

## פעילות 11 - רשת נוירונים לסיווג תמונות

מקורות:

<https://www.neuraldesigner.com/learning/examples/iris-flowers-classification>

<https://www.kaggle.com/uciml/iris>

<https://stackoverflow.com/questions/3518778/how-do-i-read-csv-data-into-a-record-array-in-numpy>

<https://archive.ics.uci.edu/ml/datasets/Iris>

[https://commons.wikimedia.org/wiki/Iris\\_\(Iridaceae\)](https://commons.wikimedia.org/wiki/Iris_(Iridaceae))

בפעילות 9 כתבנו קוד המממש פרספטרון בודד במטרה להבין ולתרגל את עקרון הפעולה של מכונה לומדת העושה שימוש בנוירון בודד. ראינו שפרספטרון מוגדר כמסווג לינארי כלומר רכיב תוכנה שבו פונקציית הסיווג מממשת משוואת קו ישר.

בפעילות 10 כתבנו מחלקה בשם NeuralNetwork העושה שימוש בעקרונות שכבר למדנו שמימשנו בקוד פרספטרון כדי לבנות רשת של נוירונים מלאכותיים Artificial neural networks - ANN. השתמשנו במחלקה NeuralNetwork כדי לבנות מכונה לומדת המממשת שער לוגי מסוג XOR.

בפעילות זו נשתמש באותה מחלקה שכתבנו בפעילות 10 כדי ללמוד בעזרתה כיצד ניתן ללמד מכונה לומדת לבצע סיווג classification של נתונים.

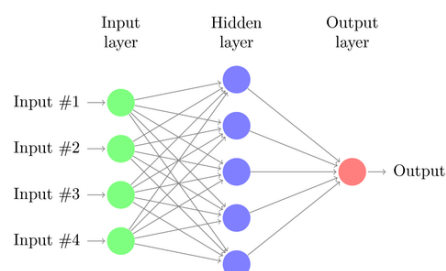
נחלק את הפעילות ל-3 שלבים:

- בשלב הראשון נכתוב מכונה לומדת לסיווג מערך נקודות של פונקציה לינארי על גבי מערכת צירים קרטזית דו-מימדית.
- בשלב השני נתרגל כתיבת מכונה לומדת לסיווג מערך נקודות של פרבולה על גבי מערכת צירים קרטזית דו-מימדית.
- בשלב השלישי ננסה, (תבינו בסוף הפעילות מדוע..) לבנות מכונה לומדת לסיווג 3 סוגים של פרחים תוך שימוש במחלקה שלנו NeuralNetwork

מטרת הפעילות היא להכין את תשתית הידע לעבודה עם מחלקות מעשיות שפותחו על ידי גופים גדולים כמו Keras ו-TensorFlow.

### המחלקה NeuralNetwork

מחלקה זו מממשת מערך פרסטרונים הבנוי בארכיטקטורה שבה כל פרספטרון מחובר לכל שאר הפרסטרונים בשכבה הבאה. עקרון זה מכונה Fully connected neural network, ניתן לראות את עקרון החיבור באיור הבא:

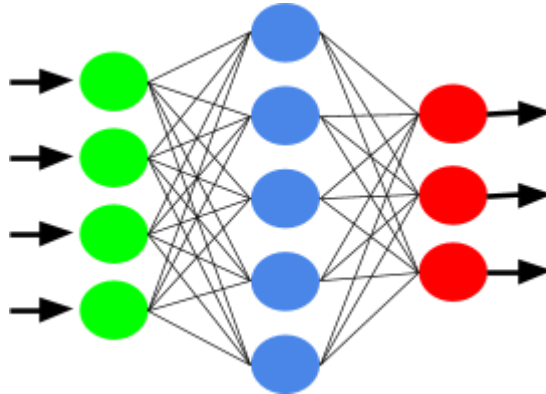


מקור התמונה: <http://www.texample.net/tikz/examples/neural-network> (לשימוש חוזר עם אפשרות לשינויים)

למחלקה זו יכולת להגדיר שכבת מבוא אחת הכוללת מספר מבואות (Input layer), שכבה אחת של נוירונים חבויים (Hidden layer) ושכבה של נוירונים במוצא (Output layer) קביעת מערך הנוירונים המרכיבים את הרשת נעשה על ידי אתחול עצם המחלקה המקבל את מספרי הנוירונים בכל שכבה. לדוגמה:

```
nn = NeuralNetwork(4,5,3)
```

הוראה זו מאתחלת עצם בשם nn מטיפוס המחלקה NeuralNetwork המייצג רשת הכוללת 4 מבואות, 5 נוירונים בשכבה האמצעית ו-3 מוצאים.



להלן מימוש המחלקה:

```
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)

```

המחלקה כוללת 4 פעולות נוסף לפעולה הבונה והם:

- הפעולות sigmoid ו- sigmoid\_derivative משמשות פעולות פנימיות במחלקה לצורך מימוש פונקציית התמסורת של הפרספטרון.
- הפעולה train המשמשת לאימון המכונה.
- הפעולה predict המשמשת לניבוי תוצאות לאחר שלב הלמידה.

## במשימה 1 מכונה לומדת לסיווג נקודות של פונקציה לינארית.

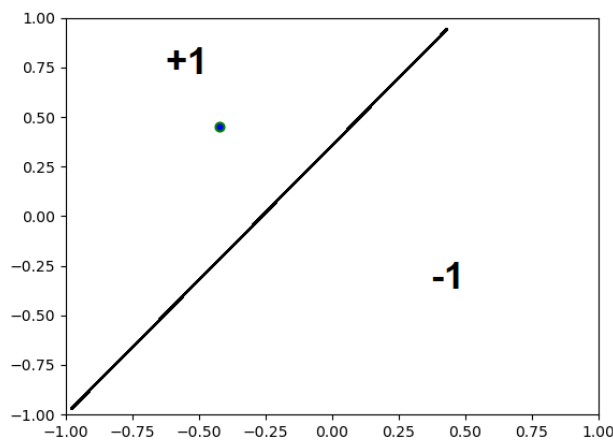
במשימה זו נכתוב מכונה לומדת לסיווג מערך נקודות של פונקציה לינארית על גבי מערכת צירים קרטזית דו-מימדית. לצורך כך נכתב מחלקה חדשה בשם point המייצגת נקודה. להלן מימוש המחלקה:

```
class point(object):
    def __init__(self,m=1,b=0):
        self.x = np.random.uniform(-1,1)
        self.y = np.random.uniform(-1,1)
        if (self.x)*m+b > self.y:
            self.label = 1
        else:
            self.label = -1
```

כפי שניתן לראות למחלקה פעולה בונה המקבלת 2 פרמטרים האחד m והשני b המייצגים את הפרמטרים של משוואת קו ישר:

$$y = mx + b$$

פעולה זו מגרילה 2 מספרים אקראיים בין מינוס אחד לאחד עבור x ו-y ומסמנת במאפיין label האם הנקודה נמצאת מעל לקו או מתחתיו.



