module3

September 25, 2022

1 Logic Gates using Numpy

[811]:
```
import numpy as np
```

[812]:
```
x1 =1
x2 =1
w1 = 1
w2 = 1
b = -1
```

2 And Gate

[813]:
```
def do_and(x1,x2):
    w =np.array([w1,w2])
    x =np.array([x1,x2])
    y = np.sum(x*w) + b

    return 0 if y <= 0 else 1
```

2.0.1 Testing out do_and numpy function

[814]:
```
do_and(0,0)
```

[814]: 0

[815]:
```
xs = [(0,0),(0,1),(1,0),(1,1)]
```

[816]:
```
for x in xs:
    y = do_and(x[0],x[1])
    print("If you do_and with {} and {}, you will have {}".format(x[0],x[1],y))
```

```
If you do_and with 0 and 0, you will have 0
If you do_and with 0 and 1, you will have 0
If you do_and with 1 and 0, you will have 0
If you do_and with 1 and 1, you will have 1
```

2.0.2 NAND Gate

[817]:
```
x1 =1
x2 =1
w1 = -1
w2 = -1
b = 2
```

[818]:
```
def do_nand(x1,x2):
    w =np.array([w1,w2])
    x =np.array([x1,x2])
    y = np.sum(x*w) + b

    return 0 if y <= 0 else 1
```

Testing Nand gate with do `do_nand`

[819]:
```
do_nand(1,1)
```

[819]:
```
0
```

[820]:
```
for x in xs:
    y = do_nand(x[0],x[1])
    print("If you do_and with {} and {}, you will have {}".format(x[0],x[1],y))
```

```
If you do_and with 0 and 0, you will have 1
If you do_and with 0 and 1, you will have 1
If you do_and with 1 and 0, you will have 1
If you do_and with 1 and 1, you will have 0
```

2.0.3 OR gate

[821]:
```
x1 =2
x2 =1
w1 = 2
w2 = 2
b = -1
```

[822]:
```
def do_or(x1,x2):
    w =np.array([w1,w2])
    x =np.array([x1,x2])
    y = np.sum(x*w) + b

    return 0 if y <= 0 else 1
```

Testing OR gate with do `do_or`

[823]:
```
do_or(0,0)
```

[823]: 0

[824]:
```
for x in xs:
    y = do_or(x[0],x[1])
    print("If you do_and with {} and {}, you will have {}".format(x[0],x[1],y))
```

```
If you do_and with 0 and 0, you will have 0
If you do_and with 0 and 1, you will have 1
If you do_and with 1 and 0, you will have 1
If you do_and with 1 and 1, you will have 1
```

2.0.4 NOR gate

[825]:
```
x1 =2
x2 =1
w1 = -2
w2 = -2
b = 1
```

[826]:
```
def do_nor(x1,x2):
    w =np.array([w1,w2])
    x =np.array([x1,x2])
    y = np.sum(x*w) + b

    return 0 if y <= 0 else 1
```

Testing NOR gate with do `do_nor`

[827]:
```
do_nor(1,1)
```

[827]: 0

[828]:
```
for x in xs:
    y = do_nor(x[0],x[1])
    print("If you do_and with {} and {}, you will have {}".format(x[0],x[1],y))
```

```
If you do_and with 0 and 0, you will have 1
If you do_and with 0 and 1, you will have 0
If you do_and with 1 and 0, you will have 0
If you do_and with 1 and 1, you will have 0
```

2.0.5 XOR Gate

[829]:
```
x1 = 1
x2 = 1
```

[830]:
```
def do_and(x1,x2):
    x = np.array([x1,x2])
    w = np.array([0.5,0.5])
```

```
    b = -0.7
    y = np.sum(x*w) + b
    return 1 if  y > 0 else 0

def do_nand(x1,x2):
    return 1 if do_and(x1,x2) == 0 else 0

def do_or(x1,x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.2
    y = np.sum(x*w)+ b
    return 1 if y > 0 else 0

def do_nor(x1,x2):
        return 1 if do_or(x1,x2) == 0 else 0

#Combing 2 logic gates

def do_xor(x1,x2):
    y1 = do_or(x1,x2)
    y2 = do_nand(x1, x2)
    y = do_and(y1, y2)

    return y
```

2.0.6 Testing out the do_xor function

[831]:
```
do_xor(1,1)
```

[831]:
```
0
```

[832]:
```
for x in xs:
    y = do_xor(x[0],x[1])
    print("If you do_xor with {} and {}, you will have {}".format(x[0],x[1],y))
```

```
If you do_xor with 0 and 0, you will have 0
If you do_xor with 0 and 1, you will have 1
If you do_xor with 1 and 0, you will have 1
If you do_xor with 1 and 1, you will have 0
```