Artificial Neural Networks

Dr. Muhammad Aqib

University Institute of Information Technology

PMAS-Arid Agriculture University Rawalpindi

Coding First Neurons

Contents

- Implementing single neuron
  - With 3 and 4 inputs
- Layer of neurons
  - Implementation of layers of neuron
- Tensors, arrays and vectors
- Dot product and vector additions
- Neurons implementation with Numpy
- Batches of data
- Matrix Product
- Matrix Transposition (Transpose)

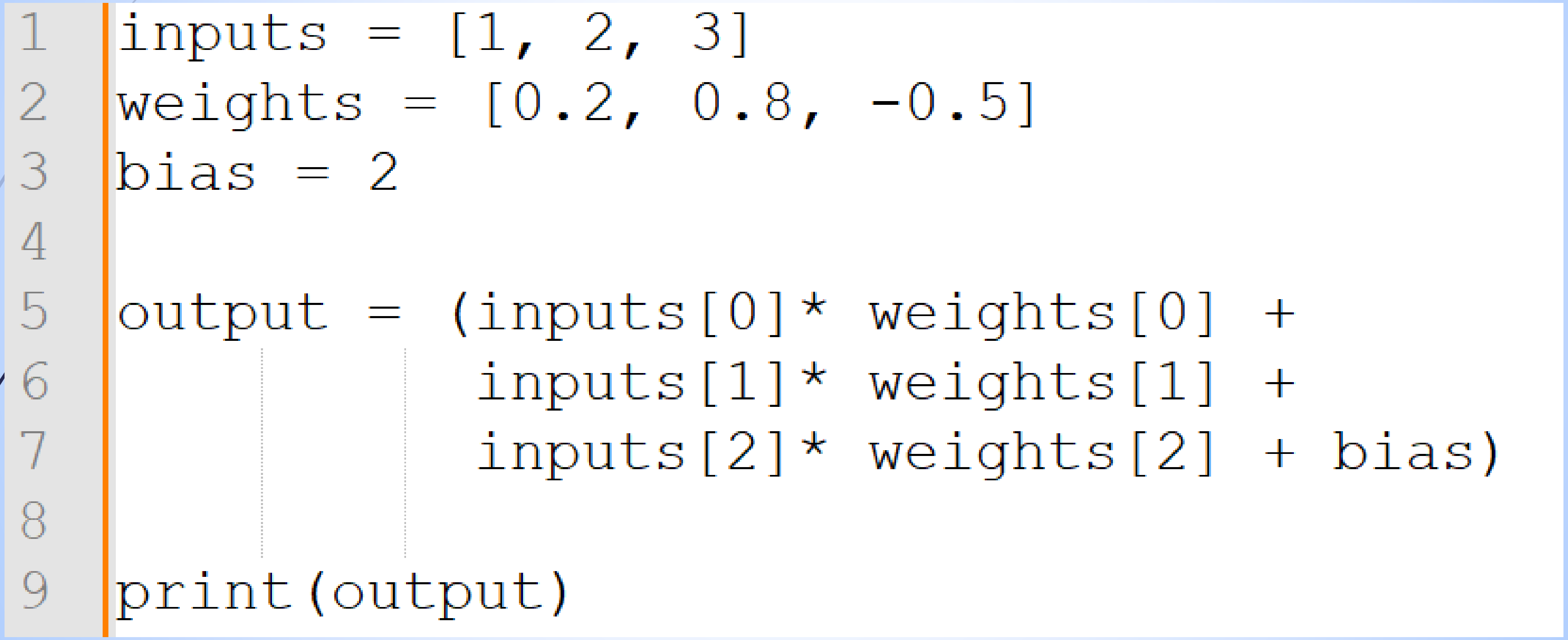
Prerequisites

- Laptop/System with Python3 installed
- Python Package Manager `pip` installed
- Libraries such as NumPy and Matplotlib

