

פעילות 10 - מימוש שער XOR על ידי מספר נזירונים

מקורות:

Implementing the XOR Gate using Backpropagation in Neural Networks

<https://towardsdatascience.com/implementing-the-xor-gate-using-backpropagation-in-neural-networks-c1f255b4f20d>

Neural Networks in Python

<https://rolisz.ro/2013/04/18/neural-networks-in-python/>

Extract 10 images from the CIFAR-10 data set

<https://gist.github.com/juliensimon/273bef4c5b4490c687b2f92ee721b546>

19 lines code

<https://medium.com/@thomascourtz/19-line-line-by-line-python-perceptron-b6f113b161f3>

11 lines code

<http://iamtrask.github.io/2015/07/12/basic-python-network/>

9 lines code

<https://medium.com/technology-invention-and-more/how-to-build-a-simple-neural-network-in-9-lines-of-python-code-cc8f23647ca1>

20 lines code

<https://medium.com/@michaeldelsole/a-single-layer-artificial-neural-network-in-20-lines-of-python-ae34b47e5fef>

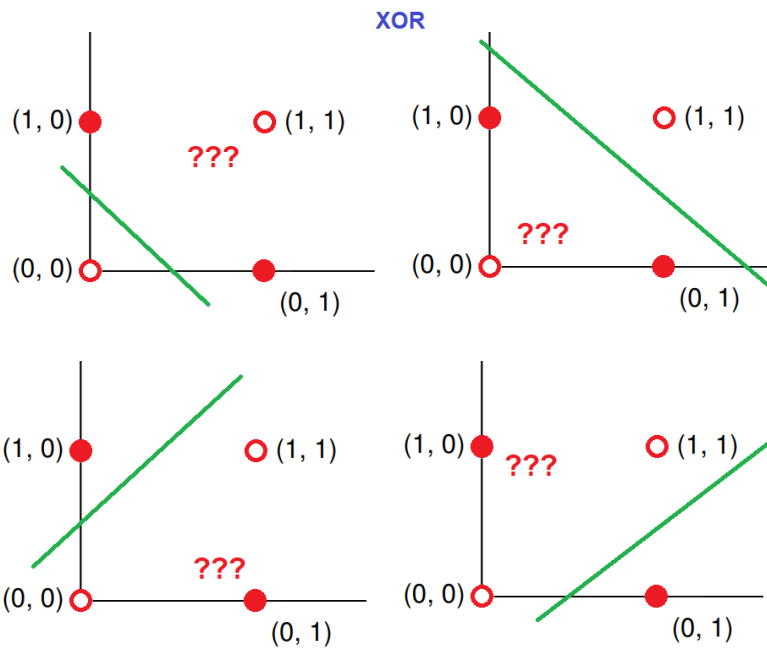
How to Create a Simple Neural Network in Python

<https://www.kdnuggets.com/2018/10/simple-neural-network-python.html>

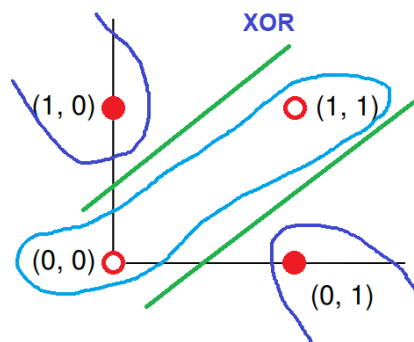
Neural Net from scratch (using Numpy)

<https://towardsdatascience.com/neural-net-from-scratch-using-numpy-71a31f6e3675>

בפעילות 9 כתבנו קוד המממש פרספטרון בודד במטרה להבין ולתרגל את עקרון הפעולה של מכונה לומדת העושה שימוש בנירון בודד. ראינו שפרספטרון מוגדר כמסווג לינארי כלומר רכיב תוכנה שבו פונקציית הסיווג מממשת משוואת קו ישר. מכאן שהצלנו ללמד פרספטרון בודד לסווג אותות בינאריים במבוא כשער לוגי מסוג AND, OR ו-NAND ולא הצלחנו לסווג לוגיקה של XOR. כי ראינו שימוש XOR אינו סיווג לינארי



