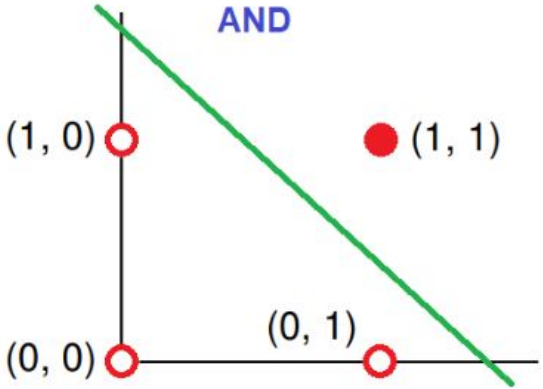
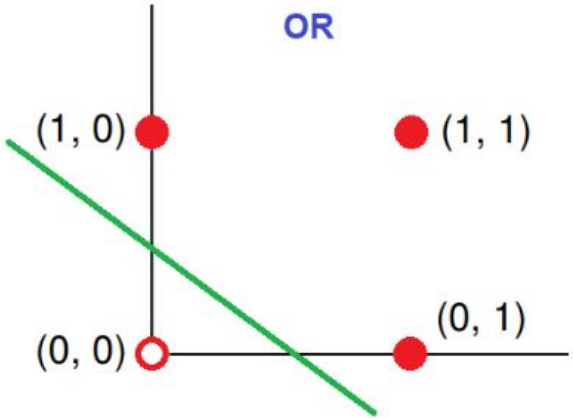
רשת נוירונים רדודה

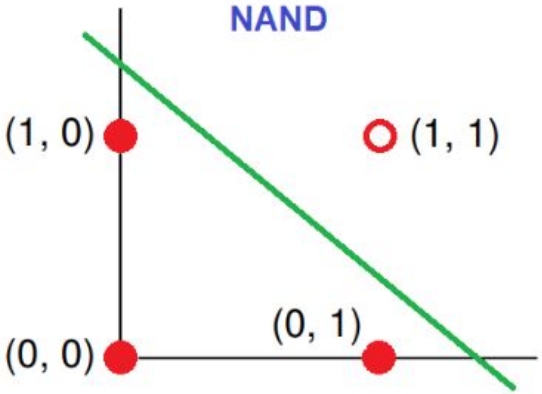
מימוש שער XOR על ידי מכונה לומדת

גדי הרמן

פרספטרון כמסווג לינארי



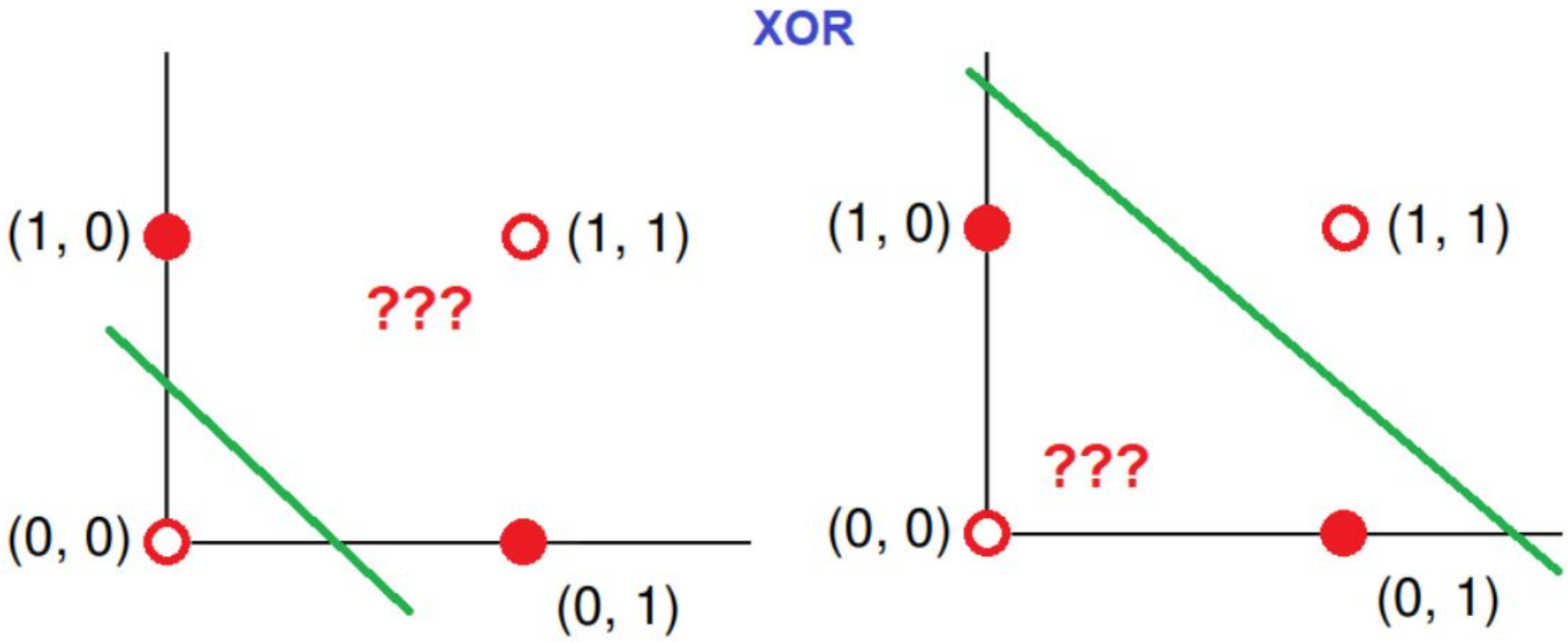




Input		Output
A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

```
[0 0] 1
[0 1] 1
[1 0] 1
[1 1] 0
```