Single Neuron

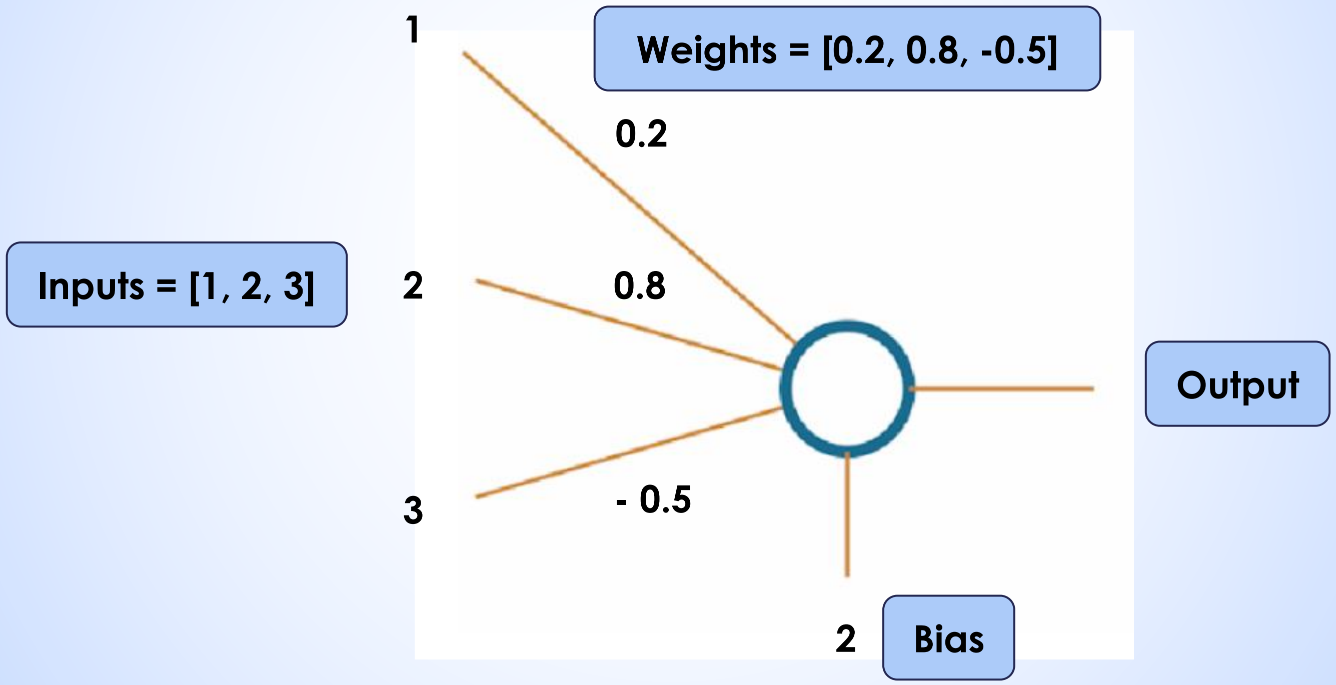
- Let’s consider a single neuron with 3 inputs.
  - inputs = [1, 2, 3] # You can consider any three input values.
- Each input gets multiplied with weights
  - weights = [0.2, 0.8, -0.5] # You can consider any three weights.
- As we are considering single neuron, and each neuron gets a single bias value so:
  - bias = 2 # You can consider any bias value.
- As already discussed, output can be calculated using given equation:
  - output = (inputs * weights) + bias

Single Neuron (cont.)



```
inputs = [1, 2, 3]
weights = [0.2, 0.8, -0.5]
bias = 2

output = (inputs[0]* weights[0] +
          inputs[1]* weights[1] +
          inputs[2]* weights[2] + bias)

print(output)
```

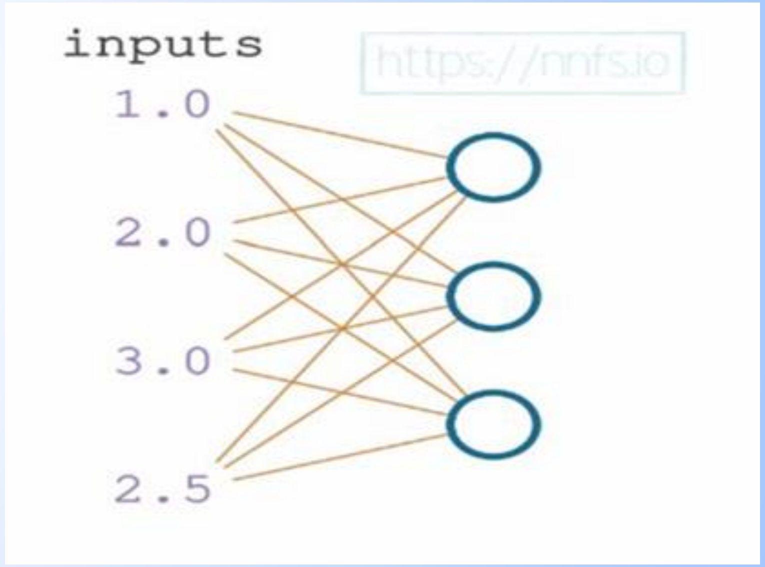
Single Neuron (cont.)



