We derived results for 1 example.

no. of elements in z_3 is equal to no. of output classes.

output vector given by z_3

$$\begin{bmatrix} 20 \\ 20 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

→ Probability of Dog is 0.5

→ Probability of cat is 0.5

Now we want to make predictions for m examples.

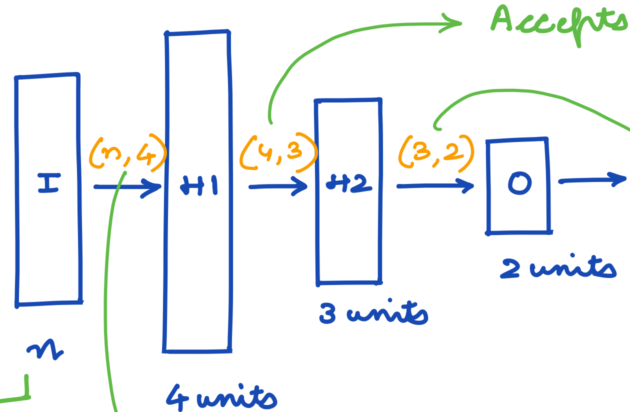
$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{(n,1)} \qquad X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots\cdots & x_n^{(1)} \\ \hline & & x^{(i)} & & \end{bmatrix}_{m \times n}$$

$$A^{[0]} = X$$

$$Z_1 = W^{[1]} \cdot A^{[0]} + b$$

Activation of 0^{th} layer

Accepts 4 features and produces 3 outputs.



Accepts 3 features and produces 2 outputs. If you have k output classes then it will of size 3Xk

n nodes to handle n features from a given example

just a linear transformation accepts n features and produces 4 outputs.

$$Z_1 = \underbrace{\underset{(m,n)}{A^{[0]}} \cdot \underset{(n,4)}{W^{[1]}}}_{(m,4)} + b^{[1]}$$

For 1 example you were reducing n features to 4.

For m examples now you are reducing all features to 4.

$$\underset{m \times n}{\begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & \cdots & x_n^1 \\ x_1^2 & x_2^2 & x_3^2 & \cdots & x_n^2 \\ \vdots & \vdots & \vdots & & \vdots \end{bmatrix}} \underset{n \times 4}{\begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2n} \\ \vdots & \vdots & \vdots & & \vdots \end{bmatrix}}$$

$$z_1^{[1]} = x_1^1 w_{11} + x_2^1 w_{21} + \cdots\cdots + x_n^1 w_{n1} \rightarrow$$ you are multiplying features with some weights

$$Z^{[1]} = \underset{m \times 4}{\begin{bmatrix} z_1^{[1][1]} & z_2^{[1][1]} & z_3^{[1][1]} & z_4^{[1][1]} \\ z_1^{[1][2]} & z_2^{[1][2]} & z_3^{[1][2]} & z_4^{[1][2]} \\ & & & \end{bmatrix}} + b^{[1]}$$

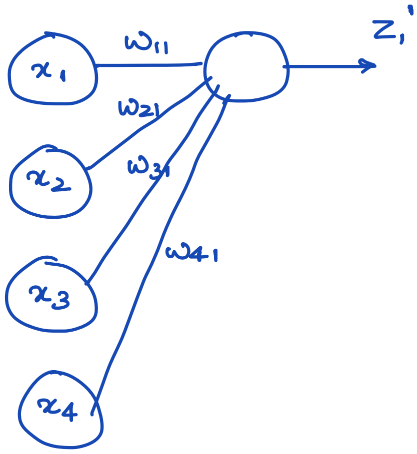
output of 2^{nd} neuron in first layer.

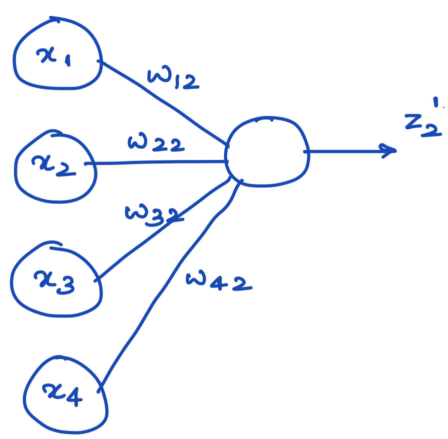
$$Z_j^{[l][i]}$$

layer no

example no

neuron no.

