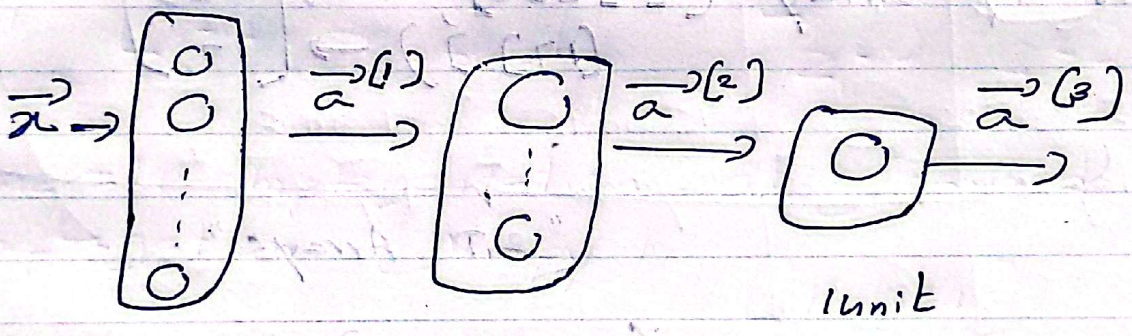
output → tf.Tensor([[0.8]], shape(1,1), dtype = float32)

1x1 matrix
Scalar

```
a2.numpy()
```

output → array([[0.8]], dtype=float32)

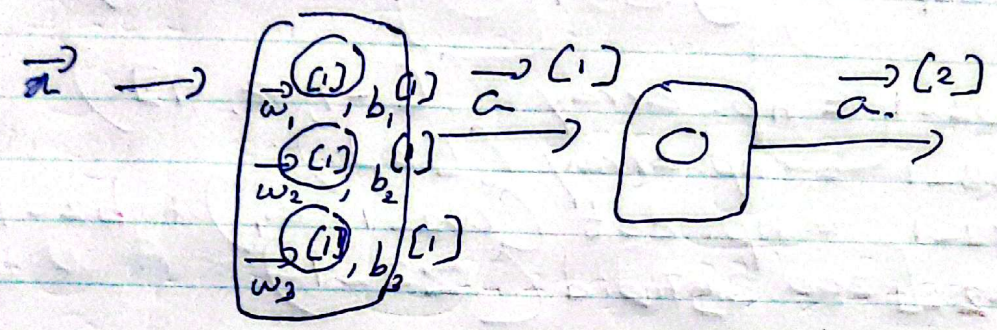
06) Building a neural network




```
layer_1 = Dense(units=25, activation='sigmaid')
layer_2 = Dense(units=15, activation='sigmaid')
layer_3 = Dense(units=1, activation='sigmaid')

model = sequential([layer_1, layer_2, layer_3])

model.compile(...)

x = np.array([[0..., 245, ..., 17], [0..., 200, ... 184]])
y = np.array([1, 0])
```

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

07] Forward propagation in a Single layer



x = np.array([200, 17])

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

```
w1_1 = np.array([1, 2])
b1_1 = np.array([-1])
z1_1 = np.dot(w1_1, x) + b1_1
a1_1 = sigmoid(z1_1)
```

$a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]})$

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

$a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$

⋮

```
a1_3 = sigmoid(z1_3)
```

$a_1 = np.array([a_{1-1}, a_{1-2}, a_{1-3}])$

Second layer (output layer)

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

```
w2_1 = np.array([[-7, 8, 9]])
b2_1 = np.array([3])
z2_1 = np.dot(we_1, a_1) + b2_1
a2_1 = Sigmoid(z2_1)
```

$w_1^{[2]} = w2_1$

08) General implementation of forward propagation (Numpy)

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

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

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

```
b = np.array([-1, 1, 2])
a_in = np.array([-2, 4])
```

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

```
def dense(a_in, W, b):
    units = W.shape[1]
    a_out = np.zeros(units)        [0, 0, 0]
    for j in range(units):         0, 1, 2
        w = W[:, j]
        z = np.dot(w, a_in) + b[j]
    a_out[j] = g(z)
return a_out
```

```
def sequential(x):
    a1 = dense(x, W1, b1)
    a2 = dense(a1, W2, b2)
    a3 = dense(a2, W3, b3)
    a4 = dense(a3, W4, b4)
    f_x = a4
return f_x
```

} 4 layers

Capital W refers to matrix

Instead of this for more efficient

all 2d arrays

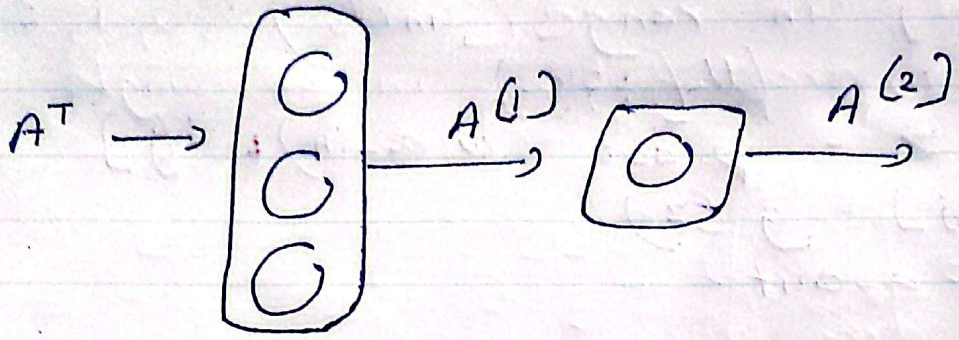
```
def dense(A_in, W, B)
    Z = np.matmul(A_in, W) + B
    A_out = g(Z)
return A_out
```

Vectorized

matrix multiplication

[[1, 0, 1]]

09] Vectorized Code



$A^T = [200 \quad 17]$

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

$B = [-1 \quad 1 \quad 2]$

$Z = A^T W + B$

$[165 \quad -531 \quad 900]$

$Z_1^{[1]} \quad Z_2^{[1]} \quad Z_3^{[1]}$

$A = g(Z)$

$[1 \quad 0 \quad 1]$

(A Transpose)

```
AT = np.array([[[200, 17]]])
W = np.array([[[1, -3, 5],
               [-2, 4, -6]]])
b = np.array([[[-1, 1, 2]]])

def dense(AT, W, b):
    Z = np.matmul(AT, W) + b
    a_out = g(Z)
    return a_out
```

$[[1, 0, 1]]$