Layers of Neurons

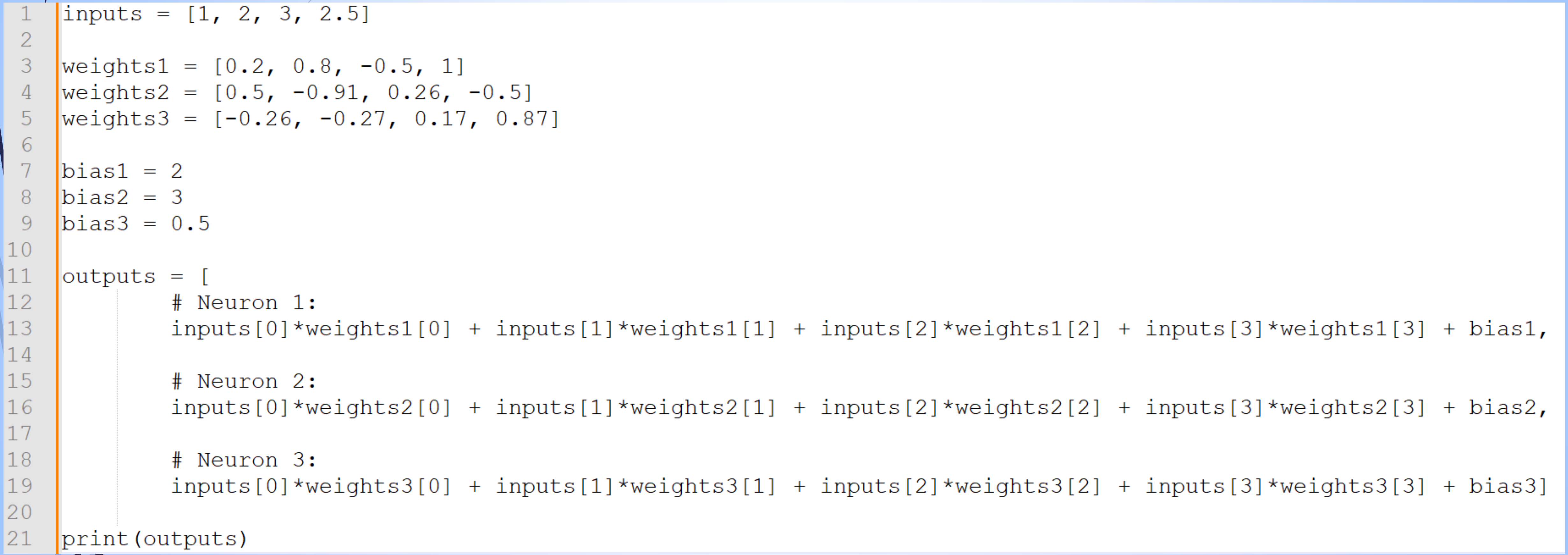
- Neural networks consists of multiple layers having more than a single neuron
- Layers are nothing more than just a group of neurons
- Each neuron in layer gets same input but has a unique set of weight and bias associated with it.
- This way each neuron produces unique output
- A basic neural network of 3 neurons in single layer

  with 4 inputs look like as shown in the figure.



- Each neuron is connected to same set of inputs making it a Fully Connected Layer

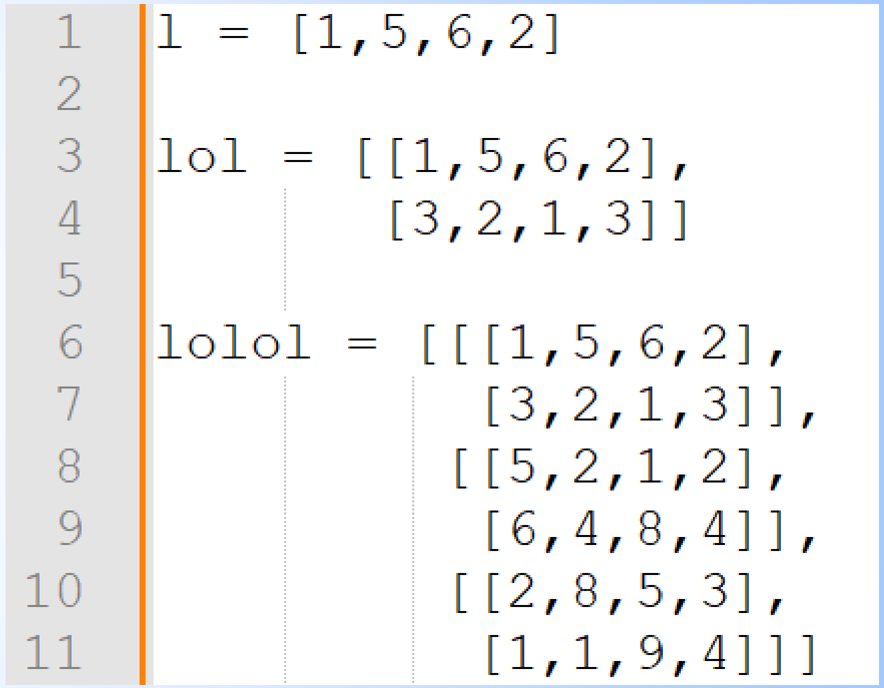
Layers of Neurons (cont.)



```python
inputs = [1, 2, 3, 2.5]

weights1 = [0.2, 0.8, -0.5, 1]
weights2 = [0.5, -0.91, 0.26, -0.5]
weights3 = [-0.26, -0.27, 0.17, 0.87]

bias1 = 2
bias2 = 3
bias3 = 0.5

outputs = [
        # Neuron 1:
        inputs[0]*weights1[0] + inputs[1]*weights1[1] + inputs[2]*weights1[2] + inputs[3]*weights1[3] + bias1,

        # Neuron 2:
        inputs[0]*weights2[0] + inputs[1]*weights2[1] + inputs[2]*weights2[2] + inputs[3]*weights2[3] + bias2,

        # Neuron 3:
        inputs[0]*weights3[0] + inputs[1]*weights3[1] + inputs[2]*weights3[2] + inputs[3]*weights3[3] + bias3]

print(outputs)
```