כמו כן נכתוב מחלקה נוספת שתשמש אותנו כמערך נקודות גם בשלב האימון של המכונה הלומדת וגם בשלב ההרצה של המכונה. להלן מימוש המחלקה:

```
class listOfpoint:
    def __init__(self,numberOfPoints,m=1,b=0):
        self.points = []
        for _ in range(numberOfPoints):
            self.points.append(point(m,b))
```

```

self.allXY=[]
self.allLBL=[]
for item in self.points:
    tmpXY = np.array([item.x , item.y])
    tmpLBL = np.array([item.label])
    self.allXY.append(tmpXY)
    self.allLBL.append(tmpLBL)
self.allXY=np.array(self.allXY)
self.allLBL=np.array(self.allLBL)

```

המחלקה listOfpoint כולל פעולה בונה המקבלת 3 פרמטרים: הראשון מספר הנקודות הרצוי ושני הפרמטרים הנוספים הם הערכים m ו-b של הקו הלינארי. המחלקה מחזירה מערך של נקודות (מערך של עצמים מטיפוס point) נדגים תוכנית מלאה המחוללת מערך 10 נקודות אקראיות כאשר לכל נקודה ערך המייצג האם הנקודה מעל לקו או מתחת לקו הבא:

$$y = 2x + 0.5$$

להלן הקוד:

```

import numpy as np

class point(object):
    def __init__(self,m=1,b=0):
        self.x = np.random.uniform(-1,1)
        self.y = np.random.uniform(-1,1)
        if (self.x)*m+b > self.y:
            self.label = 1
        else:
            self.label = -1
    def __repr__(self):
        return "\tx="+str(self.x)+"\ty="+str(self.y)+"\tlabel= "+str(self.label)+"\n"

class listOfpoint:
    def __init__(self,numberOfPoints,m=1,b=0):
        self.points = []
        for _ in range(numberOfPoints):
            self.points.append(point(m,b))
        self.allXY=[]

```

```

self.allLBL=[]
for item in self.points:
    tmpXY = np.array([item.x , item.y])
    tmpLBL = np.array([item.label])
    self.allXY.append(tmpXY)
    self.allLBL.append(tmpLBL)
self.allXY=np.array(self.allXY)
self.allLBL=np.array(self.allLBL)

```

```

MyPoints = listOfpoint(10,2,0.5)
print(MyPoints.points)

```

נקבל את הפלט הבא:

```

[  x=-0.28439100924950367  y=-0.2820753536931495  label=  1
,  x=-0.35631015836246815  y=0.7505372329924922  label=-1
,  x=-0.24806433097509384  y=-0.5992587610711555  label=  1
,  x=-0.7070520356048955  y=-0.691382012185455  label=-1
,  x=0.9910452123662656  y=0.3966342235814897  label=  1
,  x=0.7156820071896701  y=0.4560158117871338  label=  1
,  x=-0.3080654346370524  y=0.4251529954138644  label=-1
,  x=0.8532944402716673  y=0.6270611301873141  label=  1
,  x=0.07471091317154  y=-0.4301899817122472  label=  1
,  x=0.344039770853259  y=0.7488701136698508  label=  1
]

```

בגלל שקשה להבין כמות כזו של נתונים נוסיף למחלקה `listOfpoint` פעולה בשם `drawTrainingPoints` שמציגה גרף הכולל את מערך הנקודות כאשר כל נקודה שהוגדרה במאפיין `label` כאחד תקבל צבע ירוק וכל נקודה שתקבל ערך מינוס אחד צבע אדום. להלן קוד התוכנית כולל מימוש הפעולה `drawTrainingPoints`:

```

import numpy as np
import matplotlib.pyplot as plt

class point(object):
    def __init__(self,m=1,b=0):
        self.x = np.random.uniform(-1,1)
        self.y = np.random.uniform(-1,1)
        if (self.x)*m+b > self.y:
            self.label = 1
        else:
            self.label = -1

```

```

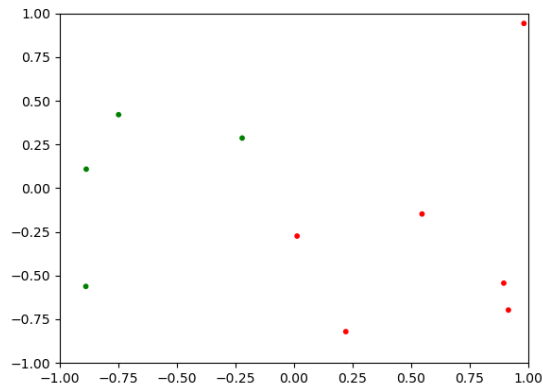
class listOfpoint:
    def __init__(self,numberOfPoints,m=1,b=0):
        self.points = []
        for _ in range(numberOfPoints):
            self.points.append(point(m,b))
        self.allXY=[]
        self.allLBL=[]
        for item in self.points:
            tmpXY = np.array([item.x , item.y])
            tmpLBL = np.array([item.label])
            self.allXY.append(tmpXY)
            self.allLBL.append(tmpLBL)
        self.allXY=np.array(self.allXY)
        self.allLBL=np.array(self.allLBL)

    def drawTrainingPoints(self):
        categories = []
        colormap = np.array(['r', 'g'])
        px = []
        py = []
        for i in range(len(self.points)):
            px.append(self.points[i].x)
            py.append(self.points[i].y)
            if self.points[i].label > 0:
                categories.append(0)
            else:
                categories.append(1)
        plt.scatter(px, py, s=10, c=colormap[categories])
        plt.axis([-1, 1, -1, 1])
        plt.show()

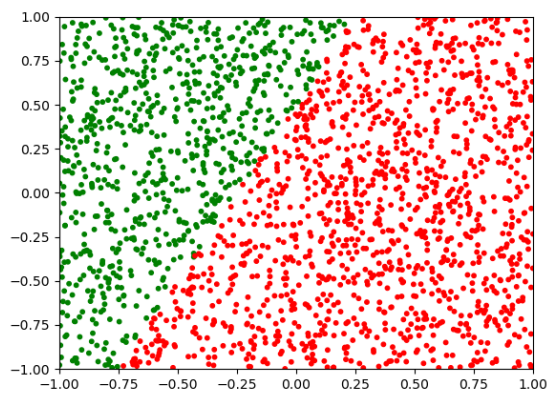
MyPoints = listOfpoint(10,2,0.5)
MyPoints.drawTrainingPoints()