פרספטרון כמסווג לינארי

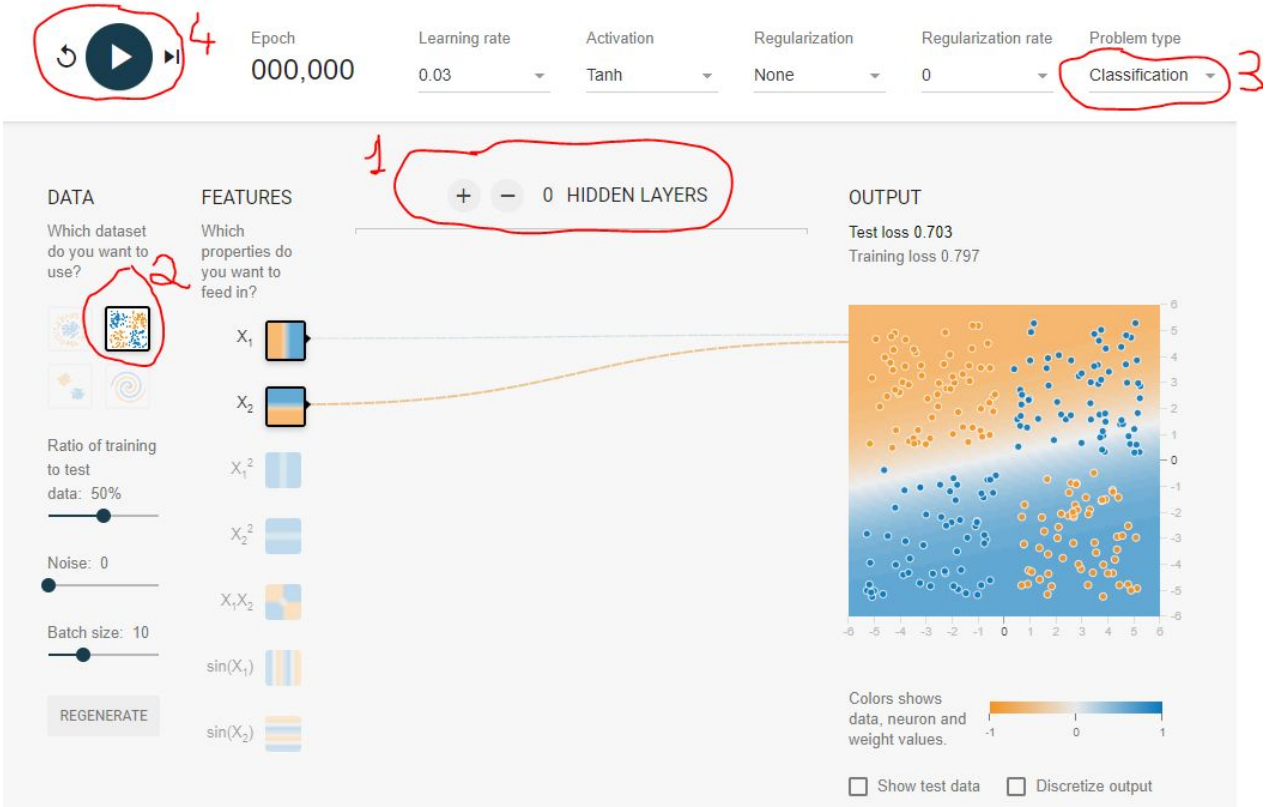


תרגול במליאה - פרספטרון כמסווג לינארי

כנסו באתר:

https://playground.tensorflow.org/

וממשו פרספטרון בודד באופן הבא:

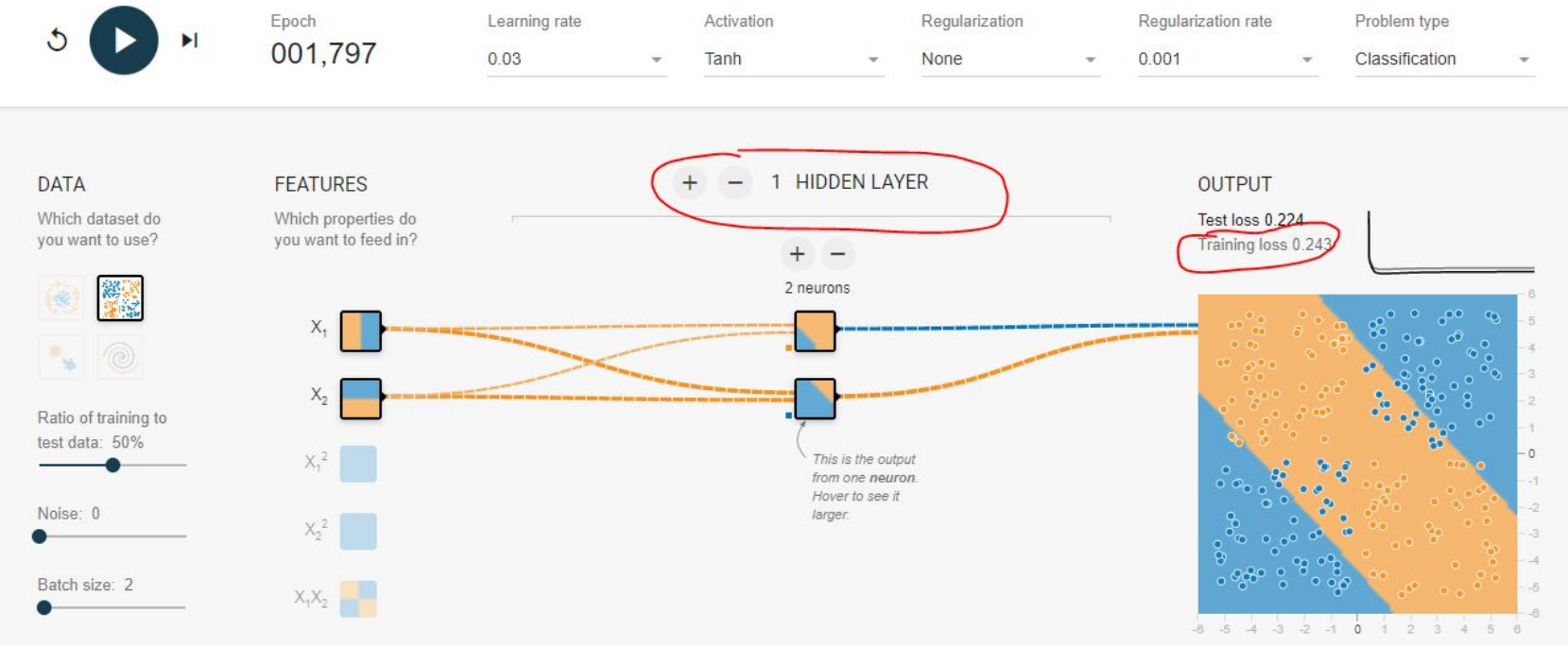


תרגול במליאה - פרספטרון כמסווג לינארי

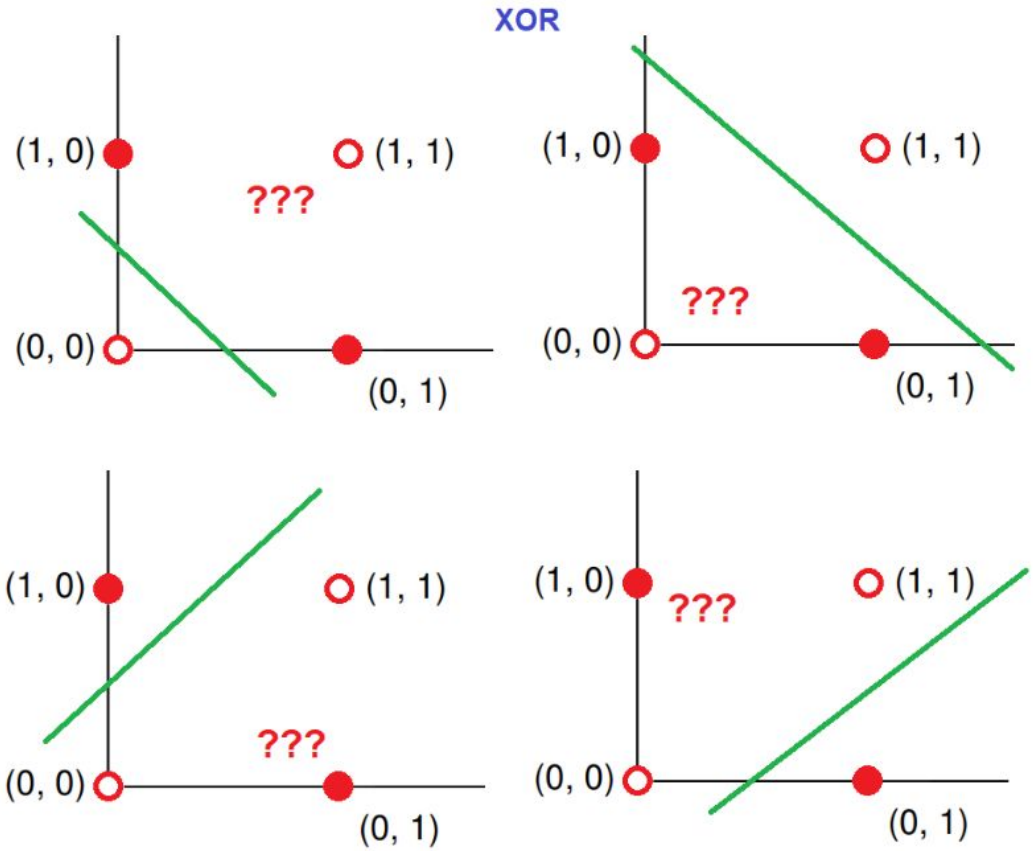
כנסו באתר:

https://playground.tensorflow.org/

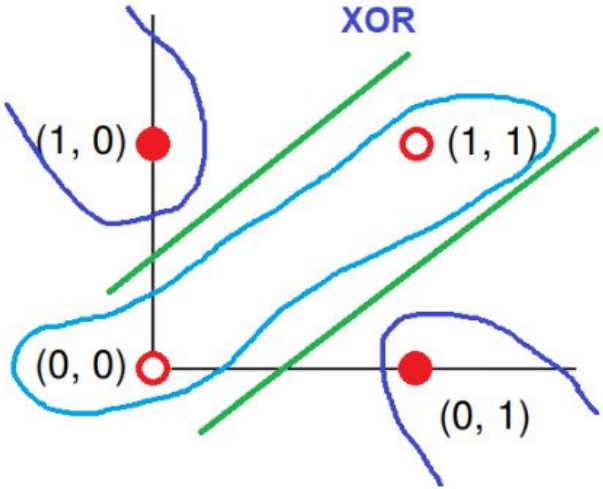
נוסיף 2 פרספטרונים נוספים כך שנקבל רשת הכולל 3 פרספטרונים האופן הבא:



