Name : Arul Kumar ARK

Roll No. : 225229103

Lab2. Design of Logic Gates using Perceptron and Keras

Part-I: Design OR gate using the concept of Perceptron

Step1: Define helper functions

In [5]:

```
import numpy as np
```

In [6]:

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

In [7]:

```
def logic_gate(w1, w2, b):
    return lambda x1, x2: sigmoid(w1 * x1 + w2 * x2 + b)
```

In [8]:

```
def test(gate):
    for a, b in (0, 0), (0, 1), (1, 0), (1, 1):
        print("{}, {}: {}".format(a, b, np.round(gate(a,b))))
```

Step2: Identify values for weights, w1 and w2 and bias, b, for OR gate.

In [9]:

```
or_gate = logic_gate(20, 20, -10)
test(or_gate)
```

```
0, 0: 0.0
0, 1: 1.0
1, 0: 1.0
1, 1: 1.0
```

Part-II: Implement the operations of AND, NOR and NAND gates

Step1: Identify values for weights, w1 and w2 and bias, b, for AND gate.

In [10]:

```
and_gate = logic_gate(20, 20, -30)
test(and_gate)
```

```
0, 0: 0.0
0, 1: 0.0
1, 0: 0.0
1, 1: 1.0
```

Step2: Identify values for weights, w1 and w2 and bias, b, for NOR gate.

In [11]:

```
nor_gate = logic_gate(-20,-20,10)
test(nor_gate)
```

```
0, 0: 1.0
0, 1: 0.0
1, 0: 0.0
1, 1: 0.0
```

Step3: Identify values for weights, w1 and w2 and bias, b, for NAND gate.

In [12]:

```
nand_gate = logic_gate(-20,-20,30)
test(nand_gate)
```

```
0, 0: 1.0
0, 1: 1.0
1, 0: 1.0
1, 1: 0.0
```

Part-III: Limitations of single neuron for XOR operation

In [13]:

```
def or_gate(x1, x2):
    return logic_gate(20, 20, -10)(x1, x2)
def nand_gate(x1, x2):
    return logic_gate(-20, -20, 30)(x1, x2)
def and_gate(x1, x2):
    return logic_gate(20, 20, -30)(x1, x2)
def xor_gate(x1, x2):
    return and_gate(or_gate(x1, x2), nand_gate(x1, x2))
for a, b in [(0, 0), (0, 1), (1, 0), (1, 1)]:
    print("{}, {}: {}".format(a, b, np.round(xor_gate(a, b))))
```

```
0, 0: 0.0
0, 1: 1.0
1, 0: 1.0
1, 1: 0.0
```

Part-IV: Logic Gates using Keras library

In [14]:

```
from tensorflow.keras.models import Sequential
```

In [15]:

```
from tensorflow.keras.layers import Dense
```

Model - AND , OR , NAND , NOR and XOR

In [16]:

```
model = Sequential()
model.add(Dense(16, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['binary_accuracy'])
```

AND gate

In [17]:

```
and_input = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
and_output = np.array([[0],[0],[0],[1]], "float32")
```

In [18]:

```
model.fit(and_input, and_output, epochs=100, verbose=2)
print(model.predict(and_input).round())
```

```
Epoch 1/100
1/1 - 1s - loss: 0.2844 - binary_accuracy: 0.5000 - 1s/epoch - 1s/step
Epoch 2/100
1/1 - 0s - loss: 0.2831 - binary_accuracy: 0.5000 - 8ms/epoch - 8ms/ste
p
Epoch 3/100
1/1 - 0s - loss: 0.2819 - binary_accuracy: 0.5000 - 10ms/epoch - 10ms/s
tep
Epoch 4/100
1/1 - 0s - loss: 0.2807 - binary_accuracy: 0.5000 - 9ms/epoch - 9ms/ste
p
Epoch 5/100
1/1 - 0s - loss: 0.2795 - binary_accuracy: 0.5000 - 12ms/epoch - 12ms/s
tep
Epoch 6/100
1/1 - 0s - loss: 0.2783 - binary_accuracy: 0.5000 - 9ms/epoch - 9ms/ste
p
Epoch 7/100
1/1 - 0s - loss: 0.2771 - binary_accuracy: 0.5000 - 11ms/epoch - 11ms/s
```

OR gate

In [19]:

```
or_input = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
or_output = np.array([[0],[1],[1],[1]], "float32")
```

