CHAPTER 2 — Coding our First Neuron

A Single Neuron →

$$output = \Sigma(weight \times input) + bias$$

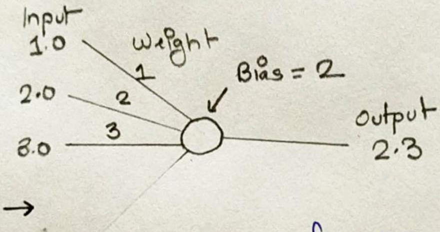
Inputs = [1,2,3]

Each input has a weight associated with it. At our network weight are actually initialized randomly and biases are set to zero / some number.

inputs = [1, 2, 3] ; weight = [0.2, 0.8, -0.5] . bias = 2 :

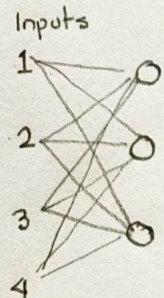
→ We are modeling more one neuron/single neuron, so only one bias.
Bias is an additional tunable value but is not associated with any input

→ output = (input [0] * weight [0] + input[1] * weight [1] + input [2] * weight [2]) + bias
= (1 × 1 + 2 × 2 + 3 × 3) + 2 = 2.3



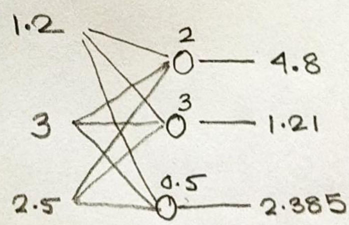
A layers of neuron →

- Neural network typically have layers of neuron that consist more than one neuron.
- Layers are nothing but group of neurons.
- Each neuron in a layer takes exactly same input.
- Input can be training set or output data of last layer but contains its own set of weight & own bias producing its own unique output.



Layers of neuron with common input

Logically,
inputs = [1.2, 3, 2.5]
weight 1 = [0.2, 0.8, -0.5, 1]
weight 2 = [0.5, -0.91, 0.26, -0.5]
weight 3 = [-0.26, -0.27, 0.17, 0.87]
bias 1 = 2, bias 2 = 3, bias 3 = 0.5



outputs = [#neuron 1,
input[0] × weight 1[0] + input[1] × weight 1[1]
+ input [2] × weight 1[2] + input[3] × weight[3] + bias 1,
#neuron 2,
input [0] × weight 2[0] + input[1] × weight 2[1] + input [2] × weight 2[2] + input [3] × weight 2[3] + bias[2]
neuron 3,
input[0] × weight 3 [0] + input[1] × weight 3[1] + input [2] × weight 3[2] + input [3] × weight 3[3]) + bias[3]]

print(output) → [4.8, 1.21, 2.385]

02.12 - In last example, we have three set of weights, three biases three neurons. Each neuron is connected on same inputs. The difference is in weight & bias. This is called Fully connected Neural Network - every neuron is connected layers has connection to every neuron from previous layers.

Tensors, Arrays and Vectors

Tensors →

- Tensors are closely related to arrays.
- A tensor object is an object that can be represent as an array.

Example of 3 dimensional Array.

Array

Example = [[[1,5,6,2], [3,2,1,3]],
[[5,2,1,2], [6,4,8,4]],
[[2,8,5,3], [1,1,9,4]]]

Shape - (3, 2, 4) Type - 3D Array
3 dimension (row) ; 2 list ; 4 elements (column)

. It contains 3 dimension, 2 list each and each list contain 4 elements
- 3 rows, 2 list, 4 columns.

Another Example = [[4,2],
[5,1],
[8,2]]

Shape - (3,2) it has 3 rows. 2 columns.
If there is "1" list in each column we do not mention.
Type - 2D Array.

Homologous Array → When number of rows = Columns. Above two example is non homologous.
Because first example is - 3 X 4 (rows, columns), second example is 3 X 2.

Example of homologus Array - [[1,2]
[3,4]] → 2X2

- One dimensional array is also known as List. Eg [1,2,3].

Dot products and Vector Addition →
- Multiply each element in our inputs and weight vectors element wise by using dot product
- Dot product of two vectors, $\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a.b = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$

A dot product of two vectors is a sum of products of consecutive vector elements.
Both vectors must be of same size (have equal number of elements).

Eg → a = [1,2,3]   b = [2,3,4]   dot_product = a[0]*b[0] + a[1]*b[1] + a[2]*b[2]
print (dot_product) # 20.

So consider a as input and b = weight. So that is what we do till now.
Therefore dot product exactly do what we needed.

- Addition of the two vectors is an operation performed element wise, which means that both vectors have to be same size.

$$\vec{a} + \vec{b} = [a_1+b_1, a_2+b_2, a_3+b_3, \ldots, a_n+b_n]$$

A Single Neuron with NumPy

Output = Σ(weight × input) + bias

```
import numpy as np
inputs = [1.0, 2.0, 3.0, -2.5]
weights = [0.2, 0.8, -0.5, 1.0]
bias = 2.0
outputs = np.dot(weights, inputs) + bias
print (outputs)  # 4.8
```

A Layer of Neuron with NumPy

- Calculate the output of a layer with 3 neurons, which means the weight will be in matrix or a list of weight vectors.
  In plain python, we wrote list of List.
  In Numpy, this will be 2 dimensional array called Matrix.