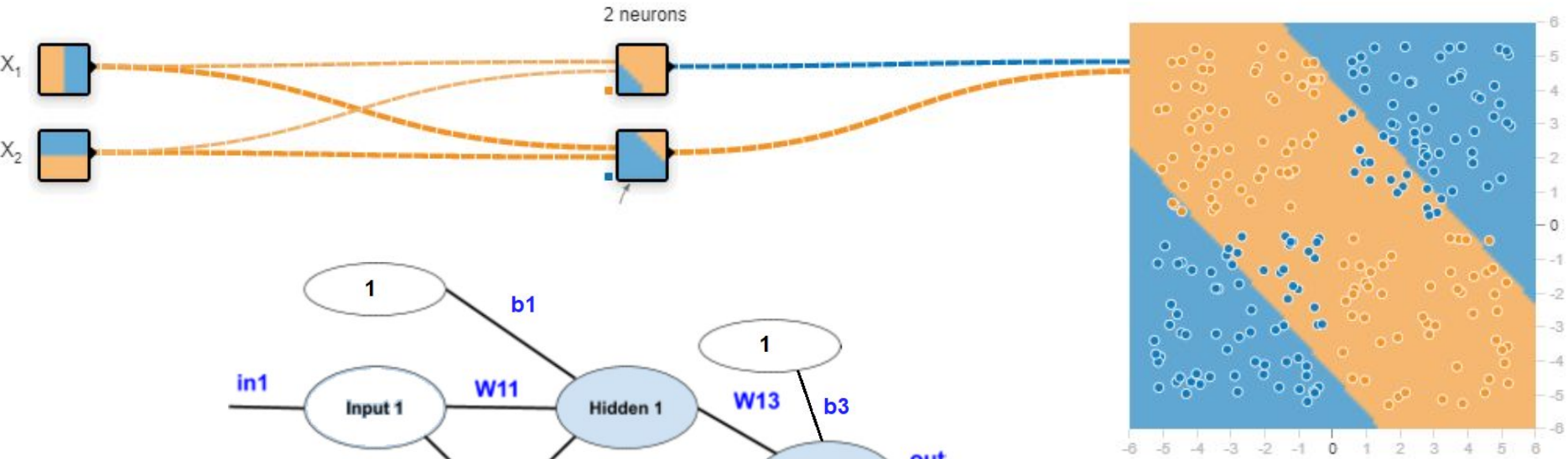
פרספטרון כמסווג לינארי

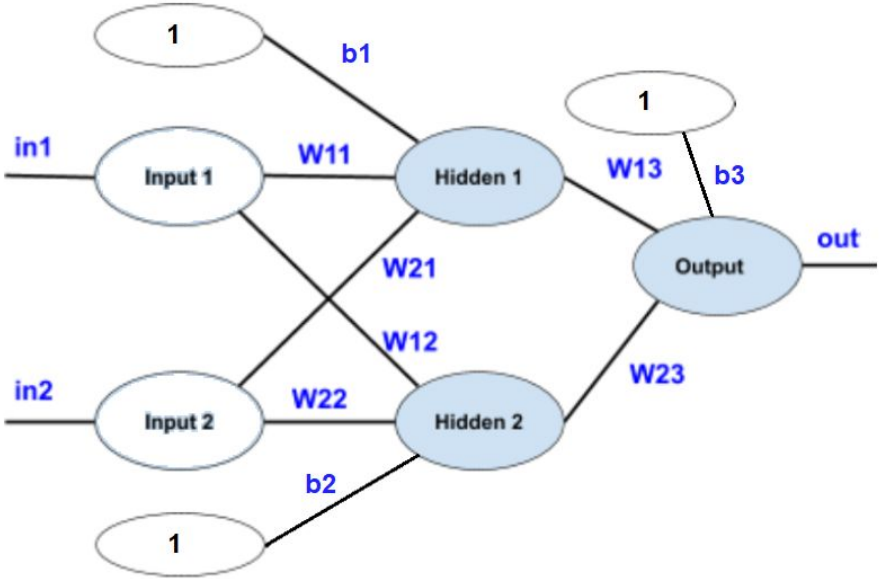


פתרון לכך יהיה מימוש רשת של מספר פרספטרונים במטרה לבנות מכונה המסוגלת ללמוד שער לוגי מסוג XOR.



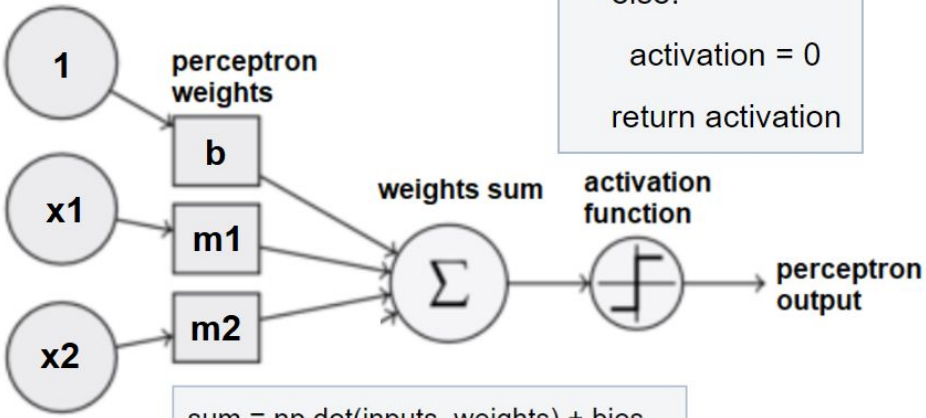
מבנה רשת נוירונים רדודה לסיווג XOR





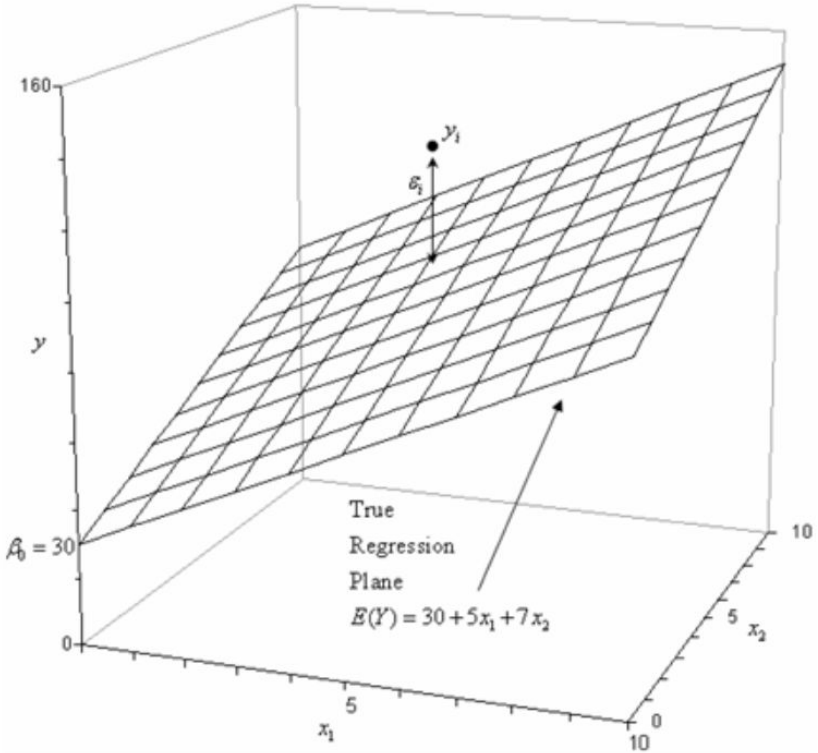
חזרה - פרספטרון כמסווג לינארי הכולל 2 מבואות

```
def Activation(s):
    if s > 0:
      activation = 1
    else:
      activation = 0
    return activation
```