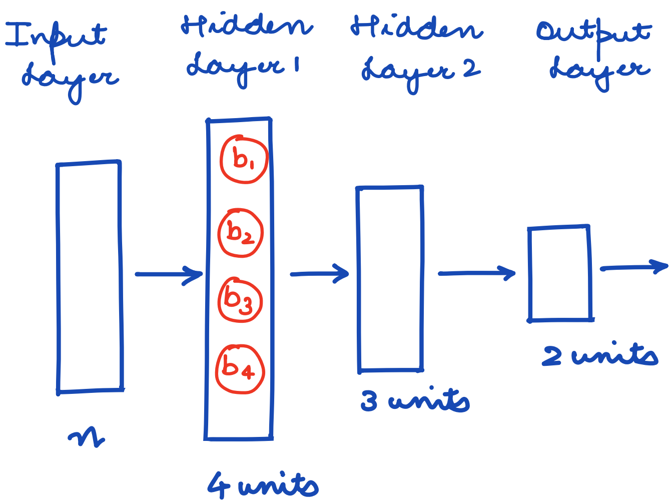




$$A^{[1]} = \sigma \begin{bmatrix} & | & \\ — & Z^{[1]} & — \\ & | & \end{bmatrix}$$

Activation fx^n

now lets talk about bias :



4 neurons
each neuron
will have 1 value
of bias.
Bias vector will
be of size 1×4

$$Z^{[1]} = \begin{bmatrix} z_1^{[1][1]} & z_2^{[1][1]} & z_3^{[1][1]} & z_4^{[1][1]} \\ z_1^{[1][2]} & z_2^{[1][2]} & z_3^{[1][2]} & z_4^{[1][2]} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}_{m \times 4} + b^{[1]}_{1 \times 4} \qquad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

broadcasting will be done automatically and $b^{[1]}$ will be added to all the values

$$Z^{[1]} = \begin{bmatrix} z_1^{[1][1]} + b_1 & z_2^{[1][1]} + b_2 & z_3^{[1][1]} + b_3 & z_4^{[1][1]} + b_4 \\ z_1^{[1][2]} + b_1 & z_2^{[1][2]} + b_2 & z_3^{[1][2]} + b_3 & z_4^{[1][2]} + b_4 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}_{m \times 4}$$

CODE:

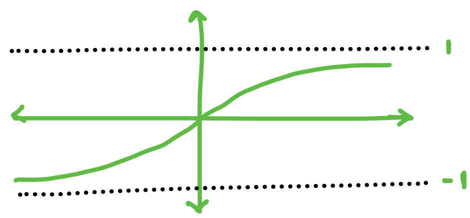
```
def forward(self, x):
    w1, w2, w3 = self.model['w1'], self.model['w2'], self.model['w3']
    b1, b2, b3 = self.model['b1'], self.model['b2'], self.model['b3']

    z1 = np.dot(x, w1) + b1
    a1 = np.tanh(z1)
    z2 = np.dot(a1, w2) + b2
    a2 = np.tanh(z2)

    z3 = np.dot(a2, w3) + b3
    y_ = softmax(z3)
```

unpack the values from dictionary

we don't have sigmoid function available directly so, we can use tanh. tanh is like sigmoid and it sequeezes your output in range -1 to 1.



we will define softmax on own.

```
def softmax(a):
    e_pa = np.exp(a)
    ans = e_pa/np.sum(e_pa, axis=1, keep dims = True)
    return ans
```

```
a = np.array([5,1,2])
np.exp(a)
```

```
array([148.4131591 ,   2.71828183,   7.3890561 ])
```

$\frac{e^5}{e^5+e^1+e^2}$ $\quad$ $\frac{e^1}{e^5+e^1+e^2}$ $\quad$ $\frac{e^2}{e^5+e^1+e^2}$

```
a = np.array([[5,1,2],[6,1,2]])
np.sum(a)
```

```
17
```

$5+1+2+6+1+2$

```
a = np.array([[5,1,2],[6,1,2]])
np.sum(a, axis=1)
```

```
array([8, 9])
```

$$\begin{array}{c} 8 \\ \begin{pmatrix} 5 & 1 & 2 \\ 6 & 1 & 2 \end{pmatrix} \\ 9 \end{array} \quad 2\times 3$$

axis 0 $\quad$ axis 1

Earlier it was 2D matrix
but output is a single matrix

```
a = np.array([[5,1,2],[6,1,2]])
np.sum(a, axis=1, keepdims=True)
```

```
array([[8],
       [9]])
```