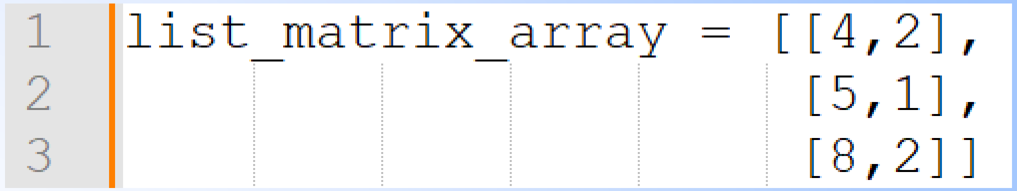
Lists, Vectors and Tensors

- Python lists are data containers containing comma separated objects, numbers etc.
- Matrix (matrices) can also be represented as a list.
- Matrix contains columns and rows – it can be called as 2D List / Array.
- Each list has a shape representing its dimensions (also called axises)
- A typical matrix has shape of (2, 2)



```
l = [1,5,6,2]

lol = [[1,5,6,2],
       [3,2,1,3]]

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

Lists (cont.)

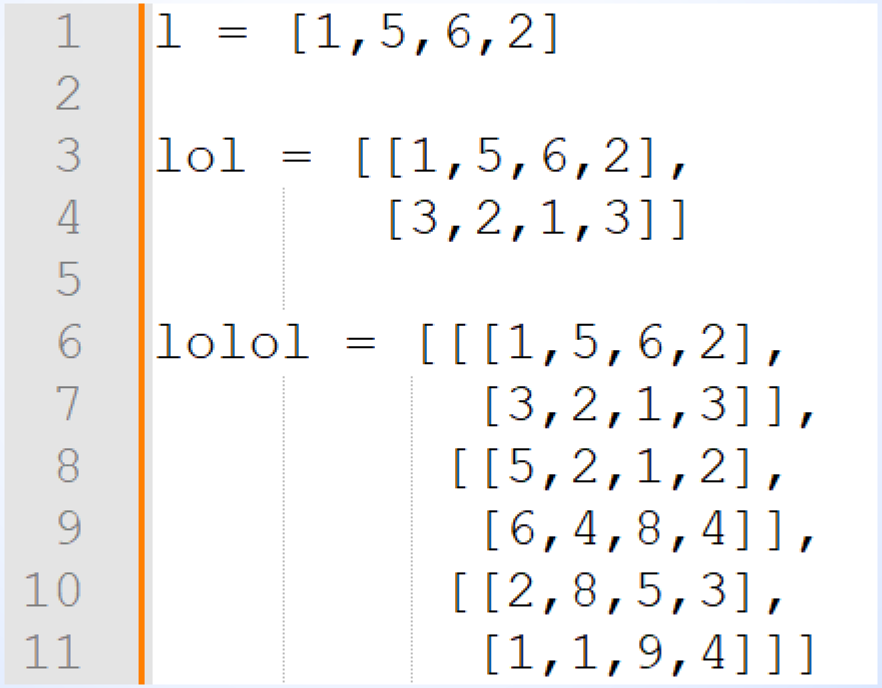


```
list_matrix_array = [[4,2],
                     [5,1],
                     [8,2]]
```

- Shape of the given matrix is (3, 2).
- As the after the starting "[" bracket, there are 3 lists inside.
- Inside each list, there are 2 elements.
- It can also be denoted as (1, 3, 2) – One List, containing 3 sublists, each having 2 elements inside.
- Neural networks expect specific input tensor shapes.\
- For example – (28, 28, 3) for MNIST data where 28 x 28 is height and width while 3 is for color channels.

Lists shapes quiz?

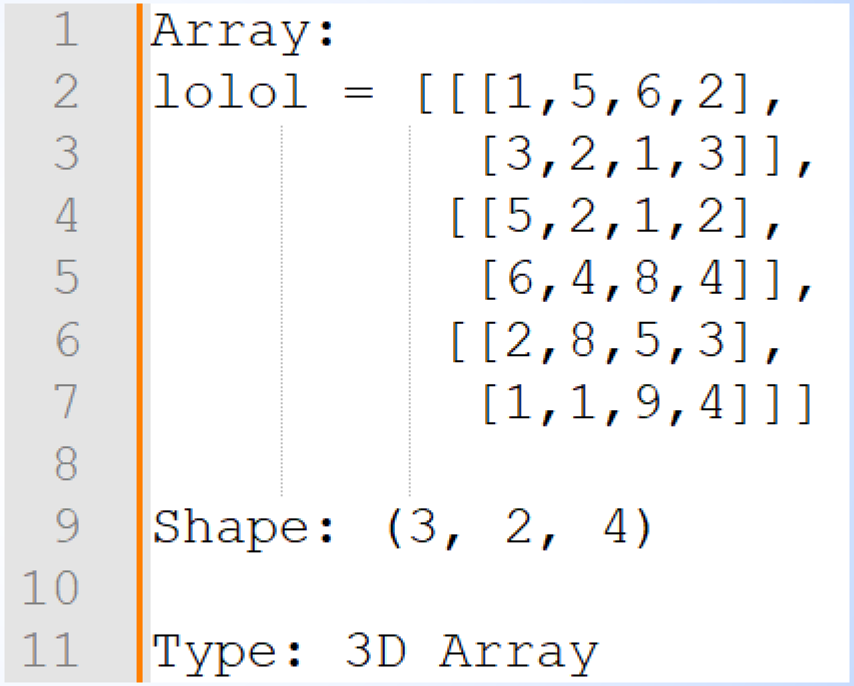
- Can you guess the shapes of these list?