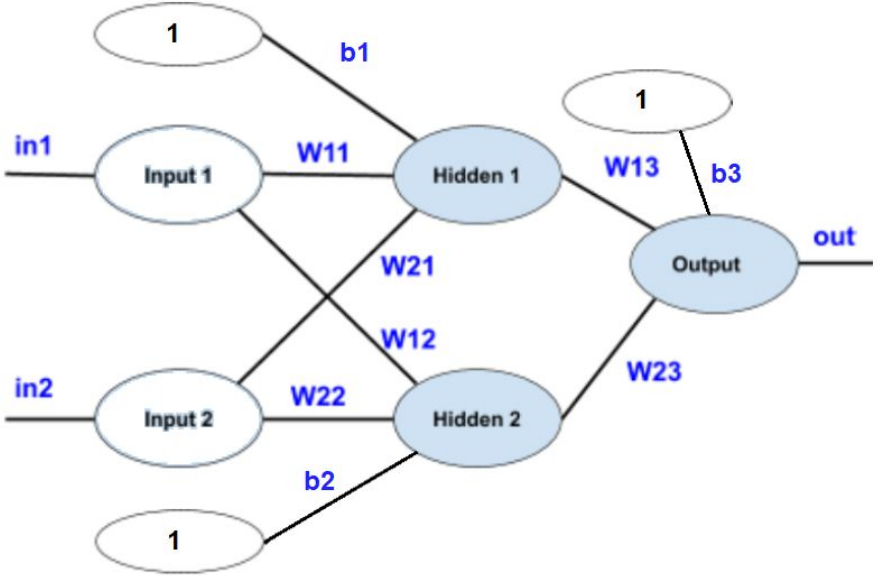


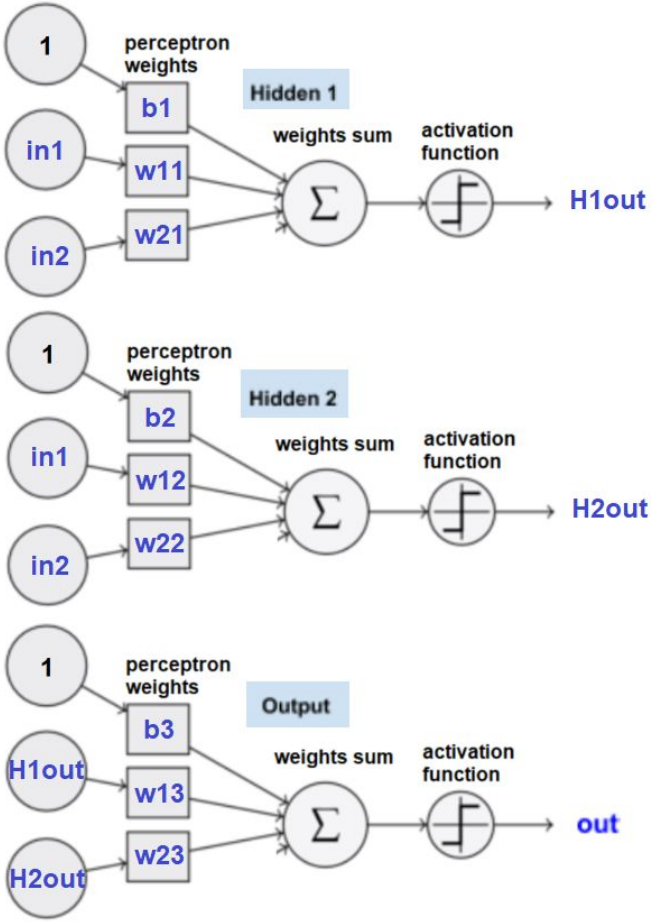

```
sum = np.dot(inputs, weights) + bios
```

```
y=m1x1+m2x2+b
```



מבנה רשת נוירונים רדודה לסיווג XOR





אלגוריתם עבור רשת נוירונים רדודה לסיווג XOR

האלגוריתם שנכתב עובד לפי השלבים הבאים:

1. נאתחל את כל המשקולות, כלומר ערכי m ו- b של אחד מהנוירונים, בערכים אקראיים בתחום שבין אפס לאחד.
2. נחשב את הפלט הכללי של מוצא רשת הנוירונים.
3. נחשב את השגיאה הכללית. כלומר ההפרש בין הערך הרצוי במוצא (אפס או אחד) לבין הערך שבמוצא הרשת ברגע זה.
4. נחשב את הנגזרת של פונקציית השגיאה עבור כל הרשת.
5. על סמך הנגזרת נשנה את המשקולות של נוירון המוצא (Output) בהתאם לשגיאה הכללית.
6. נשנה את המשקולות של 2 הנוירונים בשכבה הפנימית Hidden2 , Hidden1.
7. נחזור לבצע את כל התהליך מסעיף 2 עד שהשגיאה תהיה מינימלית

מימוש האלגוריתם

נאתחל את כל המשקולות, כלומר ערכי m ו- b של אחד מהנוירונים, בערכים אקראיים בתחום שבין אפס לאחד.