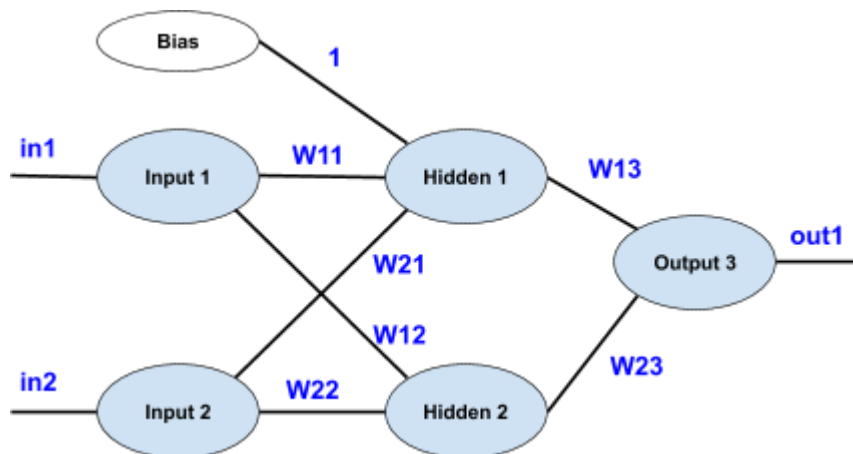
פתרון לכך יהיה מימוש רשת של מספר פרספטרונים במטרה לבנות מכונה המסוגלת ללמוד שער לוגי מסוג XOR.



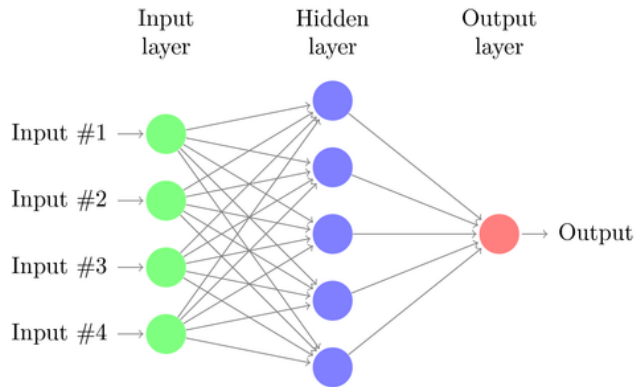
בפעילות זה נעשה שימוש בקוד תוכנה שפותח על ידי Siddhartha Dutta ופורסם בקישור הבא:

<https://gist.github.com/siddharthapdutta/a1b48b428ddeffbd5c8dc92a3cf9a625>

נבנה רשת של 5 פרספטרונים על פי המודל הבא:



המודל מבוסס על 2 פרספטרונים במבוא שיקלטו את הרמות הלוגיות של שער XOR. בנוסף 2 פרסטרונים בשכבה הפנימית ועוד פרספטרון אחד במוצא כדי לספק לנו את מוצא השער. מערך הפרסטרונים בנוי בארכיטקטורה שבה כל פרספטרון מחובר לכל שאר הפרסטרונים בשכבה הבאה. עקרון זה מכונה Fully connected neural network, ניתן לראות את עקרון החיבור באיור הבא:



מקור התמונה: <http://www.texample.net/tikz/examples/neural-network> (לשימוש חוזר עם אפשרות לשינויים)

האלגוריתם שנכתב עובד לפי השלבים הבאים:

1. נאתחל את כל המשקולות, כלומר ערכים המקשרים בין כל 2 פרספטרונים, בערכים אקראיים בתחום שבין אפס לאחד.
2. נחשב את הפלט הכללי של מוצא רשת הפרספטרונים.
3. נחשב את השגיאה הכללית. כלומר ההפרש בין הערך הרצוי במוצא (אפס או אחד) לבין הערך שבמוצא הרשת ברגע זה.
4. נשנה את המשקולות של פרספטרון המוצא (Output 3) בהתאם לשגיאה הכללית.
5. נשנה את המשקולות של 2 הפרספטרונים בשכבה הפנימית (Hidden 1, Hidden 2) בהתאם לייחוס השגיאות כפי שעשינו בשלב 4.
6. נחזור לבצע את כל התהליך מסעיף 2 עד שהשגיאה תהיה מינימלית.

