Name:P.Asha Belcilda

Rollno:225229104

Lab2. Design of Logic Gates using Perceptron and Keras

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

In [2]:
```
import numpy as np
import math
```

In [3]:
```
def sigmoid(x):
    s=1/(1+math.exp(-x))
    return s
```

In [4]:
```
def logic_gate(w1,w2,b):
    return lambda x1,x2: sigmoid(w1*x1+w2*x2+b)
```

In [5]:
```
def test(gate):
    for a,b in (0,0),(0,1),(1,0),(1,1):
        print("{},{}: {}".format(a,b,np.round(gate(a,b))))
```

In [6]:
```
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

In [7]:
```
and_gate = logic_gate(10, 10, -10)
test(and_gate)
```

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

In [8]:
```
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
```

In [9]:
```
nand_gate = logic_gate(-1,-1,2)
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 [10]:
```
def xor_gate(a,b):
    c=or_gate(a,b)
    d=nand_gate(a,b)
    return and_gate(c,d)
test(xor_gate)
```

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

In [11]:
```
def logic_gate(w1, W2, b):
    return lambda x1, x2: sigmoid(w1 * x1 + W2 * x2 + b)
def final(gate):
    for a, b in zip(result1, result2):
        print("{}, {}: {}".format(a, b, np.round(gate(a, b))))
result1 = []
result2 = []
or_gate = logic_gate(20,20,-10)
for a, b in (0, 0), (0, 1), (1, 0), (1, 1):
    result1.append(np.round(or_gate(a,b)))
nand_gate = logic_gate(-23,-25,35)
for a, b in (0, 0), (0, 1), (1, 0), (1, 1):
    result2.append(np.round(nand_gate(a,b)))
xor_gate = logic_gate(20,20,-30)
print("XOR Gate truth table \n")
print("X, Y X+Y")
final(xor_gate)
```

```
XOR Gate truth table

X, Y X+Y
0.0, 1.0: 0.0
1.0, 1.0: 1.0
1.0, 1.0: 1.0
1.0, 0.0: 0.0
```

Part-IV: Logic Gates using Keras library

In [12]:
```
pip install tensorflow==2.13.0 tensorflow-intel==2.13.0
```

```
Requirement already satisfied: tensorflow==2.13.0 in c:\users\sweth\downlo
ads\nlp\lib\site-packages (2.13.0)
Requirement already satisfied: tensorflow-intel==2.13.0 in c:\users\sweth
\downloads\nlp\lib\site-packages (2.13.0)
Requirement already satisfied: keras<2.14,>=2.13.1 in c:\users\sweth\downl
oads\nlp\lib\site-packages (from tensorflow-intel==2.13.0) (2.13.1)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\sweth\download
s\nlp\lib\site-packages (from tensorflow-intel==2.13.0) (2.3.0)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\sweth\downloads\n
lp\lib\site-packages (from tensorflow-intel==2.13.0) (1.14.1)
Requirement already satisfied: packaging in c:\users\sweth\downloads\nlp\l
ib\site-packages (from tensorflow-intel==2.13.0) (21.3)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\sweth\downl
oads\nlp\lib\site-packages (from tensorflow-intel==2.13.0) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\sweth\download
s\nlp\lib\site-packages (from tensorflow-intel==2.13.0) (16.0.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
c:\users\sweth\downloads\nlp\lib\site-packages (from tensorflow-intel==2.1
3.0) (0.31.0)
```

In [13]:
```
!pip install keras
```

```
Requirement already satisfied: keras in c:\users\sweth\downloads\nlp\lib\site
-packages (2.13.1)
```