```
l = [1,5,6,2]

lol = [[1,5,6,2],
       [3,2,1,3]]

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

List shapes

- Let's look at the solution of "list_of_list_of_lists" shape
- First Level contains 3 matrices:
  - [[1,5,6,2],
  - [3,2,1,3]]
  - —---------
  - [[5,2,1,2],
  - [6,4,8,4]]
  - —---------
  - [[2,8,5,3],
  - [1,1,9,4]]



```
Array:
lolol = [[[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
```

- That's what's inside the most outer brackets and the size of this dimension is then 3
- Each second level contains 2 matrices that's why the size of this dimension is 2.
- Lastly, each matrix contains 4 elements so the overall shape is (3, 4, 2)

Vectors, Dot Product and Tensors

- Vectors are simply called as 1D-Array/List
- Vector = [1, 2, 3, 4]
- Tensors are objects that can be represented as n-dimensional arrays
- A tensor object is an object that can be represented as an array
- More details will be given in upcoming chapters
- Whenever multiplication of vectors is performed, it is done in two ways:
- Dot Product – Results in a scalar (single value / number)
- Cross Product – Results in a vector
- Recall the first neuron example that we looked earlier in these slides
- Inputs = [1, 2, 3]; Weights = [0.2, 0.8, -0.5]; b = 2
- The calculated output was done by multiplying each input element to same corresponding value to weights array and adding bias afterwards.
- This is basically an example of dot product.

Dot Product

- A dot product of two vectors is a sum of products of consecutive vector elements.
- Both vectors must be of the same size (i.e. equal number of elements)
- Let a, b be two vectors such as [1, 2, 3] and [2, 3, 4] respectively
- Their dot product can be calculated according to the given equation

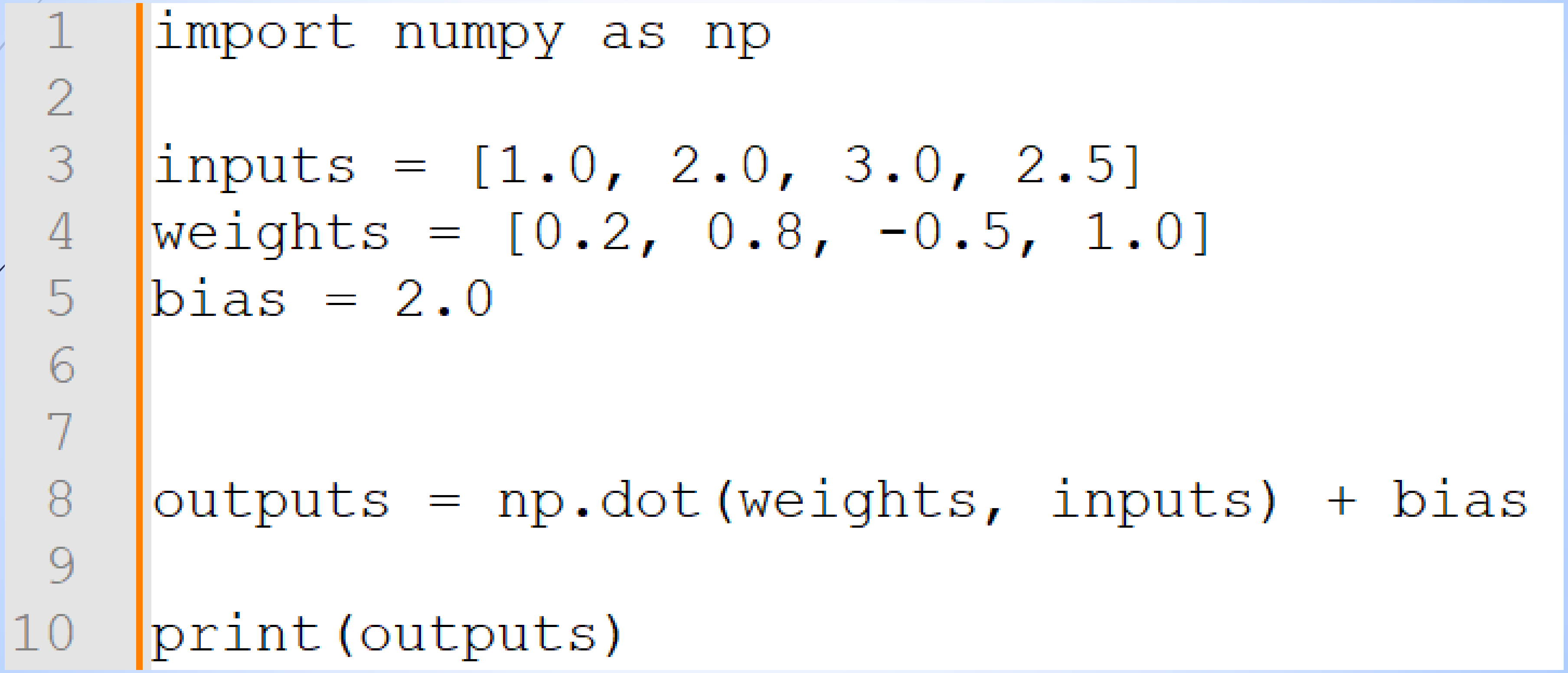
$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

$$\vec{a} \cdot \vec{b} = [1, 2, 3] \cdot [2, 3, 4] = 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 = 20$$