לצורך כתיבת קוד התוכנה עבור האלגוריתם המתואר בשלבים שהגדרנו נכתוב מחלקה בשם NeuralNetwork הממומשת להלן:

```
class NeuralNetwork:
    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.predicted_output=0

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

```

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

def train(self, inpt, exp_out, learningRate=0.1, epochs=10000):
    for _ in range(epochs):
        #Forward Propagation
        hidden_layer_activation = np.dot(inpt,self.hidden_weights)
        hidden_layer_activation += self.hidden_bias
        hidden_layer_output = self.sigmoid(hidden_layer_activation)

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

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

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

        #Updating Weights and Biases
        self.output_weights += 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

```

בשלב הבא נבנה עצם בשם `NeuralNetwork` המחלקה `NeuralNetwork`. הפעולה הבונה תקבל במבוא את מבנה הרשת הרצוי (2,2,1). כמו כן נבנה 2 מערכים האחד בשם `inputs` שישמש אותנו כנתונים המסופקים לרשת והשני בשם `expected_output` המשמש לתיג המידע או במילים אחרות הפלט הרצוי. נזמן את הפעולה `train` כדי לאמן את הרשת. פלט הרשת יהיה המערך `predicted_output` אותו נציג כפלט.

להלן קוד התוכנית המיישם שלב זה:

```

inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[1],[1],[0]])

```

```

nn = NeuralNetwork(2,2,1)

print("\nInitial hidden weights:\n",colored(nn.hidden_weights, 'red'))
print("\nInitial hidden biases:\n",colored(nn.hidden_bias, 'red'))
print("\nInitial output weights:\n",colored(nn.output_weights, 'red'))
print("\nInitial output biases:\n",colored(nn.output_bias, 'red'))

nn.train(inputs, expected_output)

print("\nFinal output bias:\n",colored(nn.hidden_weights, 'green'))
print("\nFinal output bias:\n",colored(nn.hidden_bias, 'green'))
print("\nFinal output bias:\n",colored(nn.output_weights, 'green'))
print("\nFinal output bias:\n",colored(nn.output_bias, 'green'))

print("\nOutput from neural network after 10,000 epochs:\n",colored(nn.predicted_output, 'blue'))

```

להלן מימוש הקוד המלא של רשת נירונים ללימוד לוגיקה של שער XOR:

```

import numpy as np
from termcolor import colored

class NeuralNetwork:
    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.predicted_output=0

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

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

```

```

def train(self, inpt, exp_out, learningRate=0.1, epochs=10000):
    for _ in range(epochs):
        #Forward Propagation
        hidden_layer_activation = np.dot(inpt,self.hidden_weights)
        hidden_layer_activation += self.hidden_bias
        hidden_layer_output = self.sigmoid(hidden_layer_activation)

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

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

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

        #Updating Weights and Biases
        self.output_weights += 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

#Input datasets
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[1],[1],[0]])

nn = NeuralNetwork(2,2,1)

print("\nInitial hidden weights:\n",colored(nn.hidden_weights, 'red'))
print("\nInitial hidden biases:\n",colored(nn.hidden_bias, 'red'))
print("\nInitial output weights:\n",colored(nn.output_weights, 'red'))
print("\nInitial output biases:\n",colored(nn.output_bias, 'red'))

```

```

nn.train(inputs, expected_output)

print("\nFinal output bias:\n",colored(nn.hidden_weights, 'green'))
print("\nFinal output bias:\n",colored(nn.hidden_bias, 'green'))
print("\nFinal output bias:\n",colored(nn.output_weights, 'green'))
print("\nFinal output bias:\n",colored(nn.output_bias, 'green'))

print("\nOutput from neural network after 10,000 epochs:\n",colored(nn.predicted_output, 'blue'))

```

נקבל את הפלט הבא:

```

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]]

Final output bias:
[[5.80661794 3.68715311]
 [5.80072745 3.68601568]]

Final output bias:
[[-2.39824957 -5.63739592]]

Final output bias:
[[ 7.44137486]
 [-8.0429048 ]]

Final output bias:
[[-3.35832414]]