```

## להלן פלט התוכנית:

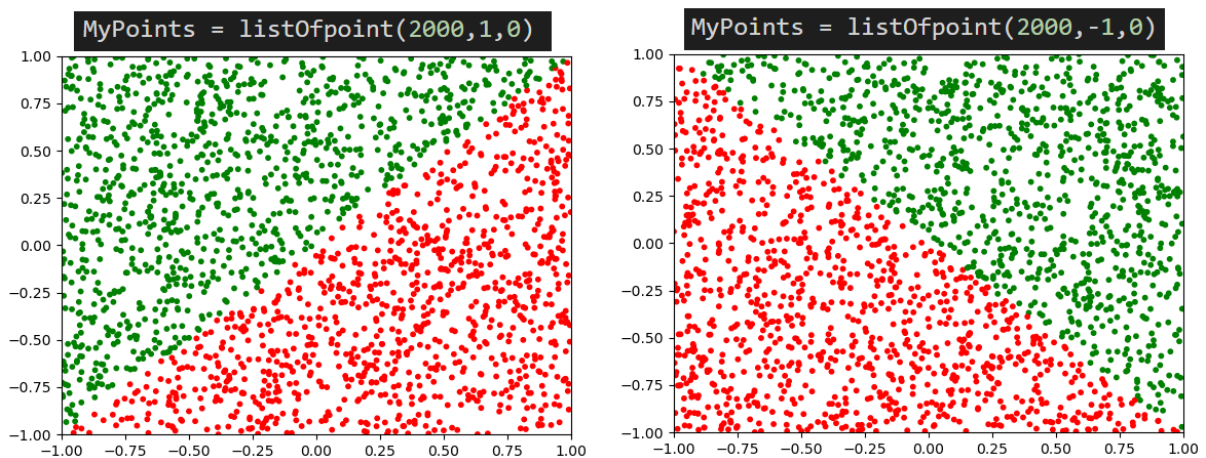


עכשיו כשיש לנו כלי גרפי להצגת נקודות נבדוק את המחלקה על 2000 נקודות ונקבל את הפלט הבא:



מאיור זה ניתן לראות את קו הגמה כפי שהגדרנו אותו בקוד התוכנית.

נשנה את משוואת הקו ונראה כיצד היא משפיעה על פלט התוכנית:



השלב הבא במשימה יהיה לזמן את המחלקה NeuralNetwork כדי לבנות את רשת הניורונים. לספק לה מערך של נקודות מתויגות (נקודה שיש לה ערך ל- $X$  ו- $Y$  כמו גם תגית label המציינת האם הנקודה מעל הקו או מתחתיו). המכונה תכנס ללמידה על ידי הפעולה train שלאחר מכן תהיה המכונה שלנו מוכנה לבדיקה.

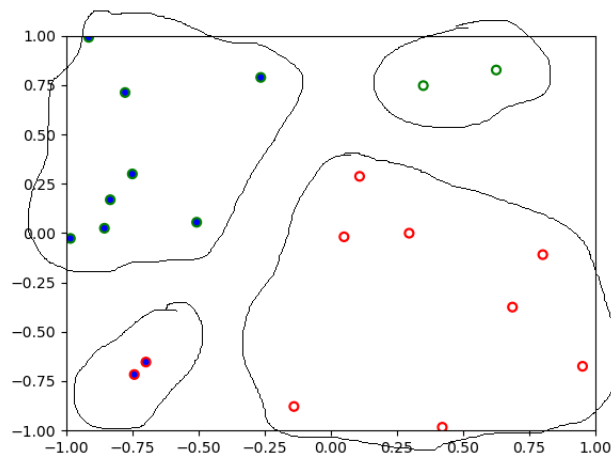


```
#-----Train the Neural Network-----
train_points = listOfpoint(200,2,0.5)
train_points.drawTrainingPoints()
nn = NeuralNetwork(2,2,1)
nn.train(train_points.allXY, train_points.allLBL)
```

כדי לבדוק אם המכונה למדה לזהות נקודה ביחס לקו המגמה. ניצור מערך נקודות חדש (כזה שהמכונה לא פגשה בו) ונזמן את הפעולה predict כדי לבדוק מה למדה. להלן הקוד:

```
#-----TEST the Neural Network-----
test_points = listOfpoint(200,2,0.5)
predict_values = nn.predict(test_points.allXY)
```

כדי לבדוק כמה נקודות מיתוך ה- 200 נקודות במערך בדיקה המכונה הצליחה התוכנה לסווג ניצור פעולה נוספת במחלקה listOfpoint שתציג גרף הכולל נקודות ארבע צורות שונות:



- עיגול אדום עם לבן - זיהוי תקין.
- עיגול ירוק עם כחול - זיהוי תקין.
- עיגול אדום עם כחול - זיהוי שגוי של נקודה.
- עיגול ירוק עם לבן - זיהוי שגוי של נקודה.

להלן מימוש הפעולה:

```
def drawMistakesPoints(self,predict_values):
    colormap1 = np.array(['r', 'g'])
    colormap2 = np.array(['w', 'b'])
    categories = []
    for i in range(len(self.allXY)):
        if self.points[i].label > 0:
            categories.append(0)
```

```

else:
    categories.append(1)

plt.scatter(self.allXY[:, 0],self.allXY[:, 1], s=40, c=colormap1[categories])

categories = []
for i in range(len(predict_values)):
    if predict_values[i] > 0.5:
        categories.append(0)
    else:
        categories.append(1)
plt.scatter(self.allXY[:, 0],self.allXY[:, 1], s=10, c=colormap2[categories])
plt.axis([-1, 1, -1, 1])
plt.show()

```

בדיקת כל הקוד יחד.  
להלן קוד התוכנה הסופי למשימה זו:

```

import numpy as np
import matplotlib.pyplot as plt

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)

class point(object):
    def __init__(self,m=1,b=0):
        """
        y=mx+b
        """
        self.x = np.random.uniform(-1,1)
        self.y = np.random.uniform(-1,1)
        if (self.x)*m+b > self.y:
            self.label = 1
        else:
            self.label = -1
```

```

class listOfpoint:
    def __init__(self,numberOfPoints,m=1,b=0):
        """
        Get number Of Points
        Get m and b parameter in y=mx+b (linear function line)
        Return list of point class
        """
        self.points = []
        for _ in range(numberOfPoints):
            self.points.append(point(m,b))

        self.allXY=[]
        self.allLBL=[]
        for item in self.points:
            tmpXY = np.array([item.x , item.y])
            tmpLBL = np.array([item.label])
            self.allXY.append(tmpXY)
            self.allLBL.append(tmpLBL)
        self.allXY=np.array(self.allXY)
        self.allLBL=np.array(self.allLBL)

    def drawTrainingPoints(self):
        """
        Draw the training inputs and the training label
        """
        categories = []
        colormap = np.array(['r', 'g'])
        px = []
        py = []
        for i in range(len(self.points)):
            px.append(self.points[i].x)
            py.append(self.points[i].y)
            if self.points[i].label > 0:
                categories.append(0)
            else:
                categories.append(1)
        plt.scatter(px, py, s=10, c=colormap[categories])
        plt.axis([-1, 1, -1, 1])
        plt.show()