Numpy – A brief introduction

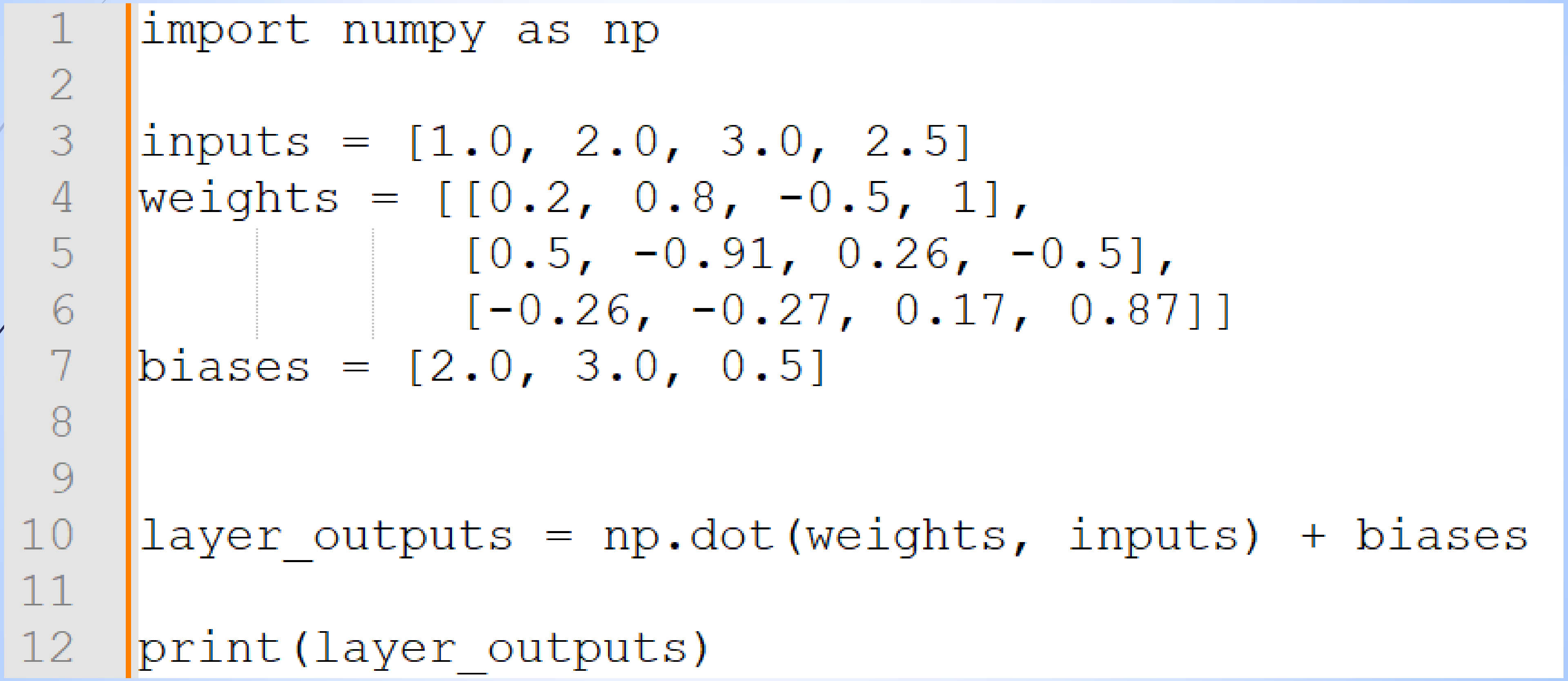
- NumPy is the fundamental package for scientific computing in Python.
- It is a Python library that provides a way to deal with multidimensional arrays, matrices
- It provides a number of functions for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, basic linear algebra, basic statistical operations, random simulation and much more …
- It can be installed using `pip` – Python Package Manager
- `pip install numpy` – This command installs latest version of NumPy.
- After installation, you can directly import it into your python codes.

Single Neuron Implementation with NumPy



```python
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)
```