Output from neural network after 10,000 epochs:
[[0.05914177]
 [0.94493628]
 [0.94495532]
 [0.05979138]]

```

ניתן לראות כי בשלב הראשון מייד לאחר שיצרנו את העצם חח שנקבעו ערכי המשקלים באופן אקראי (המספרים שבאדום).

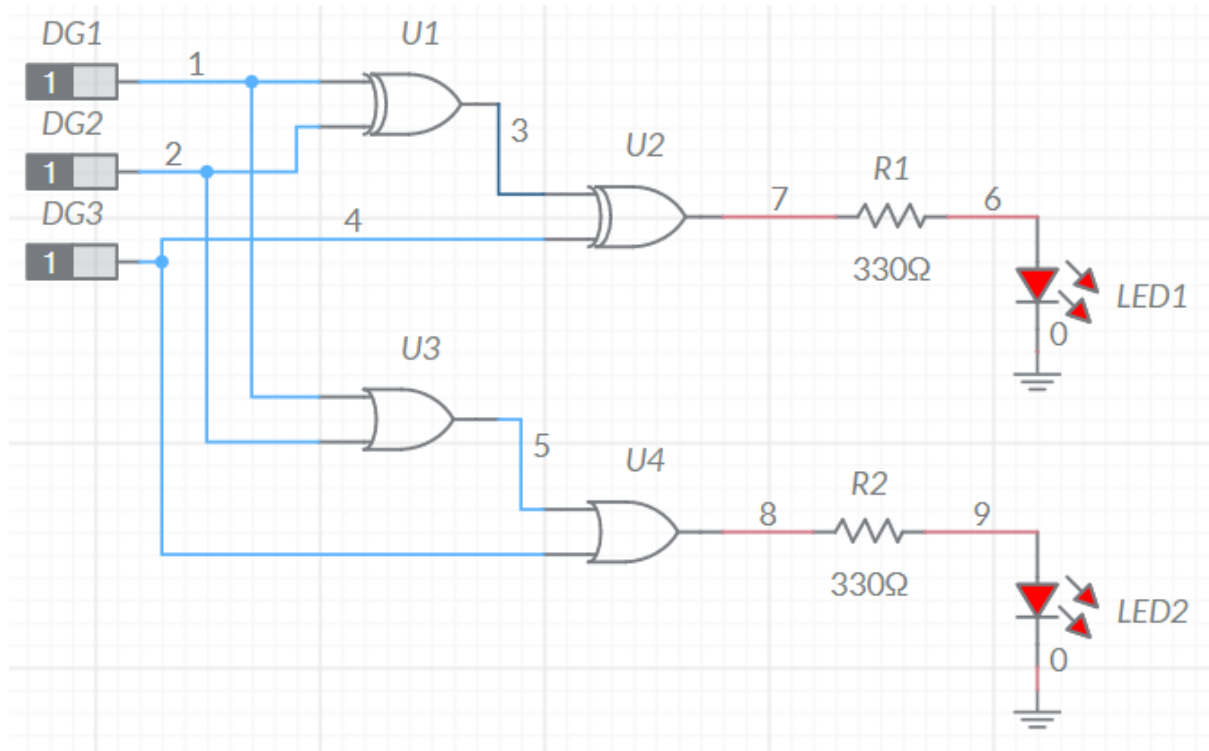
לאחר שלב האימון כלומר לאחר הפעלת הפעולה train השתנו המשקלים בהתאם לאימון המערכת (המספרים בירוק).

פלט התוכנית יהיו הערכים 0.05914177 ו-0.05979138 המייצגים אפס לוגי והערכים 0.94493628 ו-0.94495532 המייצגים אחד לוגי (המספרים הכחולים).

קיבלנו מכונה לומדת שהצליחה לאחר 10000 מחזורי אימון לאזן רשת של משקלים כך שתממש לוגיקה של XOR.

תרגיל

שנו את קוד התוכנית כדי לבנות מכונה לומדת המממשת את המעגל הלוגי הבא:



פתרון

```
inputs = np.array([
    [0,0,0],
    [0,0,1],
    [0,1,0],
    [0,1,1],
    [1,0,0],
    [1,0,1],
    [1,1,0],
    [1,1,1]])

expected_output = np.array([
    [0,0],
    [1,1],
    [1,1],
    [1,0],
    [1,1],
    [1,0],
    [1,0],
    [1,0],
```


[1,1])

nn = NeuralNetwork(3,3,2)

פלט התוכנית יהיה:

```
Initial hidden weights:
[[0.28313988 0.18121092 0.19148784]
 [0.53325367 0.67884731 0.87929008]
 [0.42779616 0.55085278 0.52707088]]

Initial hidden biases:
[[0.43497061 0.95044889 0.78710722]]