```

```

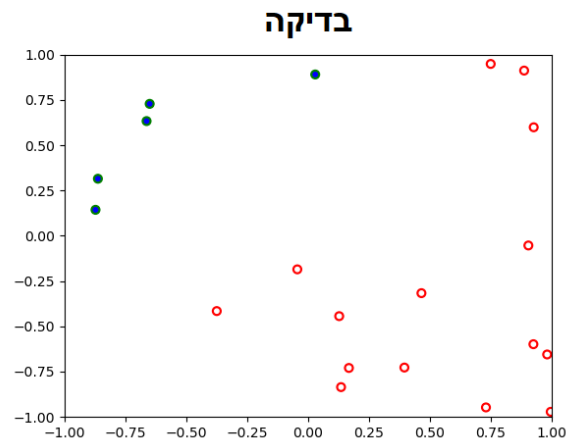
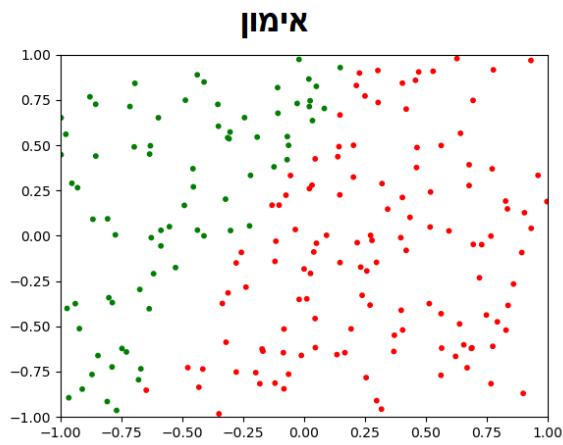
def drawMistakesPoints(self,predict_values):
    """
    Draw a comparison of the correct answers to the mistakes
    """
    colormap1 = np.array(['r', 'g'])
    colormap2 = np.array(['w', 'b'])
    #Draw the correct answers
    categories = []
    for i in range(len(self.allXY)):
        if self.points[i].label > 0:
            categories.append(0)
        else:
            categories.append(1)

    plt.scatter(self.allXY[:, 0],self.allXY[:, 1], s=40, c=colormap1[categories])
    #-----
    categories = []
    for i in range(len(predict_values)):
        if predict_values[i] > 0.5:
            categories.append(0)
        else:
            categories.append(1)
    plt.scatter(self.allXY[:, 0],self.allXY[:, 1], s=10, c=colormap2[categories])
    plt.axis([-1, 1, -1, 1])
    plt.show()

#-----Train the Neural Network-----
train_points = listOfpoint(200,2,0.5)
train_points.drawTrainingPoints()
nn = NeuralNetwork(2,2,1)
nn.train(train_points.allXY, train_points.allLBL)
#-----TEST the Neural Network-----
test_points = listOfpoint(20,2,0.5)
predict_values = nn.predict(test_points.allXY)
test_points.drawMistakesPoints(predict_values)

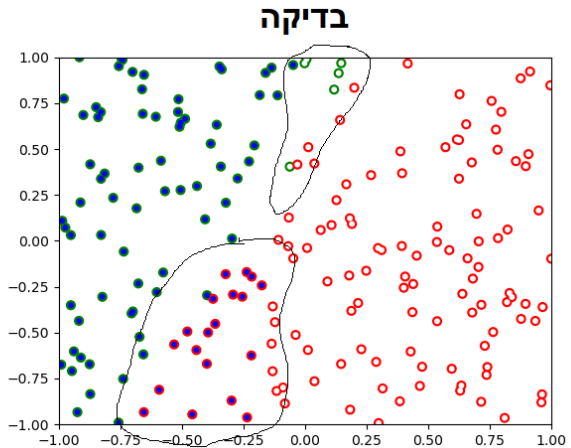
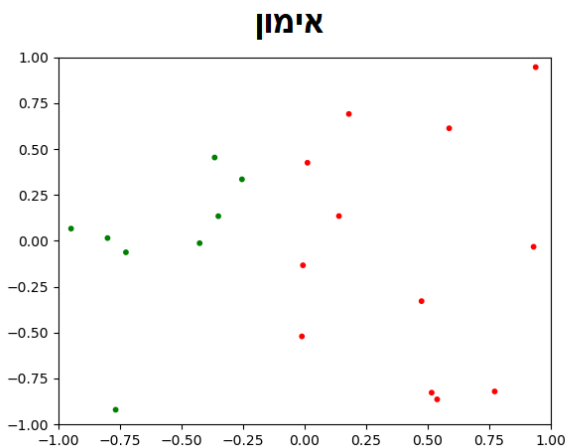
```

להלן פלט התוכנית:

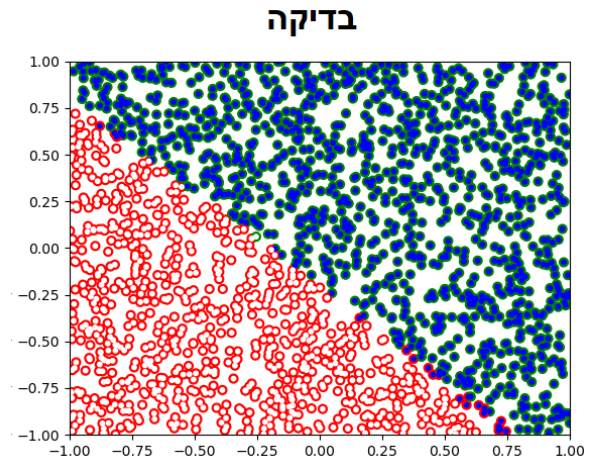
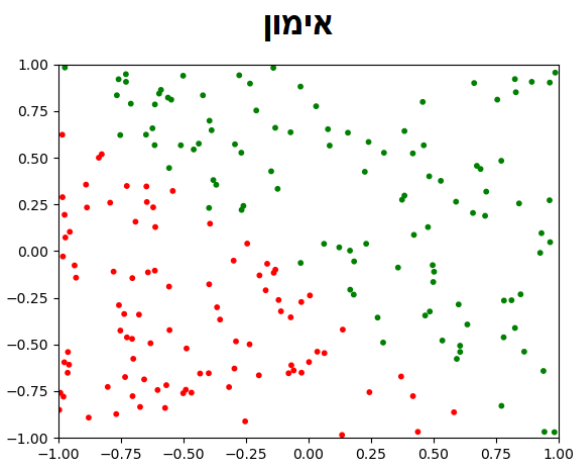


ניתן לראות שהמערכת למדה כל כך טוב שאין לה שגיאות!!!