רק שהפעם נשנה את השמות ל- weights ו- bias בהתאמה ל- m ו- b

```
def __init__(self,inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons):
    self.hidden_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeurons))
    self.hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
    self.output_weights = np.random.uniform(size=(hiddenLayerNeurons,outputLayerNeurons))
    self.output_bias = np.random.uniform(size=(1,outputLayerNeurons))
    self.hidden_layer_output = np.random.uniform(size=(1,outputLayerNeurons))
```

מימוש האלגוריתם

```
def __init__(self,inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons):
    self.hidden_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeurons))
    self.hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
    self.output_weights = np.random.uniform(size=(hiddenLayerNeurons,outputLayerNeurons))
    self.output_bias = np.random.uniform(size=(1,outputLayerNeurons))
    self.hidden_layer_output = np.random.uniform(size=(1,outputLayerNeurons))
```

```
Initial hidden weights:
 [[0.70258375 0.70650312]
 [0.41006926 0.65009292]]

Initial hidden biases:
 [[0.41183037 0.47056636]]

Initial output weights:
 [[0.45877558]
 [0.0574177 ]]

Initial output biases:
 [[0.04764644]]
```

נאתחל את כל המשקולות, כלומר ערכי m ו- b של אחד מהנוירונים, בערכים אקראיים בתחום שבין אפס לאחד.

רק שהפעם נשנה את השמות ל- weights ו- bias בהתאמה ל- m ו- b

מימוש האלגוריתם

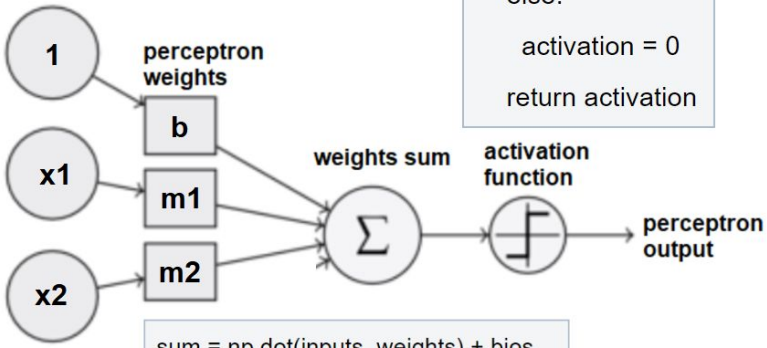
נחשב את הפלט הכללי של מוצא רשת הנוירונים.

```
def predict(self, inpt):
    #Forward Propagation
    hidden_layer_activation = np.dot(inpt,self.hidden_weights)
    hidden_layer_activation += self.hidden_bias
    self.hidden_layer_output = self.sigmoid(hidden_layer_activation)

    output_layer_activation = np.dot(self.hidden_layer_output,self.output_weights)
    output_layer_activation += self.output_bias
    return self.sigmoid(output_layer_activation)
```

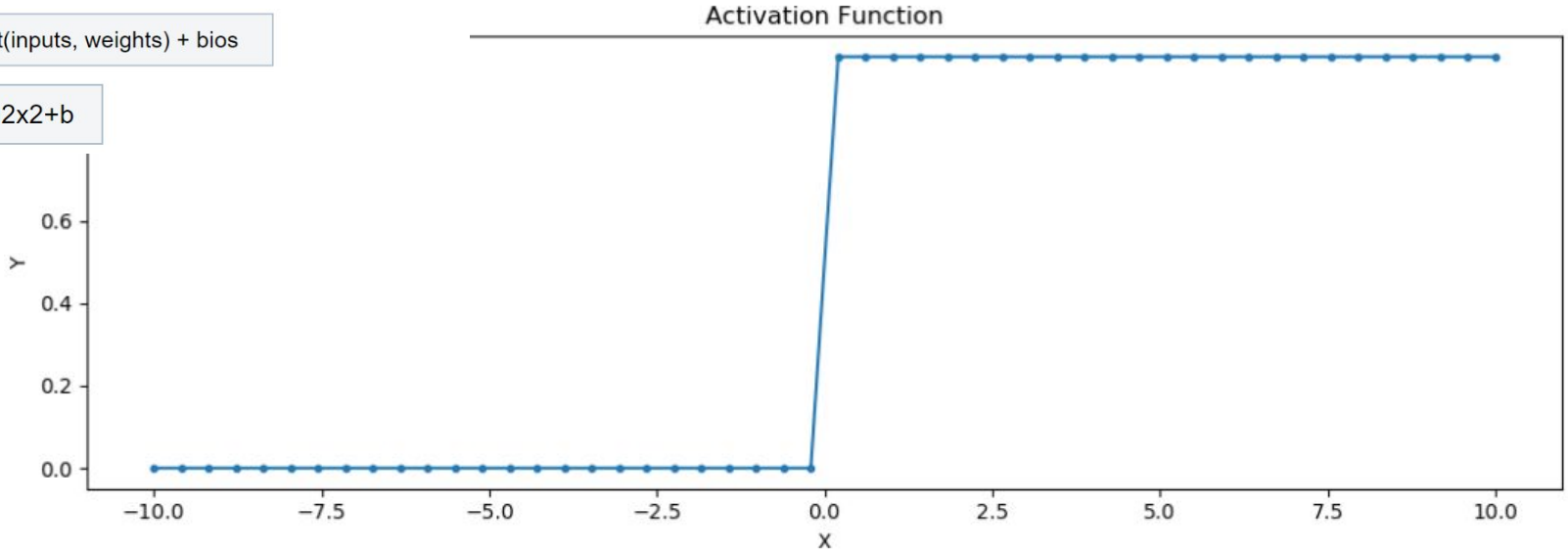
החלפת פונקציית האקטיבציה

```
def Activation(s):
    if s > 0:
      activation = 1
    else:
      activation = 0
    return activation
```