Now the output is a 2D Array
It preserves the shape

$a = [1, 2, 3]$

$$\frac{e^{a_i}}{\Sigma e^{a_i}}$$

$$\frac{e^1}{e^1+e^2+e^3} \quad \frac{e^2}{e^1+e^2+e^3} \quad \frac{e^3}{e^1+e^2+e^3}$$

It will produce a vector

$$\sum_{i=1}^{n}\left(\frac{e^{a_i}}{\sum_{i=1}^{n} e^{a_i}}\right) = 1$$

$$\frac{e^1}{e^1+e^2+e^3} + \frac{e^2}{e^1+e^2+e^3} + \frac{e^3}{e^1+e^2+e^3} = 1$$

```python
def softmax(a) :
    e_pa = np.exp(a)
    ans = e_pa / np.sum(e_pa, axis=1, keepdims=True)
    return ans
```

```python
class NeuralNetwork :

    def __init__(self, input_size, layers, output_size) :

        np.random.seed(0)

        model = {} # Dictionary

        # First Layer
        model['W1'] = np.random.randn(input_size, layers[0])
        model['b1'] = np.zeros((1, layers[0]))

        # Second Layer
        model['W2'] = np.random.randn(layers[0], layers[1])
        model['b2'] = np.zeros((1, layers[1]))

        # Third Layer
        model['W3'] = np.random.randn(layers[1], output_size)
        model['b3'] = np.zeros((1, layers[2]))

        self.model = model

    def forward(self,x) :

        W1,W2,W3 = self.model['W1'], self.model['W2'], self.model['W3']
        b1,b2,b3 = self.model['b1'], self.model['b2'], self.model['b3']

        z1 = np.dot(x,W1) + b1
        a1 = np.tanh(z1)

        z2 = np.dot(a1,W2) + b2
        a2 = np.tanh(z2)

        z3 = np.dot(a2,W3) + b3
        y_ = softmax(z3)
```

Use of Softmax:

a = np.array([[10, 10], [20, 20]])

a_ = softmax(a)

print(a_)

```python
a = np.array([[10,10],[20,20]])
a_ = softmax(a)
print(a_)
```

```
[[0.5 0.5]
 [0.5 0.5]]
```

a = np.array([[10, 20]])

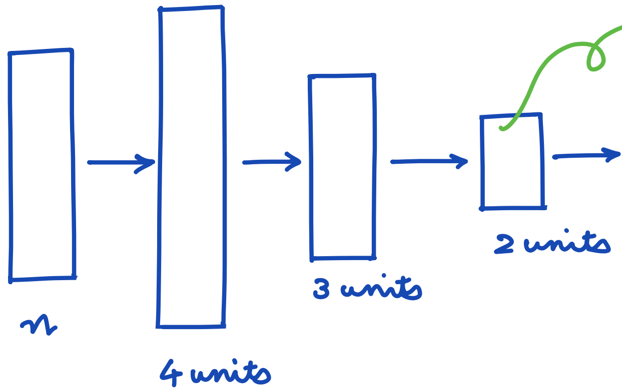
a_ = softmax(a)

print(a_)

```
a = np.array([[10,20]])
a_ = softmax(a)
print(a_)
```

```
[[4.53978687e-05 9.99954602e-01]]
```

Probability is high for value 20 and low for value 10.



For 1 example, last layer will produce an output of size 2.

For m examples, last layer will produce output of size m x 2

$$\text{Output } z_3 = \begin{bmatrix} 10 & 10 \\ 20 & 30 \end{bmatrix}_{m \times k}$$

Output for 1st example. If I pass it through softmax I will get 0.5 and 0.5 that means there is equal probability that output belongs to class 1 & class 2

no. of classes

k = 2

Output for 2nd example.
If I pass it through softmax,
you might get 0.1 and 0.9
So, there is 90% probability
that it belongs to class 2
because it has high activation value

$$\text{output of softmax} = \begin{array}{cc} \text{Dog} & \text{Cat} \\ \end{array}$$

$$\begin{bmatrix} 0.5 & 0.5 \\ 0.1 & 0.9 \end{bmatrix}$$

In this case clearly cat has a higher probability