נאתגר את המערכת ביותר נקודות בדיקה ופחות אימון:



כאשר סיפקנו לערכת 20 נקודות לאימון ו- 200 נקודות בדיקה ניתן לראות שהמערכת לא הצליחה ללמוד מספיק טוב כך שקיבלנו המון נקודות שגויות. נראה ש- 200 נקודות אימון מספיקות כדי לאמן רשת מסוג זה.



## תרגיל

שנו את האלגוריתם של המחלקות point ו- listOfpoint כדי לבדוק האם אותה רשת נוירונים שכתבנו יכולה לזהות נקודות המסווגות ביחס לפרבולה.

$$y = ax^2 + bx + c$$

## פתרון

להלן פתרון שלם של התרגיל:

```
import numpy as np
import matplotlib.pyplot as plt
#np.random.seed(1000)

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)

class parabolaPoint(object):
    def __init__(self,a=1,b=0,c=0):
        """
        y=a*x*x+b*x+c
        """
        self.x = np.random.uniform(-1,1)
        self.y = np.random.uniform(-1,1)
        if (self.x)*(self.x)*a + b*(self.x) + c > self.y:
            self.label = 1
        else:
            self.label = -1

class ParabolaListOfpoint:
    def __init__(self,numberOfPoints,a=1,b=0,c=0):
        """
        Get number Of Points
        Get a , b and c parameter in y=axx+bx+c (Parabola function)
        Return list of point class
        """
        self.points = []

```



```

for _ in range(numberOfPoints):
    self.points.append(parabolaPoint(a,b,c))

self.allXY=[]
self.allLBL=[]
for item in self.points:
    tmpXY = np.array([item.x , item.y])
    tmpLBL = np.array([item.label])
    self.allXY.append(tmpXY)
    self.allLBL.append(tmpLBL)
self.allXY=np.array(self.allXY)
self.allLBL=np.array(self.allLBL)

def drawTrainingPoints(self):
    """
    Draw the training inputs and the training label
    """
    categories = []
    colormap = np.array(['r', 'g'])
    px = []
    py = []
    for i in range(len(self.points)):
        px.append(self.points[i].x)
        py.append(self.points[i].y)
        if self.points[i].label > 0:
            categories.append(0)
        else:
            categories.append(1)
    plt.scatter(px, py, s=10, c=colormap[categories])
    plt.axis([-1, 1, -1, 1])
    plt.show()

def drawMistakesPoints(self,predict_values):
    """
    Draw a comparison of the correct answers to the mistakes
    """
    colormap1 = np.array(['r', 'g'])
    colormap2 = np.array(['w', 'b'])
    #Draw the correct answers
    categories = []

```

```

for i in range(len(self.allXY)):
    if self.points[i].label > 0:
        categories.append(0)
    else:
        categories.append(1)

plt.scatter(self.allXY[:, 0],self.allXY[:, 1], s=40, c=colormap1[categories])
#-----
categories = []
for i in range(len(predict_values)):
    if predict_values[i] > 0.5:
        categories.append(0)
    else:
        categories.append(1)
plt.scatter(self.allXY[:, 0],self.allXY[:, 1], s=10, c=colormap2[categories])
plt.axis([-1, 1, -1, 1])
plt.show()

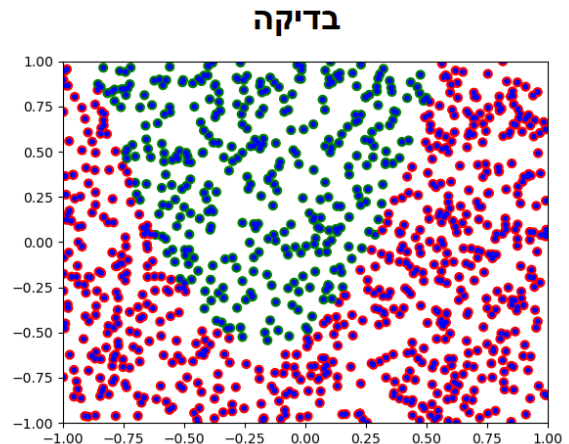
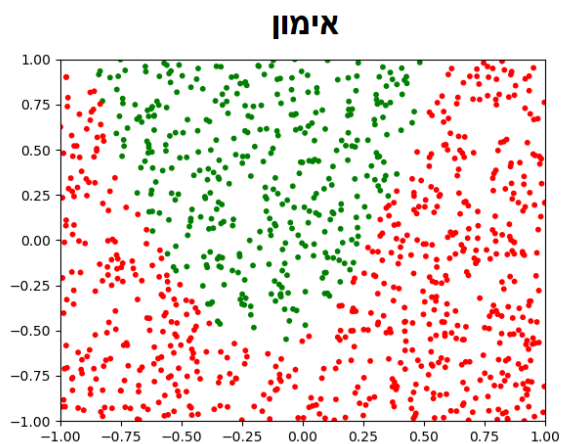
```

```

#----Train the Neural Network-----
train_points = ParabolaListOfpoint(1000,3,-1,-0.5)
train_points.drawTrainingPoints()
nn = NeuralNetwork(2,2,1)
nn.train(train_points.allXY, train_points.allLBL)
#-----TEST the Neural Network-----
test_points = ParabolaListOfpoint(1000,3,-1,-0.5)
predict_values = nn.predict(test_points.allXY)
test_points.drawMistakesPoints(predict_values)

```

נקבל את הפלט הבא:

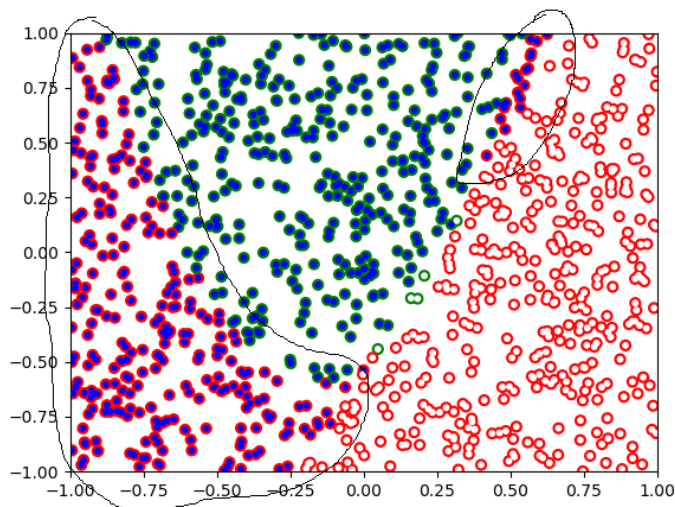


## שאלה

נסו להריץ את התוכנית מספר פעמים ובדקו האם בכל הפעמים המערכת משלימה את תהליך הלמידה באופן נכון?