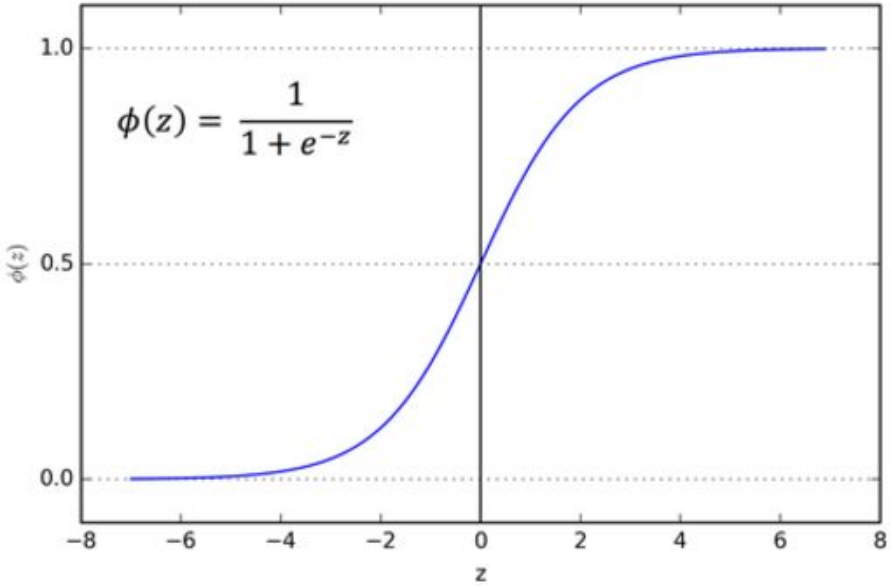


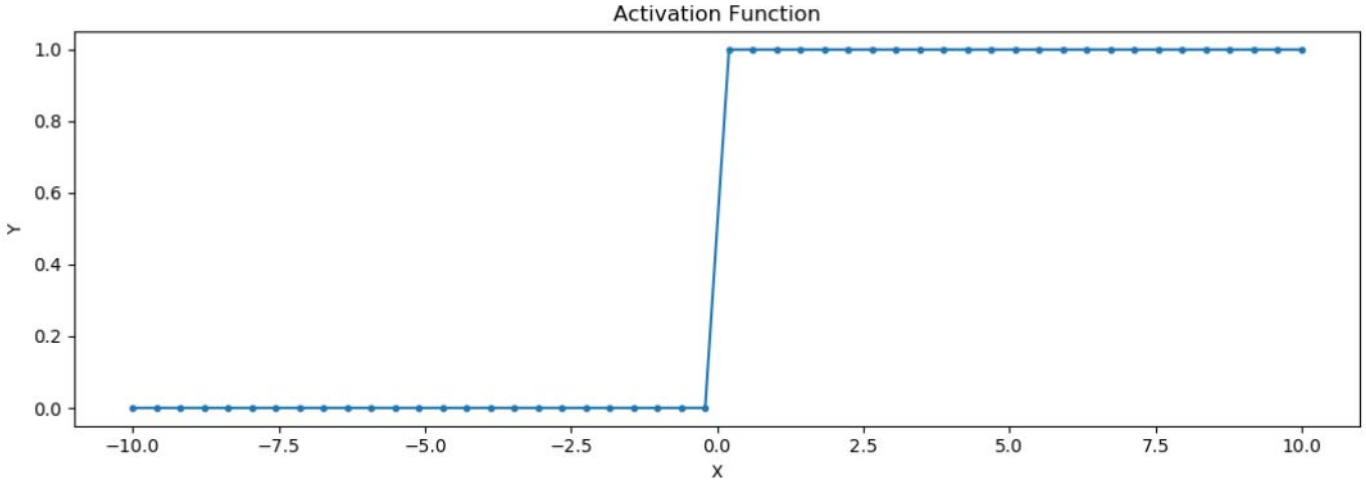

```
sum = np.dot(inputs, weights) + bios
```

$y=m_1x_1+m_2x_2+b$



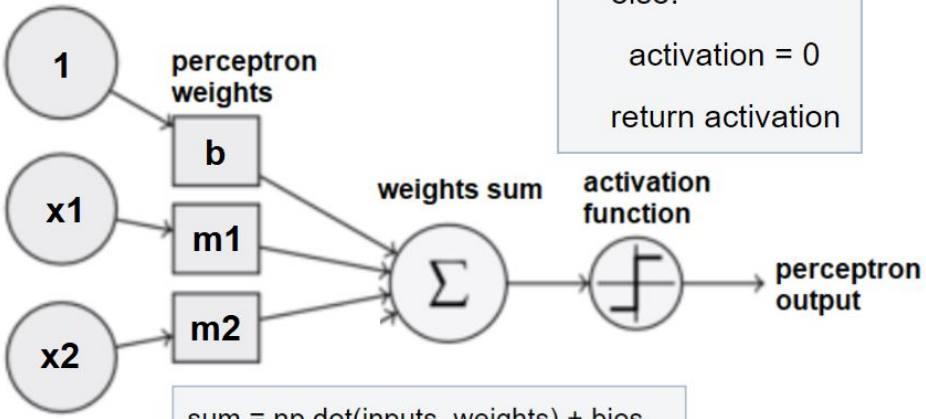
החלפת פונקציית האקטיבציה





החלפת פונקציית האקטיבציה

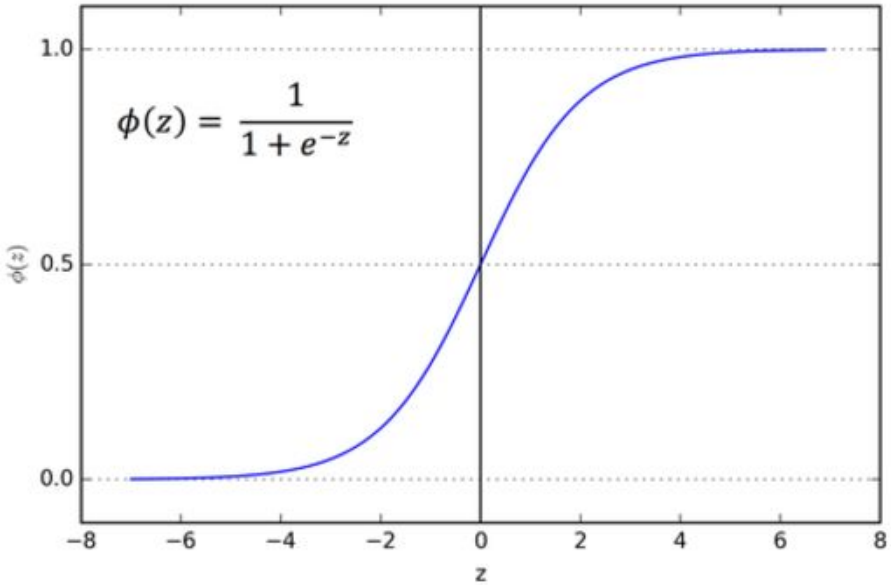
```
def Activation(s):
    if s > 0:
      activation = 1
    else:
      activation = 0
    return activation
```