Layer of Neurons with NumPy



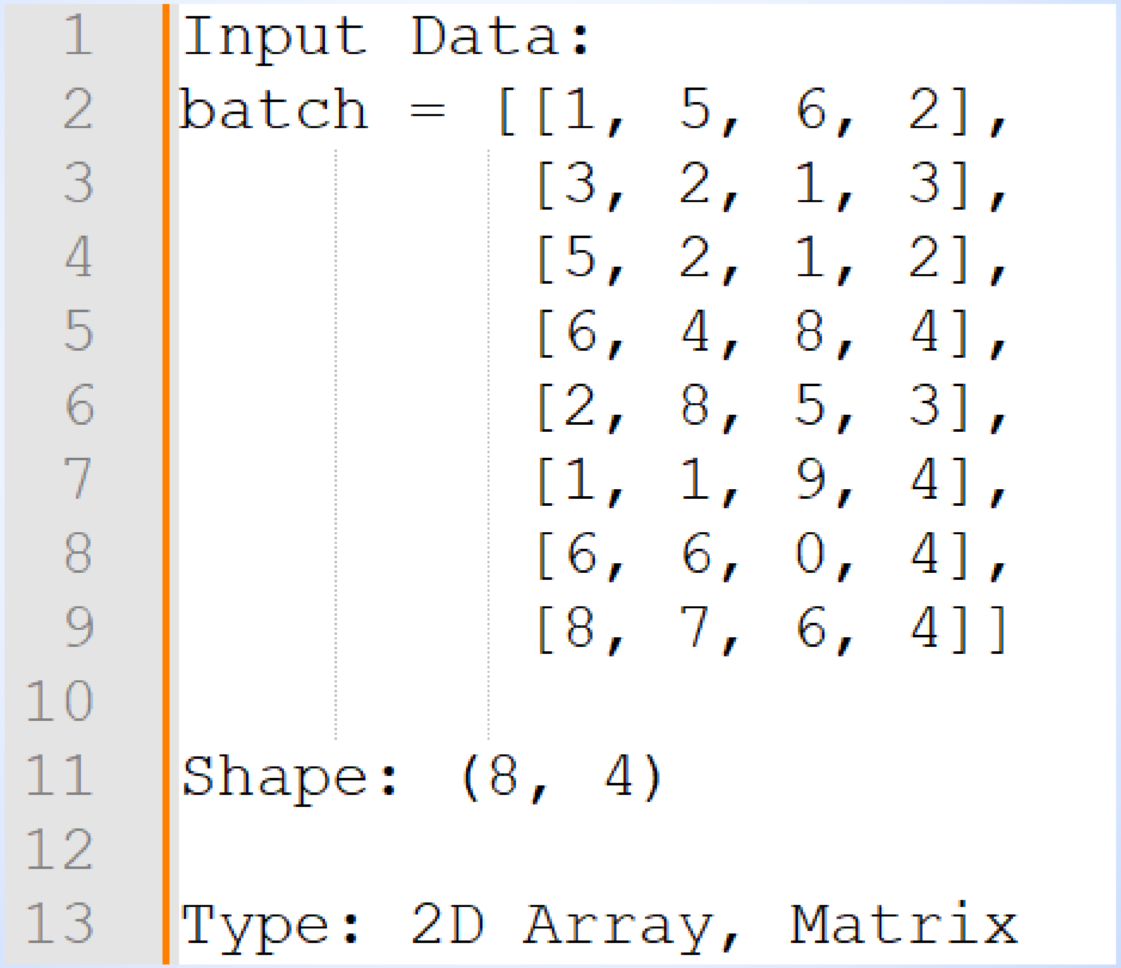
```python
import numpy as np

inputs = [1.0, 2.0, 3.0, 2.5]
weights = [[0.2, 0.8, -0.5, 1],
           [0.5, -0.91, 0.26, -0.5],
           [-0.26, -0.27, 0.17, 0.87]]
biases = [2.0, 3.0, 0.5]


layer_outputs = np.dot(weights, inputs) + biases

print(layer_outputs)
```

Batches of Data

- To train neural networks, data is typically given in batches.

- Neural networks take in many samples (batches) at a time for two reasons:
  - It is faster to train in batches – Thanks to parallel processing
  - Batches help with generalization: Training in batches gives higher chance of making more meaningful changes to weights and biases; Without batches, you may likely to be fitting to single example rather than making useful changes to weights and biases that are useful for entire dataset.

Batch of Data – Example



```
Input Data:
batch = [[1, 5, 6, 2],
         [3, 2, 1, 3],
         [5, 2, 1, 2],
         [6, 4, 8, 4],
         [2, 8, 5, 3],
         [1, 1, 9, 4],
         [6, 6, 0, 4],
         [8, 7, 6, 4]]

Shape: (8, 4)

Type: 2D Array, Matrix
```

Matrix Product

- Matrix product is defined as an operation of performing dot products of all combinations of rows from 1st matrix and the columns of the 2nd matrix between two or more matrices.
- To perform a matrix product, the size of the second dimension of the 1st matrix must match the size of the first dimension of the 2nd matrix (Dimension means shape – Recall!)
- For example: first matrix has shape of (5, 4) and second matrix has shape of (4, 7)
  - Matrix product is possible because the “4” – 2nd dimension from 1st Matrix matches with “4” – 1st Dimension of 2nd Matrix
  - The resulting matrix will be of shape (5, 7)

Matrix Product (cont.)

$$a = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

"a" is row vector (1, 3)

$$b = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

"b" is column vector (3, 1)

$$ab = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 20 \end{bmatrix}$$

Matrix Transposition (Transpose)