Initial output weights:
[[0.34086321 0.98019856]
 [0.98678013 0.37193812]
 [0.44883311 0.21333144]]

Initial output biases:
[[0.24574132 0.62743056]]

Final output bias:
[[5.76084941 5.60119379 2.80679961]
 [5.75454525 5.60060656 2.80678003]
 [5.75863548 5.60098595 2.80679227]]

Final output bias:
[[-2.34257844 -7.46913962 -7.07027116]]

Final output bias:
[[ 6.00219582  8.02484469]
 [ 5.38432295 -8.58665768]
 [ 3.66347857  8.09480403]]

Final output bias:
[[-3.2093862 -3.72548229]]

Output from neural network after 10,000 epochs:
[[0.06434594 0.04653115] [0,0]
 [0.96684554 0.95292849] [1,1]
 [0.96681364 0.95289993] [1,1]
 [0.99984041 0.07192422] [1,1]
 [0.9668628 0.95294376] [1,1]
 [0.99984043 0.07191851] [1,0]
 [0.99984042 0.07192223] [1,0]
 [0.99998467 0.89488692] [1,1]
 [1,1]
 [1,0]
 [1,0]
 [1,1]]
```

תרגיל

הוסיפו לקוד המחלקת NeuralNetwork פעולה בשם predict המחלקה תקבל 3 ערכים המייצגים מידע לוגי המסופק לרשת הנוירונים שכתבנו בפעילות הקודמת. הפעולה תחזיר את פלט הרשת לאותם ערכים.

על התוכנית לקלוט את 3 הערכים הבינאריים מהמשתמש.

התוכנה תפסיק לקלוט כאשר אחד הערכים הנקלט מהמשתמש אינו מייצג רמה לוגית.

```

import numpy as np
from termcolor import colored

class NeuralNetwork:
    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.predicted_output=0

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

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

    def train(self, inpt, exp_out, learningRate=0.1, epochs=10000):
        for _ in range(epochs):
            #Forward Propagation
            hidden_layer_activation = np.dot(inpt,self.hidden_weights)
            hidden_layer_activation += self.hidden_bias
            hidden_layer_output = self.sigmoid(hidden_layer_activation)

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

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

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

```

```

#Updating Weights and Biases
self.output_weights += 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

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

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

inputs = np.array([ [0,0,0],
                    [0,0,1],
                    [0,1,0],
                    [0,1,1],
                    [1,0,0],
                    [1,0,1],
                    [1,1,0],
                    [1,1,1]])
expected_output = np.array([[0,0],
                             [1,1],
                             [1,1],
                             [1,0],
                             [1,1],
                             [1,0],
                             [1,0],
                             [1,1]])

nn = NeuralNetwork(3,3,2)
nn.train(inputs, expected_output)

```

```

A = int(input("Entar A (1 or 0):"))
B = int(input("Entar B (1 or 0):"))
C = int(input("Entar C (1 or 0):"))
while (A==1 or A==0) and (B==1 or B==0) and (C==1 or C==0):
    tests = np.array([ [A,B,C] ])
    print(colored(nn.predict(tests), 'blue'))
    A = int(input("Entar A (1 or 0):"))
    B = int(input("Entar B (1 or 0):"))
    C = int(input("Entar C (1 or 0):"))

```

דוגמה לפלט התוכנית:

```

Entar A (1 or 0):1
Entar B (1 or 0):1
Entar C (1 or 0):1
[[0.99997403 0.91514828]]
Entar A (1 or 0):0
Entar B (1 or 0):0
Entar C (1 or 0):0
[[0.04963973 0.01574641]]
Entar A (1 or 0):0
Entar B (1 or 0):0
Entar C (1 or 0):1
[[0.97360728 0.95856444]]
Entar A (1 or 0):0
Entar B (1 or 0):1
Entar C (1 or 0):0
[[0.97352764 0.95849873]]
Entar A (1 or 0):1
Entar B (1 or 0):1
Entar C (1 or 0):0
[[0.9999102 0.05840762]]
Entar A (1 or 0):0
Entar B (1 or 0):0
Entar C (1 or 0):2

```

תנאי השימוש

תנאי השימוש במסמך זה הם לפי הסטנדרט הבא:

You are free:

to Share – to copy, distribute and transmit the material
to Remix – to adapt the material

Under the following conditions:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial — You may not use the material for commercial purposes.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.