

תרגול 7 - רשת נוירונים רדודה (מימוש שער XOR)

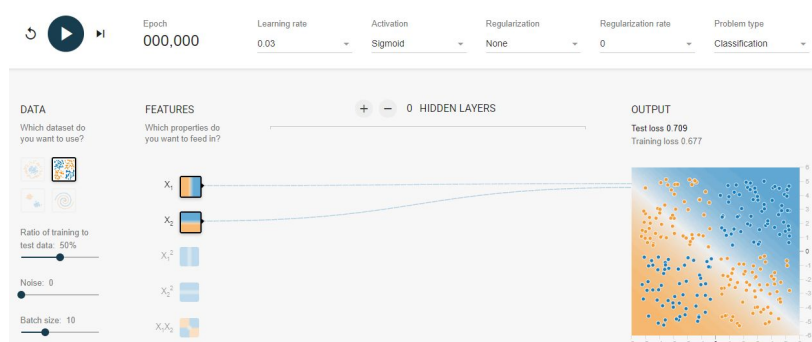
מימוש שער XOR על ידי נוירון בודד (האם זה אפשרי? אם כן למה שלא ננסה לכתוב אחד כזה?)

שלב א' (בדיקת מימוש XOR על ידי נוירון בודד)

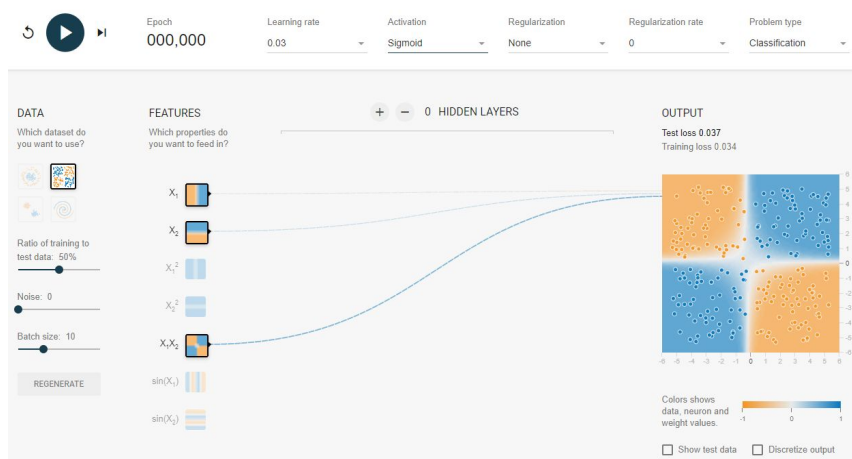
היכנסו לאתר:

<https://playground.tensorflow.org/>

והריצו את הסימולציה הבאה:



הוסיפו לסימולציה שלכם Feature הכולל את מכפלת X_1 ב- X_2 ובדקו שוב.



ענו על השאלות הבאות:

1. האם נוירון בודד (ללא תוספות) מסוגל לבצע סיווג נתונים המייצגים לוגיקת XOR? נמק את תשובתך.
2. האם הוספת Feature הכולל את מכפלת X_1 ב- X_2 משנה את התוצאה הסיווג?
3. האם הוספת Feature הכולל את מכפלת X_1 ב- X_2 משנה את תכונות היסוד של הנוירון? נמק את תשובתך.
4. כיצד לפי דעתכם הוספת ה- Feature השפיע על יכולת הסיווג של הנוירון?

שלב ב' (בדיקת הסימולציה בקוד)

השתמשו במחלקה Perceptron שכתבנו במפגש הקודם כדי לבדוק האם הוספת Feature הכולל את מכפלת X_1 ב- X_2 אפשר לכתב מכונה לומדת שמשווגת לוגיקה של XOR תוך כדי שימוש בפרספטרון בודד. לרשותכם המחלקה Perceptron כפי שלמדנו.

```
class Perceptron(object):
    def __init__(self, numOfInputs, epochs=200, learningRate=0.01):
        self.epochs = epochs
        self.learningRate = learningRate
        self.weights = np.zeros(numOfInputs)
        self.bios = 1

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

    def predict(self, inputs):
        sum = np.dot(inputs, self.weights) + self.bios
        out = self.Activation(sum)
        return out

    def train(self, inputs, labels):
        for _ in range(self.epochs):
            for i in range(len(inputs)):
                prd = self.predict(inputs[i])
                self.weights -= (prd - labels[i]) * inputs[i] * self.learningRate
                self.bios -= (prd - labels[i]) * self.learningRate
```

לאחר שהמחלקה Perceptron מוכנה ביצעו מספר אימונים עליה על פי המאפיינים הבאים:

1. אימון למידת לוגיקת XOR ללא הוספת Feature
2. אימון למידת לוגיקת XOR יחד עם Feature הכולל את מכפלת X_1 ב- X_2

דווחו מה מתוך האימונים הצליח.

שלב ג' (כתיבת גרסה חדשה למחלקה Perceptron)

המחלקה Perceptron שכתבנו בשיעור הקודם משתמשת פונקציית אקטיבציה שאינה גזירה. על כן בקוד המחלקה שמימשנו התבססנו על חישוב הפרמטרים של m ו- b על ידי הנגזרת COST לפי m ואז הנגזרת COST לפי b ולא הכנסנו את פונקציית האקטיבציה לחישוב ה-COST על כן גם לא גזרנו אותה!

במשימה זו נכתוב גרסה חדשה של המחלקה Perceptron תוך כדי התבססות על המחלקה NeuralNetwork כפי שלמדנו בשיעור זה. כלומר Perceptron העושה שימוש בפונקציית האקטיבציה מסוג .sigmoid

לרשותכם המחלקה NeuralNetwork כפי שלמדנו.

```
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.hidden_layer_output = np.random.uniform(size=(1, outputLayerNeurons))

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

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

    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)

    def train(self, inpt, exp_out, learningRate=0.1, epochs=10000):
        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
```