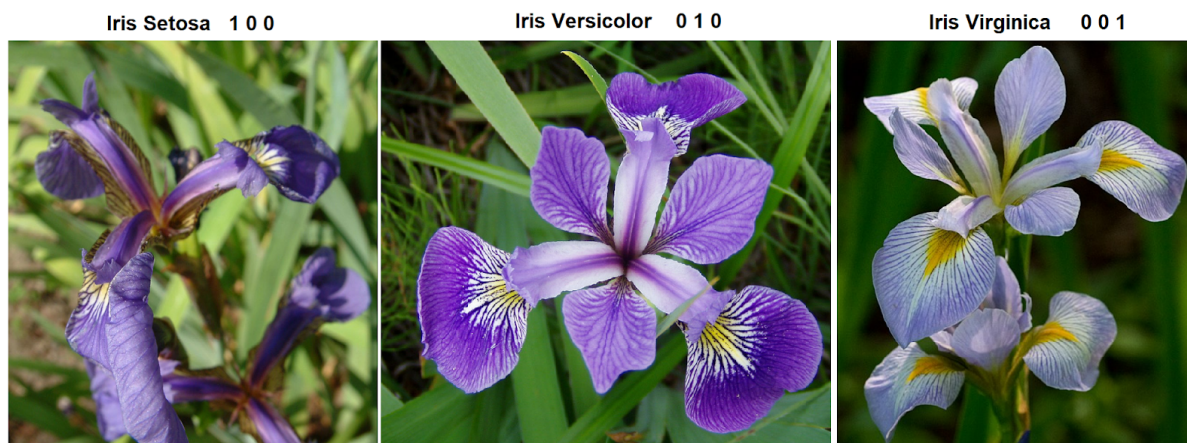
## תשובה

לא ולא, יש מצבים שעבור אותו הקלט בדיוק נקבל תשובות בסגנון הבא:



## במשימה 3 - מכונה לומד לזהות שלוש סוגי פרחים.

במטרה לבחון את המחלקה NeuralNetwork על נתוני אמת נשתמש במערך נתונים קיים הכולל מידע מתויג על 3 סוגים שונים של אירוסים. להלן תמונות של שלשות האירוסים:



מקור התמונה: [https://commons.wikimedia.org/wiki/Iris\\_lridaceae](https://commons.wikimedia.org/wiki/Iris_lridaceae) (לשימוש חוזר עם אפשרות לשינויים)

כיצד אם כן ניקח מערך של פיקסלים המייצגים תמונות של אירוסים ונכניס אותם למבוא מכונה לומדת כדי לבנות אלגוריתם לזיהוי תמונות?

נדגים את הבעיה: אם כל תמונה כוללת 500 פיקסלים על 500 פיקסלים יהיה צורך ליצור רשת נוירונים הכוללת 500\*500 נוירונים במבוא. כלומר רשת של 250000 נוירונים במבוא. כמו כן יהיה צורך ב-3 נוירונים במוצא עבור אבחון 3 סוגי האירוסים.

רשת זו גדולה על מידותיה של המחלקה שלנו. על כן ננסה לזהות מתוך התמונות מספר מאפיינים שיתנו לו את הבחנות המבדילות עבור שלושת סוגי הפרחים (דוגמה לכך היא לאפיין לפי צבע, גדול....). בדרך זו ניתן לצמצם משמעותית את מספר הנירונים במבוא המכונה ולאפשר למחלקה שלנו לנסות לפתח אלגוריתם שיבדיל בין שלושת האירוסים.

נעזר במאגר נתונים מתוקף מחקרית המסווג את שלושת הפרחים שלנו על פי 4 מאפיינים שהם:

- אורך ה- petal (אורך עלי כותרת של איריס)
- רחב ה- petal (רוחב עלי כותרת של איריס)
- אורך ה- sepal (אורך עלי הגביע של איריס)
- רחב ה- sepal (רוחב עלי הגביע של איריס)

להלן תמונה שתסביר את המאפיינים Features של הפרח



קישור למסמך תיקוף הנתונים בקישור הבא:

<http://archive.ics.uci.edu/ml/datasets/Iris>

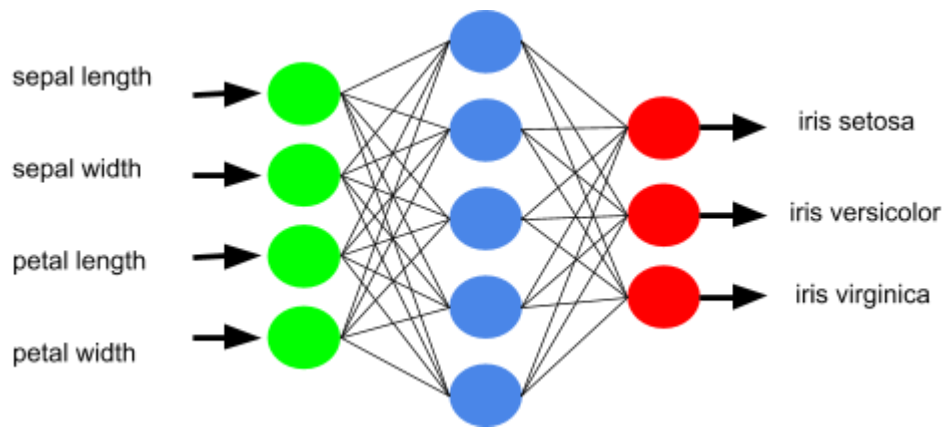
ניתן להוריד את קובץ הנתונים ישירות מהקישור הבא:

[https://www.neuraldesigner.com/files/datasets/iris\\_flowers.csv](https://www.neuraldesigner.com/files/datasets/iris_flowers.csv)

נפתח את קובץ הנתונים על ידי תוכנת Excel ונקבל את הנתונים הבאים:

