Machine Learning - Introduction to Artificial Neural Networks (Unit 7)

Overview

This unit introduced Artificial Neural Networks (ANNs), inspired by biological neurons and central to Industry 4.0 decision-making (Goodfellow et al., 2016). It focused on their structure, learning process, and key functions that drive decision-making.

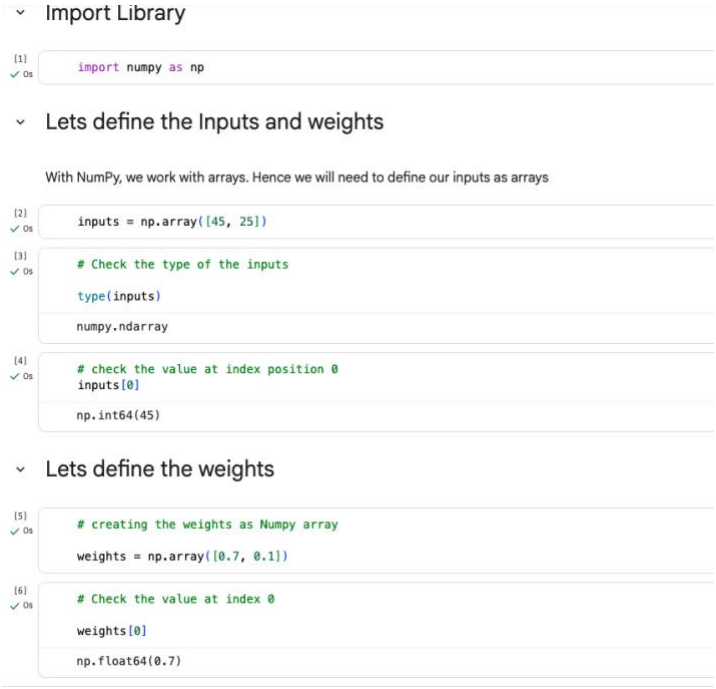
What I Have Learned

I learned how artificial neurons work together to improve predictions, using activation functions like ReLU and sigmoid. The perceptron activity in Google Colab made training easier to follow and gave me a solid foundation in building and training neural networks.

Activity: Perceptron

a. Simple perceptron

This activity helped me understand how a perceptron works. Using NumPy, I calculated weighted sums and applied a step function for binary output. Testing different weights showed how small changes affect results. It gave me a clear view of how perceptrons act as the foundation of neural networks (Rashid, 2021).



Import Library

```
[1] ✓ 0s
import numpy as np
```

Lets define the Inputs and weights

With NumPy, we work with arrays. Hence we will need to define our inputs as arrays

```
[2] ✓ 0s
inputs = np.array([45, 25])
```

```
[3] ✓ 0s
# Check the type of the inputs
type(inputs)
numpy.ndarray
```

```
[4] ✓ 0s
# check the value at index position 0
inputs[0]
np.int64(45)
```

Lets define the weights

```
[5] ✓ 0s
# creating the weights as Numpy array
weights = np.array([0.7, 0.1])
```

```
[6] ✓ 0s
# Check the value at index 0
weights[0]
np.float64(0.7)
```

Create the Sum Function

The dot function is called the dot product from linear algebra. If you are dealing with a huge dataset, The processing differen the for loop used in the last notebook and this dot product will significantly be different.

[7]
```
def sum_func(inputs, weights):
    return inputs.dot(weights)
```

[8]
```
# for weights = [0.7, 0.1]

s_prob1 = sum_func(inputs, weights)
s_prob1
```
```
np.float64(34.0)
```

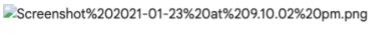
Create Step function

[9]
```
def step_function(sum_func):
  if (sum_func >= 1):
    print(f'The Sum Function is greater than or equal to 1')
    return 1
  else:
        print(f'The Sum Function is NOT greater')
        return 0
```

Result

[10]
```
step_function(s_prob1 )
```
```
The Sum Function is greater than or equal to 1
1
```