sum = np.dot(inputs, weights) + bios

y=m1x1+m2x2+b

```
def sigmoid(self,x):
    return 1.0/(1.0 + np.exp(-x))
```

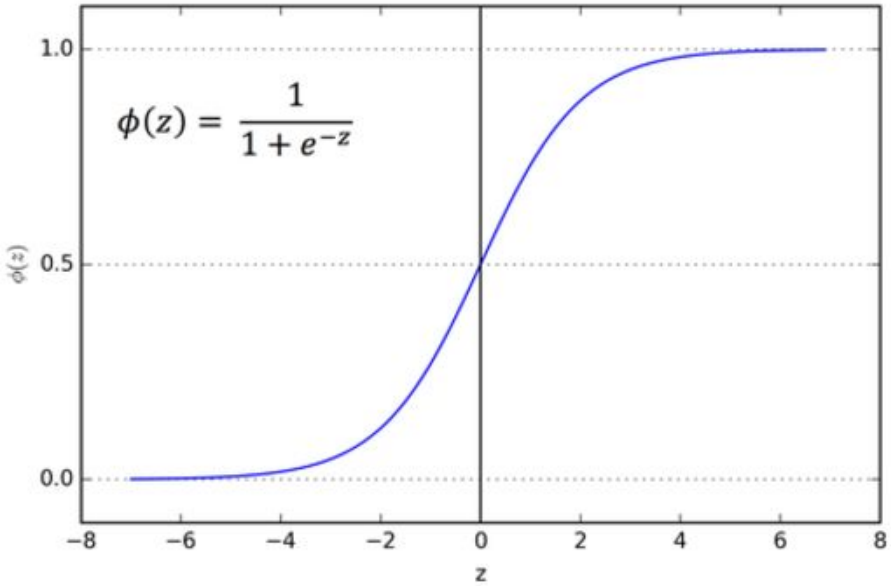


החלפת פונקציית האקטיבציה

```python
def sigmoid_derivative(self,x):
    return x * (1.0 - x)
```

```python
def sigmoid(self,x):
    return 1.0/(1.0 + np.exp(-x))
```

$$\frac{d\sigma}{dx} = x(1-x)$$



1. נחשב את השגיאה הכללית. כלומר ההפרש בין הערך הרצוי במוצא (אפס או אחד) לבין הערך שבמוצא הרשת.

2. נחשב את הנגדרת של פונקציית השגיאה עבור כל הרשת.

3. נשנה את המשקולות של נוירון המוצא (Output) בהתאם לשגיאה הכללית.

4. נשנה את המשקולות של 2 הנוירונים בשכבה הפנימית , Hidden1 Hidden2.

```
#Back Propagation
error = exp_out - predicted_output

d_predicted_output = error * self.sigmoid_derivative(predicted_output)
error_hidden_layer = d_predicted_output.dot(self.output_weights.T)
d_hidden_layer = error_hidden_layer * self.sigmoid_derivative(self.hidden_layer_output)

#Updating Weights and Biases
self.output_weights += self.hidden_layer_output.T.dot(d_predicted_output) * learningRate
self.output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * learningRate
self.hidden_weights += inpt.T.dot(d_hidden_layer) * learningRate
self.hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * learningRate
```

מימוש האלגוריתם

נחזור לבצע את כל התהליך עד שהשגיאה תהיה מינימלית

```
for _ in range(epochs):
    #Forward Propagation
    predicted_output = self.predict(inpt)

    #Back Propagation
    error = exp_out - predicted_output
    d_predicted_output = error * self.sigmoid_derivative(predicted_output)

    error_hidden_layer = d_predicted_output.dot(self.output_weights.T)
    d_hidden_layer = error_hidden_layer * self.sigmoid_derivative(self.hidden_layer_output)

    #Updating Weights and Biases
    self.output_weights += self.hidden_layer_output.T.dot(d_predicted_output) * learningRate
    self.output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * learningRate
    self.hidden_weights += inpt.T.dot(d_hidden_layer) * learningRate
    self.hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * learningRate
```

נבדוק את תהליך הלמידה

```
Initial hidden weights:
 [[0.90118317 0.46510789]
 [0.60151963 0.13516914]]

Initial hidden biases:
 [[0.52074871 0.31160313]]

Initial output weights:
 [[0.97013587]
 [0.77382653]]

Initial output biases:
 [[0.22350718]]
```

```
Final output bias:
 [[5.85010698 3.77001055]
 [5.82345896 3.76460499]]

Final output bias:
 [[-2.44010535 -5.76815595]]

Final output bias:
 [[ 7.6049737 ]
 [-8.23243882]]

Final output bias:
 [[-3.43629945]]
```

```
Output from neural network after 10,000 epochs:
 [[0.05456275]
 [0.94982758]
 [0.94990778]
 [0.05402802]]
```

```
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[1],[1],[0]])
nn = NeuralNetwork(2,2,1)
```

```
test = nn.predict(inputs)
print("\nOutput from neural network after 10,000 epochs:\n",colored(test, 'blue'))
```

מימוש שער XOR על ידי
נוירון בודד (האם זה אפשרי?
אם כן למה שלא ננסה נכתוב
אחד כזה?)