	A	B	C	D	E
1	sepal_length	sepal_width	petal_length	petal_width	class
2	5.1	3.5	1.4	0.2	iris_setosa
3	4.9	3	1.4	0.2	iris_setosa
4	4.7	3.2	1.3	0.2	iris_setosa
5	4.6	3.1	1.5	0.2	iris_setosa
6	5	3.6	1.4	0.2	iris_setosa
7	5.4	3.9	1.7	0.4	iris_setosa
8	4.6	3.4	1.4	0.3	iris_setosa
9	5	3.4	1.5	0.2	iris_setosa
10	4.4	2.9	1.4	0.2	iris_setosa
11	4.9	3.1	1.5	0.1	iris_setosa
12	5.4	3.7	1.5	0.2	iris_setosa
13	4.8	3.4	1.6	0.2	iris_setosa
14	4.8	3	1.4	0.1	iris_setosa
15	4.3	3	1.1	0.1	iris_setosa
16	5.8	4	1.2	0.2	iris_setosa
17	5.7	4.4	1.5	0.4	iris_setosa
18	5.4	3.9	1.3	0.4	iris_setosa
19	5.1	3.5	1.4	0.3	iris_setosa
20	5.7	3.8	1.7	0.3	iris_setosa
97	5.7	3	4.2	1.2	iris_versicolor
98	5.7	2.9	4.2	1.3	iris_versicolor
99	6.2	2.9	4.3	1.3	iris_versicolor
100	5.1	2.5	3	1.1	iris_versicolor
101	5.7	2.8	4.1	1.3	iris_versicolor
102	6.3	3.3	6	2.5	iris_virginica
103	5.8	2.7	5.1	1.9	iris_virginica
104	7.1	3	5.9	2.1	iris_virginica
105	6.3	2.9	5.6	1.8	iris_virginica
106	6.5	3	5.8	2.2	iris_virginica
107	7.6	3	6.6	2.1	iris_virginica
108	4.9	2.5	4.5	1.7	iris_virginica
109	7.3	2.9	6.3	1.8	iris_virginica
110	6.7	2.5	5.8	1.8	iris_virginica

קיבלנו קובץ המכיל דגימות של 150 פרחים (50 מכל סוג) כאשר לכל פרח מדדו את 4 המאפיינים שקבענו כדי להבדיל בין השלושה. כמובן הקובץ מכיל עמודה חמישה הכוללת את שם הפרח שאותו מדדו. מכאן שיש לנו 150 שורות של מידע מתויג. אותם נכניס למכונה לומדת הכוללת 4 מבואות ו- 3 מוצאים.



בשלב הבא נקלוט את נתוני הקובץ לתוך מערך numpy שאנו מכירים. להלן דוגמת הקוד:

```
import numpy as np
from termcolor import colored

vir_iris_data = np.genfromtxt('iris_for_ML.csv', delimiter=',')
print(vir_iris_data)
```

נקבל את הפלט הבא:

```
[[5.1 3.5 1.4 0.2 1. ]
 [4.9 3.  1.4 0.2 1. ]
 [4.7 3.2 1.3 0.2 1. ]
 [4.6 3.1 1.5 0.2 1. ]
 [5.  3.6 1.4 0.2 1. ]
 [5.4 3.9 1.7 0.4 1. ]
 [4.6 3.4 1.4 0.3 1. ]
 [5.  3.4 1.5 0.2 1. ]
 [4.4 2.9 1.4 0.2 1. ]
 [4.9 3.1 1.5 0.1 1. ]
 [5.4 3.7 1.5 0.2 1. ]
 [4.8 3.4 1.6 0.2 1. ]
 [4.8 3.  1.4 0.1 1. ]
```

בשלב הבא יש צורך להתאים את מערך הנתונים כדי שיכנס לרשת הניורונים באופן הבא:

- נמחק מהקובץ את שורת הכותרת.
- נשנה את שם הפרח למספרים 1 עד 3 בהתאמה לשם הפרח.
- נערבב את השורות בקובץ כדי שהפרחים לא יהיה מסודרים לפי סוג.
- נחלק את המערך ל- 4 מערכים על פי הפירוט הבא:
  - a. מערך אימונים הכולל רק את הנתונים.
  - b. מערך הכולל את התגיות של מערך האימונים.
  - c. מערך בדיקה הכולל רק את הנתונים.
  - d. מערך הכולל את התגיות של מערך הבדיקה.
- נעצב את 2 מערכי התגיות כך שמקום שם הפרח נקבל 1 במוצא המייצג את שם הפרח הנכון ו-0 בשני האחרים.

```
import numpy as np
from termcolor import colored

vir_iris_data = np.genfromtxt('iris_for_ML.csv', delimiter=',')
random_iris_data = np.random.permutation(vir_iris_data)

test_data = random_iris_data[0:10,:4]
train_data = random_iris_data[10:,:4]

test_lbl = random_iris_data[0:10,4:]
train_lbl = random_iris_data[10:,4:]

tmp=[]
for i in range(len(test_lbl)):
    if test_lbl[i] == 1:
        t = np.array([1,0,0])
    elif test_lbl[i] == 2:
        t = np.array([0,1,0])
    else:
        t = np.array([0,0,1])
    tmp.append(t)
new_test_lbl=np.array(tmp)

tmp=[]
for i in range(len(train_lbl)):
    if train_lbl[i] == 1:
        t = np.array([1,0,0])
    elif train_lbl[i] == 2:
        t = np.array([0,1,0])
    else:
        t = np.array([0,0,1])
    tmp.append(t)
new_train_lbl=np.array(tmp)
```

```

print("\nrandom_iris_data: ",colored(random_iris_data, 'red'),"\n")
print("\ntest_data: ",colored(test_data, 'green'),"\n")
print("\ntrain_data: ",colored(train_data, 'blue'),"\n")
print("\ntest_lbl: ",colored(test_lbl, 'green'),"\n")
print("\nnew_test_lbl: ",colored(new_test_lbl, 'green'),"\n")
print("\ntrain_lbl: ",colored(train_lbl, 'blue'),"\n")
print("\nnew_train_lbl: ",colored(new_train_lbl, 'blue'),"\n")