Previously,
```
inputs = [1.0, 2.0, 3.0, 2.5]
weights = [[0.2, 0.8, -0.5, 1.0],
           [0.5, -0.91, 0.26, -0.5], [-0.26, -0.27, 0.17, 0.87]]
biases = [2.0, 3.0, 0.5]
```

```
np.dot(weights, inputs) = (np.dot(weight[0], inputs), np.dot[(weight[1], inputs)],
                          (np.dot(weight[2], inputs)]
# [2.8, -1.79, 1.885]
output = np.dot(weigh, inputs) + biases   # array([4.8, 1.21, 2.385])
```

Now, directly we can write
```
layer_outputs = np.dot(weights, inputs) + biases
print (layer_outputs)  # array([4.8, 1.21, 2.385])
```

Batch of Data →

Some we receive, one sample / one observation – [1, 2, 3, 3.5]

two samples / two observation – [[1, 2, 3, 3.5],
[4, 1, 2, 4]]

So, we get input data in batches to help in generalizing the model.

Sample Input Data. batch = [[1, 5, 6, 2],
[3, 2, 1, 3],
[4, 6, 7, 4],
[5, 6, 2, 3],
[2, 7, 1, 3]].

Shape = (5, 4)

Type = 2D Array, Matrix

Row × Column.

So we reach batch of data.
One row is one list.

Sample Dataframe.

Features / Records	F_1	F_2	F_3	F_4
1	1	5	6	2
2	3	2	1	3
3	4	6	7	4
4	5	6	2	3
5	2	7	1	3

02.4 Matrix Product

- Matrix product is an operation in which we have 2 matrices
- We dot products of all the combinations of rows from the first matrix and columns of 2nd matrix.
- To perform matrix product, size of second dimension must match with size of first dimension of right matrix. For eg. (5, 4) and (4, 7)
  $\underbrace{4\ \ 4}_{\text{match}}$
- Can be multiplied - (2, 3) (3, 1) $\leftarrow$ same . (4, 8) (8, 16) $\leftarrow$ multiply

Cannot be multiplied - (2, 1) (2, 1) not same (4, 8) (8, 9) not same.

Suppose $a = [1, 2, 3]$ $b = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$

1×3 (Row, Colum) 3×1

$a \cdot b$
$\cancel{a \times b} = [1\times2 + 2\times3 + 3\times4]$
$= [2+6+12] = [20]$

In other word, row and columns vectors of matrices with one of their dimension being of size 1, we perform matrix product on them instead of dot products.

Transposition of Matrix

We can also write $\vec{a} \cdot \vec{b} = ab^T$, T is nothere transposition for now.

Transposition simply modifies a matrix that row become column and vice versa.

$$\begin{bmatrix} 01 & 02 & 03 \\ 04 & 05 & 06 \\ 07 & 08 & 09 \end{bmatrix} \xrightarrow{\text{Transpose}} \begin{bmatrix} 01 & 04 & 07 \\ 02 & 05 & 08 \\ 03 & 06 & 09 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \xrightarrow{\text{Transpose}} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

With Numpy Code,

$a = [1, 2, 3]$
$b = [2, 3, 4]$

$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ 1×3 $\begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$ 3×1

```
a = np.array (a)
b = np.array ([b]).T
np.dot (a,b)  # 20.
```

which is same as above ans.
so $\vec{a} \cdot \vec{b} = ab^T$

Numpy does not have dedicated method for performing matrix product - Dot product and matrix product are both implemented in a single method : np.dot()

A layer of Neuron and Batch of Data with NumPy.

- Initially, we were able to perform dot products on the inputs and the weights without transposition because the weight were a matrix, but input was just a vector.
- When input becomes batch of input (a matrix), we need to perform matrix product.

Now we can implement what we learned from code

```
import numpy as np.
inputs = [[1.0, 2.0, 3.0, 2.5],
          [2.0, 5.0, -1.0, 2.0],
          [-1.5, 2.7, 3.3, -0.8]]          3 x 4.

weight = [[0.2, 0.8, -0.5, 1.0],
          [0.5, -0.91, 0.26, -0.5],
          [-0.26, -0.27, 0.17, 0.87]]      3 x 4.

bias = [2.0, 3.0, 0.5].
layer_output = np.dot(inputs, np.array(weights).T) + bias.
print (layer_output)   # [[4.8   1.21   2.385],
                          [8.9   -1.81   0.2],
                          [1.41   1.05   0.026]]
```

we cannot multiply $\underbrace{(3 \times 4) \cdot (3 \times 4)}_{\text{not same}}$

So transponse.

$\underbrace{(3 \times 4) \cdot (4 \times 3)}_{\text{same}}$

$$\underbrace{\begin{bmatrix} 1.0 & 2.0 & 3.0 & 2.5 \\ 2.0 & 5.0 & -1.0 & 2.0 \\ -1.5 & 2.7 & 3.3 & -0.8 \end{bmatrix}}_{\text{Input}} \underbrace{\begin{bmatrix} 0.2 & 0.5 & -0.2 \\ 0.8 & -0.9 & -0.2 \\ -0.5 & 0.2 & 0.17 \\ 1.0 & -0.5 & 0.87 \end{bmatrix}}_{\text{weight}^T} = \underbrace{\begin{bmatrix} 2.8 & -1.79 & 1.855 \\ 6.9 & -4.81 & -0.3 \\ -0.59 & -1.949 & -0.47 \end{bmatrix}}_{\text{Input} \times (\text{weight}^T)} + \underbrace{\begin{bmatrix} 2 & 3 & 5 \end{bmatrix}}_{\text{Bias}} = \begin{bmatrix} 4.8 & 1.21 & 2.3 \\ 8.9 & -1.8 & 0.2 \\ 1.41 & 1.05 & 0.02 \end{bmatrix}$$