Multilayer Perceptron

October 2, 2022

0.1 Sigmoid Function

[1]:
```python
#import libraries
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```python
# sigmoid activation function
def sigmoid(x):
    return 1/(1 + np.exp(-x))
```

0.2 Implementation of Multilayer Perceptron

[3]:
```python
#define weights and bias as w and b, x is input array, lastly 'a' being the node
x = np.array([0.5, 1])
w1 = np.array([[0.1, 0.2, 0.15], [0.6, 0.2, 0.8]])
b1 = np.array([0.1, 0.2, 0.05])
```

[4]:
```python
#calculate the dot product and sum with bias
a1 = np.dot(x, w1) + b1
a1
```

[4]:
```
array([0.75 , 0.5  , 0.925])
```

[5]:
```python
#calculate the activation function for a1
z1 = sigmoid(a1)
z1
```

[5]:
```
array([0.6791787 , 0.62245933, 0.71605979])
```

[6]:
```python
#define the weights for second node
w2 = np.array([[0.1, 0.2], [0.05, 0.5], [0.1, 0.2]])
w2
```

[6]:
```
array([[0.1 , 0.2 ],
       [0.05, 0.5 ],
       [0.1 , 0.2 ]])
```

[7]:
```python
#define bias for second node

b2 = np.array([0.4, 0.5])
b2
```

[7]: array([0.4, 0.5])

[8]:
```python
#calculate the dot product and sum with bias
a2 = np.dot(z1, w2) + b2
a2
```

[8]: array([0.57064682, 1.09027736])

[9]:
```python
#calculate the activation function for a2

z2 = sigmoid(a2)
z2
```

[9]: array([0.63891241, 0.74843395])

[10]:
```python
#define weight and bias for second node

w3 = np.array([[0.1, 0.2], [0.9, 0.8]])
b3 = np.array([0.9, 0.9])
```

[11]:
```python
#calculate the activation function
a3 = np.dot(z2, w3) + b3
a3
```

[11]: array([1.63748179, 1.62652964])

[12]:
```python
#calculate the activation function for a3
z3 = sigmoid(a3)
z3
```

[12]: array([0.83719199, 0.83569368])