In [20]:

```
model.fit(or_input, or_output, epochs=100, verbose=2)
print(model.predict(or_output).round())
```

```
Epoch 1/100
1/1 - 0s - loss: 0.2346 - binary_accuracy: 0.7500 - 17ms/epoch - 17ms/s
tep
Epoch 2/100
1/1 - 0s - loss: 0.2348 - binary_accuracy: 0.7500 - 12ms/epoch - 12ms/s
tep
Epoch 3/100
1/1 - 0s - loss: 0.2347 - binary_accuracy: 0.7500 - 14ms/epoch - 14ms/s
tep
Epoch 4/100
1/1 - 0s - loss: 0.2343 - binary_accuracy: 0.7500 - 15ms/epoch - 15ms/s
tep
Epoch 5/100
1/1 - 0s - loss: 0.2337 - binary_accuracy: 0.7500 - 9ms/epoch - 9ms/ste
p
Epoch 6/100
1/1 - 0s - loss: 0.2328 - binary_accuracy: 0.7500 - 11ms/epoch - 11ms/s
tep
Epoch 7/100
```

NAND gate

In [ ]:

```
nand_input = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
nand_output = np.array([[1],[1],[1],[0]], "float32")
```

In [ ]:

```
model.fit(nand_input, nand_output, epochs=100, verbose=2)
print(model.predict(nand_input).round())
```

NOR gate

In [21]:

```
nor_input = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
nor_output = np.array([[1],[0],[0],[0]], "float32")
```

In [22]:

```
model.fit(nor_input,nor_output , epochs=100, verbose=2)
print(model.predict(nor_inputnor).round())
```

```
Epoch 1/100
1/1 - 0s - loss: 0.4615 - binary_accuracy: 0.0000e+00 - 12ms/epoch - 12
ms/step
Epoch 2/100
1/1 - 0s - loss: 0.4624 - binary_accuracy: 0.0000e+00 - 12ms/epoch - 12
ms/step
Epoch 3/100
1/1 - 0s - loss: 0.4628 - binary_accuracy: 0.0000e+00 - 11ms/epoch - 11
ms/step
Epoch 4/100
1/1 - 0s - loss: 0.4628 - binary_accuracy: 0.0000e+00 - 9ms/epoch - 9m
s/step
Epoch 5/100
1/1 - 0s - loss: 0.4625 - binary_accuracy: 0.0000e+00 - 9ms/epoch - 9m
s/step
Epoch 6/100
1/1 - 0s - loss: 0.4618 - binary_accuracy: 0.0000e+00 - 7ms/epoch - 7m
s/step
Epoch 7/100
```

XOR gate

In [28]:

```
xor_input = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
xor_output = np.array([[0],[1],[1],[0]], "float32")
```

In [29]:

```
model.fit(xor_input, xor_output, epochs=100, verbose=2)
print(model.predict(xor_inputxor).round())
```

...

NOT gate

In [23]:

```
model = Sequential()
model.add(Dense(16, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error',
optimizer='adam',
metrics=['binary_accuracy'])
```

In [24]:

```
not_input = np.array([[0],[1]], "float32")
not_output = np.array([[1],[0]], "float32")
```

In [25]:

```
model.fit(not_input, not_output, epochs=100, verbose=2)
print(model.predict(not_input).round())
```

```
Epoch 1/100
1/1 - 1s - loss: 0.2774 - binary_accuracy: 0.0000e+00 - 1s/epoch - 1s/s
tep
Epoch 2/100
1/1 - 0s - loss: 0.2763 - binary_accuracy: 0.0000e+00 - 22ms/epoch - 22
ms/step
Epoch 3/100
1/1 - 0s - loss: 0.2752 - binary_accuracy: 0.0000e+00 - 21ms/epoch - 21
ms/step
Epoch 4/100
1/1 - 0s - loss: 0.2741 - binary_accuracy: 0.0000e+00 - 14ms/epoch - 14
ms/step
Epoch 5/100
1/1 - 0s - loss: 0.2730 - binary_accuracy: 0.0000e+00 - 8ms/epoch - 8m
s/step
Epoch 6/100
1/1 - 0s - loss: 0.2719 - binary_accuracy: 0.0000e+00 - 7ms/epoch - 7m
s/step
Epoch 7/100
```

In [ ]:

```
```