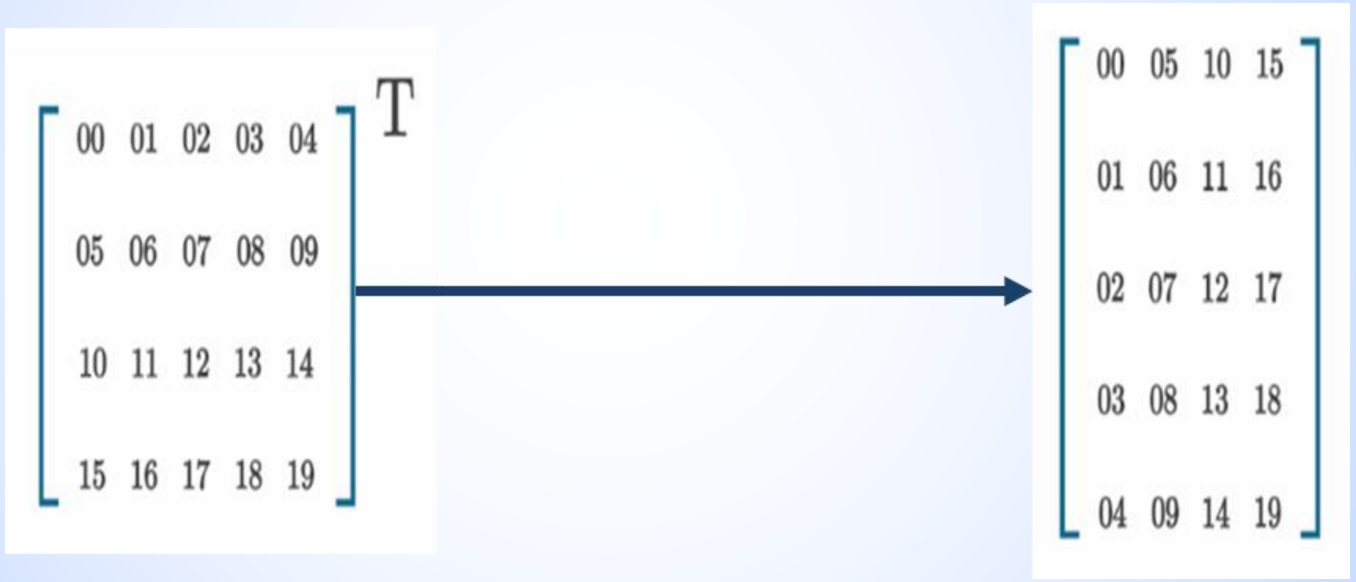
- What if a = [1, 2, 3]; b = [2, 3, 4] ? Is this multiplication possible?
    - No! Because of (1, 3) and (1, 3) shape.. Recall matrix product multiplication rule
- In some cases, the data you are given is not in appropriate shape (compatible) to be used.
- In such cases, transpose can help you fix this shape compatibility issue!
- Transposition simply modifies a matrix in a way that its rows become columns and columns become rows
- So, if you apply transpose to b, its shape will become (3, 1) – Shown in previous slide
- Hence, it makes it compatible to multiply "a" and "b" matrices.

Matrix Transpose Examples

- Transpose is represented by putting a “T” or “t” symbol on matrices.

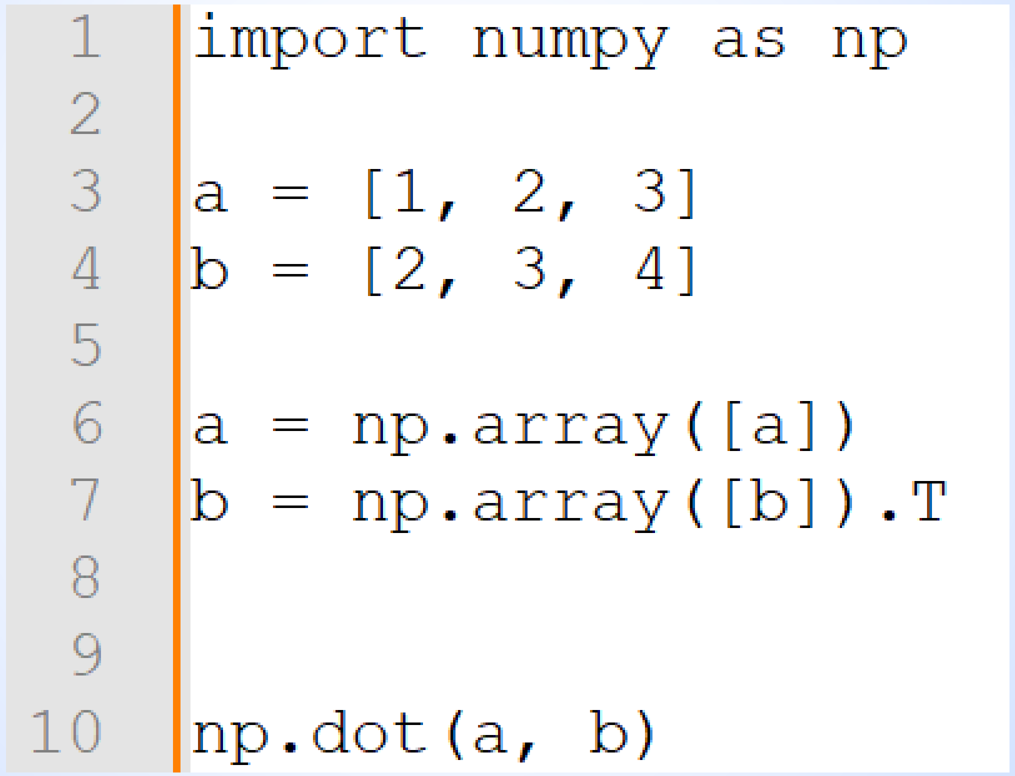
$$ab^T = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 20 \end{bmatrix}$$

Matrix Transpose Examples



$$
\begin{bmatrix} 00 & 01 & 02 & 03 & 04 \\ 05 & 06 & 07 & 08 & 09 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \end{bmatrix}^T \longrightarrow \begin{bmatrix} 00 & 05 & 10 & 15 \\ 01 & 06 & 11 & 16 \\ 02 & 07 & 12 & 17 \\ 03 & 08 & 13 & 18 \\ 04 & 09 & 14 & 19 \end{bmatrix}
$$

Matrix Transpose Code Example



```python
import numpy as np

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

a = np.array([a])
b = np.array([b]).T


np.dot(a, b)
```

Thank You