In [14]:
```
import tensorflow as tf
```

In [15]:
```
#AND gate
# the four different states of the AND gate
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
# the four expected results in the same order
target_data = np.array([[0],[0],[0],[1]], "float32")
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(16, input_dim=2, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error',
optimizer='adam',
metrics=['binary_accuracy'])
model.fit(training_data, target_data, epochs=100, verbose=2)
print(model.predict(training_data).round())
```

```
Epoch 1/100
1/1 - 1s - loss: 0.2469 - binary_accuracy: 0.5000 - 928ms/epoch - 928ms/st
ep
Epoch 2/100
1/1 - 0s - loss: 0.2462 - binary_accuracy: 0.5000 - 3ms/epoch - 3ms/step
Epoch 3/100
1/1 - 0s - loss: 0.2455 - binary_accuracy: 0.5000 - 0s/epoch - 0s/step
Epoch 4/100
1/1 - 0s - loss: 0.2447 - binary_accuracy: 0.5000 - 7ms/epoch - 7ms/step
Epoch 5/100
1/1 - 0s - loss: 0.2440 - binary_accuracy: 0.5000 - 7ms/epoch - 7ms/step
Epoch 6/100
1/1 - 0s - loss: 0.2432 - binary_accuracy: 0.5000 - 15ms/epoch - 15ms/step
Epoch 7/100
1/1 - 0s - loss: 0.2425 - binary_accuracy: 0.5000 - 0s/epoch - 0s/step
Epoch 8/100
1/1 - 0s - loss: 0.2418 - binary_accuracy: 0.5000 - 8ms/epoch - 8ms/step
Epoch 9/100
1/1 - 0s - loss: 0.2410 - binary_accuracy: 0.5000 - 7ms/epoch - 7ms/step
```

In [16]:

```python
#OR gate
# the four different states of the OR gate
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
# the four expected results in the same order
target_data = np.array([[0],[1],[1],[1]], "float32")
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(16, input_dim=2, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error',
optimizer='adam',
metrics=['binary_accuracy'])
model.fit(training_data, target_data, epochs=100, verbose=2)
print(model.predict(training_data).round())
```

```
Epoch 1/100
1/1 - 1s - loss: 0.2858 - binary_accuracy: 0.2500 - 800ms/epoch - 800ms/st
ep
Epoch 2/100
1/1 - 0s - loss: 0.2843 - binary_accuracy: 0.0000e+00 - 20ms/epoch - 20ms/
step
Epoch 3/100
1/1 - 0s - loss: 0.2827 - binary_accuracy: 0.0000e+00 - 11ms/epoch - 11ms/
step
Epoch 4/100
1/1 - 0s - loss: 0.2812 - binary_accuracy: 0.0000e+00 - 9ms/epoch - 9ms/st
ep
Epoch 5/100
1/1 - 0s - loss: 0.2797 - binary_accuracy: 0.0000e+00 - 5ms/epoch - 5ms/st
ep
Epoch 6/100
1/1 - 0s - loss: 0.2783 - binary_accuracy: 0.0000e+00 - 7ms/epoch - 7ms/st
ep
Epoch 7/100
```

In [17]:
```
#NOT gate
# the four different states of the NOT gate
training_data = np.array([[0],[1]], "float32")
# the four expected results in the same order
target_data = np.array([[1],[0]], "float32")
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(16, input_dim=1, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error',
optimizer='adam',
metrics=['binary_accuracy'])
model.fit(training_data, target_data, epochs=100, verbose=2)
print(model.predict(training_data).round())
```

```
Epoch 1/100
1/1 - 1s - loss: 0.3034 - binary_accuracy: 0.0000e+00 - 786ms/epoch - 786m
s/step
Epoch 2/100
1/1 - 0s - loss: 0.3024 - binary_accuracy: 0.0000e+00 - 14ms/epoch - 14ms/
step
Epoch 3/100
1/1 - 0s - loss: 0.3015 - binary_accuracy: 0.0000e+00 - 12ms/epoch - 12ms/
step
Epoch 4/100
1/1 - 0s - loss: 0.3005 - binary_accuracy: 0.0000e+00 - 12ms/epoch - 12ms/
step
Epoch 5/100
1/1 - 0s - loss: 0.2995 - binary_accuracy: 0.0000e+00 - 7ms/epoch - 7ms/st
ep
Epoch 6/100
1/1 - 0s - loss: 0.2986 - binary_accuracy: 0.0000e+00 - 4ms/epoch - 4ms/st
ep
Epoch 7/100
```