If the is weights = [- 0.7, 0.1]



[11]
```
weights = [-0.7, 0.1]
```

[12]
```
# for weights = [- 0.7, 0.1]

s_prob2 = sum_func(inputs, weights)

round(s_prob2, 2)  #round to 2 decimal places
```
```
np.float64(-29.0)
```

Result

[13]
```
step_function(s_prob2)
```
```
The Sum Function is NOT greater
0
```

By changing the input values and weights observe different results

[14]
```
inputs = np.array([33, 8])
```

[15]
```
inputs[0]
```
```
np.int64(33)
```

[16]
```
weights = np.array([0.8, 0.2])
```

[17]
```
weights[1]
```
```
np.float64(0.2)
```

[18]
```
def sum_func(inputs, weights):
    return inputs.dot(weights)
```

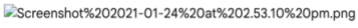
[19]
```
s_prob1 = sum_func(inputs, weights)
s_prob1
```
```
np.float64(28.000000000000004)
```

[20]
```
def step_function(sum_func):
  if (sum_func >= 1):
    print(f'The Sum Function is greater than or equal to 1')
    return 1
  else:
        print(f'The Sum Function is NOT greater')
        return 0
```

[21]
```
step_function(s_prob1 )
```
```
The Sum Function is greater than or equal to 1
1
```

b. Perceptron & Operator

Author: Dr Mike Lakoju, CardiffMet



Import Library

```
import numpy as np
```

Define "Inputs, outputs and weights" as Numpy arrays

Inputs

```
# Creating input values as a matrix not as a vector
inputs = np.array([[0,0], [0,1], [1,0], [1,1]])
```

```
# Chcking the shape of the inputs

inputs.shape
```

```
(4, 2)
```

Outputs

```
outputs = np.array([0, 0, 0, 1])
```

```
#Checking the shape of the outputs

outputs.shape
```

```
(4,)
```

Weights

```
# one weight for x1 and one for x2
weights = np.array([0.0, 0.0])
```

Learning Rate

```
learning_rate = 0.1
```

Step function

```
# This is our Activation function

def step_function(sum):
  if (sum >= 1):
    #print(f'The Sum of Weights is Greater or equal to 1')
    return 1
  else:
        #print(f'The Sum of Weights is NOT > or = to 1')
        return 0
```

Process Output

We define a function that allows us to calculate/ proc
calculate the sum function using Numpy. Finally, we c

```
def cal_output(instance):
  sum_func = instance.dot(weights)
  return step_function(sum_func)
```

We pass it as alist in a numpy array ...

```
cal_output(np.array([[1,1]]))
```

```
0
```

```
def train():
    #
    total_error_value = 1
    # While the total_error_value is not equal to zero. we are asumming that at the s
    while (total_error_value != 0):
        #making the total_error 0 so we can do other calculations
        total_error_value = 0
        #Looping into each row of the dataset (remember indexing in python starts at
        for i in range(len(outputs)):
            #Calculating predictions
            prediction = cal_output(inputs[i])
            # Calculating the absolute value of the error
            error = abs(outputs[i] - prediction)
            #Updating the error
            total_error_value  += error

            if error > 0:
                for j in range(len(weights)):
                    #updating the weights for x1 and x2
                    weights[j] = weights[j] + (learning_rate * inputs[i][j] * error)
                    print('Weight updated to: ' + str(weights[j]))
        print('Total error Value: ' + str(total_error_value))
```

```
train()
```

```
Weight updated to: 0.1
Weight updated to: 0.1
Total error Value: 1
Weight updated to: 0.2
Weight updated to: 0.2
Total error Value: 1
Weight updated to: 0.30000000000000004
Weight updated to: 0.30000000000000004
Total error Value: 1
Weight updated to: 0.4
Weight updated to: 0.4
Total error Value: 1
Weight updated to: 0.5
Weight updated to: 0.5
Total error Value: 1
Total error Value: 0
```

References

Rashid, T. (2021) *Make Your Own Neural Network*. 2nd edn. CreateSpace Independent Publishing Platform.