```

נבחן את הנתונים שקיבלנו:

```

random_iris_data:      test_data:      train_data:      test_lbl:      new_test_lbl:      train_lbl:      new_train_lbl:
[[5.7 3.8 1.7 0.3 1. ] [[5.7 3.8 1.7 0.3] [[5.7 2.6 3.5 1. ] [[1.]      [[1 0 0]      [[2.]      [[0 1 0]
[6.  2.9 4.5 1.5 2.  ] [6.  2.9 4.5 1.5] [5.5 2.4 3.7 1.  ] [2.]      [[0 1 0]      [[2.]      [[0 1 0]
[6.9 3.1 4.9 1.5 2.  ] [6.9 3.1 4.9 1.5] [6.6 2.9 4.6 1.3] [2.]      [[0 1 0]      [[2.]      [[0 1 0]
[5.9 3.  5.1 1.8 3.  ] [5.9 3.  5.1 1.8] [7.1 3.  5.9 2.1] [3.]      [[0 0 1]      [[3.]      [[0 0 1]
[6.3 2.7 4.9 1.8 3.  ] [6.3 2.7 4.9 1.8] [5.  3.3 1.4 0.2] [3.]      [[0 0 1]      [[1.]      [[1 0 0]
[5.4 3.7 1.5 0.2 1.  ] [5.4 3.7 1.5 0.2] [7.7 3.  6.1 2.3] [1.]      [[1 0 0]      [[3.]      [[0 0 1]
[4.6 3.2 1.4 0.2 1.  ] [4.6 3.2 1.4 0.2] [4.9 2.5 4.5 1.7] [1.]      [[1 0 0]      [[3.]      [[0 0 1]
[5.1 3.8 1.9 0.4 1.  ] [5.1 3.8 1.9 0.4] [4.7 3.2 1.3 0.2] [1.]      [[1 0 0]      [[1.]      [[1 0 0]
[4.9 3.1 1.5 0.1 1.  ] [4.9 3.1 1.5 0.1] [4.9 2.4 3.3 1.  ] [1.]      [[1 0 0]      [[2.]      [[0 1 0]
[5.1 3.4 1.5 0.2 1.  ] [5.1 3.4 1.5 0.2]] [5.7 2.8 4.5 1.3] [1.]      [[1 0 0]      [[2.]      [[0 1 0]
[5.7 2.6 3.5 1.  2.  ] [7.9 3.8 6.4 2.  ] [7.9 3.8 6.4 2.  ] [3.]      [[0 0 1]
[5.5 2.4 3.7 1.  2.  ] [4.9 3.1 1.5 0.1] [4.9 3.1 1.5 0.1] [1.]      [[1 0 0]
[6.6 2.9 4.6 1.3 2.  ] [6.1 2.8 4.7 1.2] [6.1 2.8 4.7 1.2] [2.]      [[0 1 0]
[7.1 3.  5.9 2.1 3.  ] [4.8 3.4 1.9 0.2] [4.8 3.4 1.9 0.2] [1.]      [[1 0 0]
[5.  3.3 1.4 0.2 1.  ] [6.9 3.2 5.7 2.3] [6.9 3.2 5.7 2.3] [3.]      [[0 0 1]
[7.7 3.  6.1 2.3 3.  ] [7.7 3.  6.1 2.3 3.  ] [7.7 3.  6.1 2.3 3.  ] [1.]      [[1 0 0]
[4.9 2.5 4.5 1.7 3.  ] [4.9 2.5 4.5 1.7 3.  ] [4.9 2.5 4.5 1.7 3.  ] [1.]      [[1 0 0]
[4.7 3.2 1.3 0.2 1.  ] [4.7 3.2 1.3 0.2 1.  ] [4.7 3.2 1.3 0.2 1.  ] [1.]      [[1 0 0]

```

נשלב את קוד הכנת מבנה הנתונים יחד עם רשת הניורונים שלנו ונקבל את הקוד הבא:

```

import numpy as np
import matplotlib.pyplot as plt

#np.random.seed(1000)

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)

#-----Get the iris data-----
vir_iris_data = np.genfromtxt('iris_for_ML.csv', delimiter=',')
random_iris_data = np.random.permutation(vir_iris_data)
test_data = random_iris_data[0:10,:4]
train_data = random_iris_data[10:,:4]

test_lbl = random_iris_data[0:10,4:]
train_lbl = random_iris_data[10:,4:]

tmp=[]
for i in range(len(test_lbl)):
    if test_lbl[i] == 1:
        t = np.array([1,0,0])
    elif test_lbl[i] == 2:
        t = np.array([0,1,0])
    else:
        t = np.array([0,0,1])
    tmp.append(t)
new_test_lbl=np.array(tmp)

tmp=[]
for i in range(len(train_lbl)):
    if train_lbl[i] == 1:
        t = np.array([1,0,0])
    elif train_lbl[i] == 2:
        t = np.array([0,1,0])
    else:
        t = np.array([0,0,1])
    tmp.append(t)
new_train_lbl=np.array(tmp)

```

```
#-----Train the Neural Network-----
nn = NeuralNetwork(4,5,3)
nn.train(train_data, new_train_lbl,0.1,1000)
#-----TEST the Neural Network-----
predict_values = nn.predict(test_data)
print(predict_values)
print(new_test_lbl)
```

לאחר הרצה ראשונה של הקוד שכתבנו נראה שהמכונה לא לומדת מהנתונים שסיפקנו לה כיצד לסווג 10 פרחים שנמצאים במערך הבדיקה כי קיבלנו את התוצאות הבאות:

תוצאות הסיוג לאחר אימון	נתוני אמת
[0.23069832 0.37856582 0.37428857]	[1 0 0]
[0.23061625 0.3785718 0.37431743]	[0 0 1]
[0.23061844 0.3785698 0.37431308]	[0 0 1]
[0.23066896 0.37856895 0.374299 ]	[0 1 0]
[0.23061507 0.3785697 0.37431369]	[0 0 1]
[0.23061331 0.37857014 0.37431529]	[0 0 1]
[0.23061784 0.37856973 0.37431313]	[0 0 1]
[0.2307883 0.37855972 0.37426055]	[1 0 0]
[0.23062144 0.37856953 0.3743118 ]	[0 1 0]
[0.23061493 0.37856952 0.37431341]	[0 0 1]

אנו אמורים לקבל ערך גבוה הקרוב ל-1 במקום שנתוני האמת מראים 1 וערך הקרוב ל-0 במקומות שנתוני האמת מראים 0. במקרה שלנו אנו רחוקים מאוד ממצב זה. אך אם נריץ מספיק פעמים את התוכנית שלנו על אותם נתונים בדיוק, נקבל מדי פעם סידור משקלים במערכת הניורונים שיספק לנו זיהוי טוב יחסית. להלן אחד הניסויים שייצאו:

תוצאות הסיוג לאחר אימון	נתוני אמת
[0.01699056 0.24673029 0.51423597]	[0 0 1]
0.97446049 0.00312413 0.0144766 ]	[1 0 0]
[0.01699109 0.24672718 0.51423206]	[0 0 1]
[0.01699068 0.24672954 0.51423504]	[0 0 1]
[0.01699138 0.24672509 0.51422951]	[0 0 1]
[0.01699312 0.24671319 0.5142148 ]	[0 0 1]
[0.01699077 0.24672904 0.51423441]	[0 0 1]
[0.01699063 0.24672989 0.51423547]	[0 0 1]
0.97499774 0.00308402 0.01430741]	[1 0 0]
0.97514501 0.00307188 0.01425616]	[1 0 0]

ניתן לראות בברור התאמה בין נתוני האמת לבין מוצאי המערכת לאחר האימון. כמובן אנו לא במערכת למידת מכונה אופטימלית כי אז היינו צריכים לקבל תוצאות מובהקות יותר מאלו שקיבלנו בפעילות זו.

בפעילות הבאה נראה שלא כל המכונות הלומדות מבוססות על רשתות נוירונים מלאכותיים ANN. נלמד ליישם מכונה לומדת מבוססת על אלגוריתם KNN ונבדוק האם אלגוריתם זה מסוגל לסווג טוב יותר את הפרחים שלנו. כמו כן בפרק הבא נפתח מכונות לומדות המבוססות על אלגוריתמים של רשתות נוירונים מלאכותיים ANN העושות שימוש בכלי תוכנה מקצועיים שם אנו צפויים לקבל למידת מכונה ברמה הרבה יותר טובה.

## 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.