In [18]:

```python
#NAND gate
# the four different states of the NAND gate
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
# the four expected results in the same order
target_data = np.array([[1],[1],[1],[0]], "float32")
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(16, input_dim=2, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error',
optimizer='adam',
metrics=['binary_accuracy'])
model.fit(training_data, target_data, epochs=100, verbose=2)
print(model.predict(training_data).round())
```

```
Epoch 1/100
1/1 - 1s - loss: 0.2481 - binary_accuracy: 0.5000 - 783ms/epoch - 783ms/st
ep
Epoch 2/100
1/1 - 0s - loss: 0.2475 - binary_accuracy: 0.7500 - 12ms/epoch - 12ms/step
Epoch 3/100
1/1 - 0s - loss: 0.2469 - binary_accuracy: 0.7500 - 11ms/epoch - 11ms/step
Epoch 4/100
1/1 - 0s - loss: 0.2463 - binary_accuracy: 0.7500 - 9ms/epoch - 9ms/step
Epoch 5/100
1/1 - 0s - loss: 0.2458 - binary_accuracy: 0.7500 - 11ms/epoch - 11ms/step
Epoch 6/100
1/1 - 0s - loss: 0.2452 - binary_accuracy: 0.7500 - 1ms/epoch - 1ms/step
Epoch 7/100
1/1 - 0s - loss: 0.2446 - binary_accuracy: 0.7500 - 6ms/epoch - 6ms/step
Epoch 8/100
1/1 - 0s - loss: 0.2440 - binary_accuracy: 0.7500 - 5ms/epoch - 5ms/step
Epoch 9/100
1/1 - 0s - loss: 0.2435 - binary_accuracy: 0.7500 - 9ms/epoch - 9ms/step
```

In [19]:
```python
#NOR gate
# the four different states of the NOR gate
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
# the four expected results in the same order
target_data = np.array([[1],[0],[0],[0]], "float32")
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(16, input_dim=2, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error',
optimizer='adam',
metrics=['binary_accuracy'])
model.fit(training_data, target_data, epochs=100, verbose=2)
print(model.predict(training_data).round())
```

```
Epoch 1/100
1/1 - 1s - loss: 0.2945 - binary_accuracy: 0.0000e+00 - 777ms/epoch - 777m
s/step
Epoch 2/100
1/1 - 0s - loss: 0.2930 - binary_accuracy: 0.0000e+00 - 9ms/epoch - 9ms/st
ep
Epoch 3/100
1/1 - 0s - loss: 0.2914 - binary_accuracy: 0.0000e+00 - 8ms/epoch - 8ms/st
ep
Epoch 4/100
1/1 - 0s - loss: 0.2899 - binary_accuracy: 0.0000e+00 - 5ms/epoch - 5ms/st
ep
Epoch 5/100
1/1 - 0s - loss: 0.2883 - binary_accuracy: 0.0000e+00 - 8ms/epoch - 8ms/st
ep
Epoch 6/100
1/1 - 0s - loss: 0.2867 - binary_accuracy: 0.0000e+00 - 10ms/epoch - 10ms/
step
Epoch 7/100
```

In [20]:

```
#XOR gate
# the four different states of the XOR gate
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
# the four expected results in the same order
target_data = np.array([[0],[1],[1],[0]], "float32")
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(16, input_dim=2, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error',
optimizer='adam',
metrics=['binary_accuracy'])
model.fit(training_data, target_data, epochs=100, verbose=2)
print(model.predict(training_data).round())
```

```
Epoch 1/100
1/1 - 1s - loss: 0.2425 - binary_accuracy: 0.7500 - 785ms/epoch - 785ms/st
ep
Epoch 2/100
1/1 - 0s - loss: 0.2421 - binary_accuracy: 0.5000 - 5ms/epoch - 5ms/step
Epoch 3/100
1/1 - 0s - loss: 0.2418 - binary_accuracy: 0.7500 - 7ms/epoch - 7ms/step
Epoch 4/100
1/1 - 0s - loss: 0.2414 - binary_accuracy: 0.7500 - 6ms/epoch - 6ms/step
Epoch 5/100
1/1 - 0s - loss: 0.2410 - binary_accuracy: 0.7500 - 12ms/epoch - 12ms/step
Epoch 6/100
1/1 - 0s - loss: 0.2406 - binary_accuracy: 0.7500 - 6ms/epoch - 6ms/step
Epoch 7/100
1/1 - 0s - loss: 0.2403 - binary_accuracy: 0.7500 - 7ms/epoch - 7ms/step
Epoch 8/100
1/1 - 0s - loss: 0.2399 - binary_accuracy: 0.7500 - 12ms/epoch - 12ms/step
Epoch 9/100
1/1 - 0s - loss: 0.2395 - binary_accuracy: 0.7500 - 7ms/epoch - 7ms/step
```