# תכנות מכונה לומדת צעד אחר צעד מהיסוד

## פעילות 9 - יישום פרספטרון בודד

מקורות:

https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

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

https://github.com/salimoha/simple_nn_1/blob/master/nn.py

https://blog.paperspace.com/part-1-generic-python-implementation-of-gradient-descent-for-nn-optimization/

https://blog.paperspace.com/part-2-generic-python-implementation-of-gradient-descent-for-nn-optimization/

https://blog.paperspace.com/part-3-generic-python-implementation-of-gradient-descent-for-nn-optimization/

https://blog.paperspace.com/part-4-generic-python-implementation-of-gradient-descent-for-nn-optimization/

Power of a Single Neuron

https://towardsdatascience.com/power-of-a-single-neuron-perceptron-c418ba445095

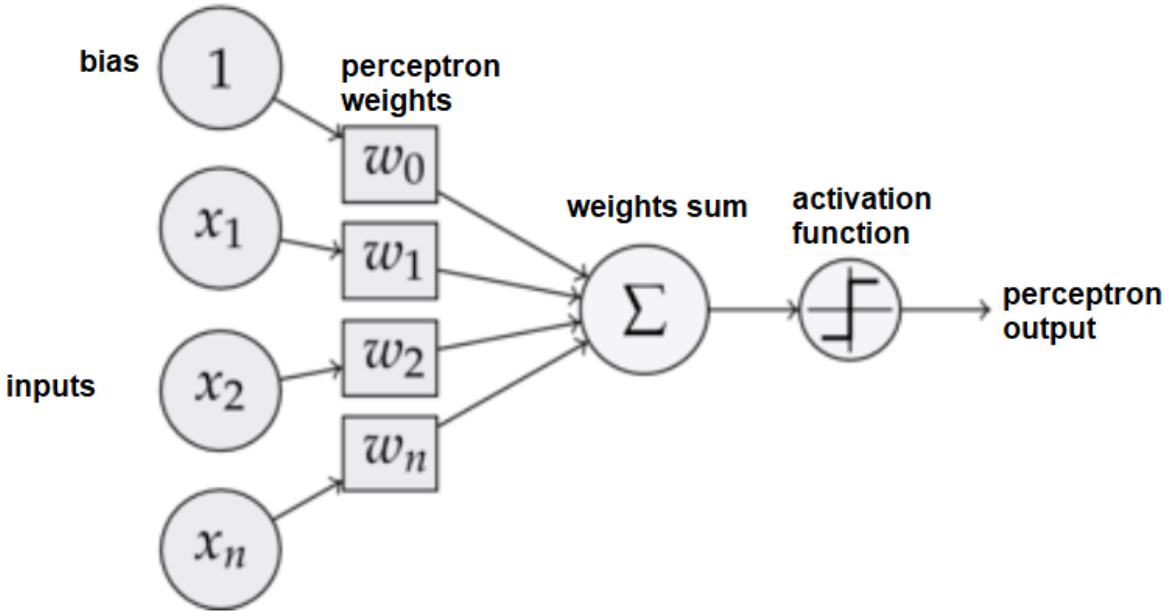
אפשר ללמוד וללמד מכונות לומדות המבוססת רשתות נוירונים בלי לראות או להבין את אופן פעולתו של נוירון בודד. שפת Python מאפשרת לממש רשתות נוירונים גדולות, מורכבות ויעילות. הצורך שלי להבין את עקרון הפעולה של נוירון בודד דרך כתיבת מעשית של נוירון הייתה מבחינתי אתגר ומטרה פדגוגית חשובה. כנראה שכתבתי את אחד הפרספטרון הפחות יעילים אלגוריתמית אך הוא עובד ומסגול ללמוד לנבא את פעולתם של מערכת צירופים כפי שהתלמידים שלנו לומדים בכיתה י'. את פריצת הדרך להבנת הנושא קיבלתי מסדרה של סרטונים בשם The coding train שהפיק פרופסור Daniel Shiffman המלמד ב- New York University להלן קישור לאתר ההרצאות:

https://thecodingtrain.com/

פרספטרון Perceptron היא היחידה הבסיסית המרכיבה רשת נוירונים. בפעילות זו נלמד לתכנת מהיסוד כלומר ללא שימוש בספריות קוד מוכנות בנושא מודל שימושי ומעשי של פרספטרון. אנו נאמן פרספטרון כמכונה לומדת שמסוגלת לנבא מערכת לוגית בעלות מספר מבואות ומוצא אחד. לדוגמה נלמד את הפרספטרון לפעול כמו שער לוגי מסוג AND. דרך התנסות זו אנו נלמד את עקרונות הפעולה של מכונות לומדות המבוססת על רשתות נוירונים. להלן תיאור טבלת המצבים של שער לוגי AND אותו נממש בלמידת מכונה.

| Input A | Input B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

תרשים המלבנים כללי של פרספטרון מתואר להלן:




ניתן לראות שפרספטרון בודד מורכב מ-2 מרכיבים עיקריים. הראשון פונקציית סכום של מכפלות בין כל אחד מהאותות במבוא כפול המשקל שלהם. המרכיב השני היא פונקציית מעבר activation function המקבלת את סכום מכפלות המבוא במשקלים ומוציאה במוצא הפרספטרון אות בודד בטווח ערכים רצוי. כמו כן ניתן לראות שהפרספטרון מקבל מבוא נוסף בשם bios הכולל ערך קבוע.

נממש בקוד את כל אחד מהמרכיבים. להלן קטע קוד בתוכנית שנפתח למימוש סכום המכפלות של אותות המבוא במשקלים.

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

ניתן לראות שימוש בפונקציה dot שהכרנו בפעילות 3 כאשר למדנו לעבוד עם מטריצות תוך שימוש בספריה המתמטית NumPy. פונקציה זו סוכמת את מכפלות אותות המבוא inputs במשקלים weights. לאחר קבלת הסכום נוסיף לסכום את ערכו של bias.

להלן קטע קוד למימוש פונקציית המעבר activation function

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

```
    activation = 0
  return activation
```

פעולה זו מממשת את פונקצית מעבר שמקבלת במבוא ערך לתוך משתנה s במידה וערכו של s גדול מ-0 פונקציית המעבר תוציא 1 אחרת היא תוציא אפס.

זה המקום לציין שפרספטרונים שונים כוללים פונקציות מעבר שונות המתאימות לצרכים שונים. בפעיליות ההמשך נלמד לעבוד עם פונקציות מעבר כדוגמת Sigmoid ו- TanH.

לפרטים נוספים ניתן לקבל בקישור הבא:

https://en.wikipedia.org/wiki/Activation_function

להלן תוכנית לבדיקת פונקציית המעבר כך שנפיק גרף המתאר את התנהגותה לאות מבוא ידוע:

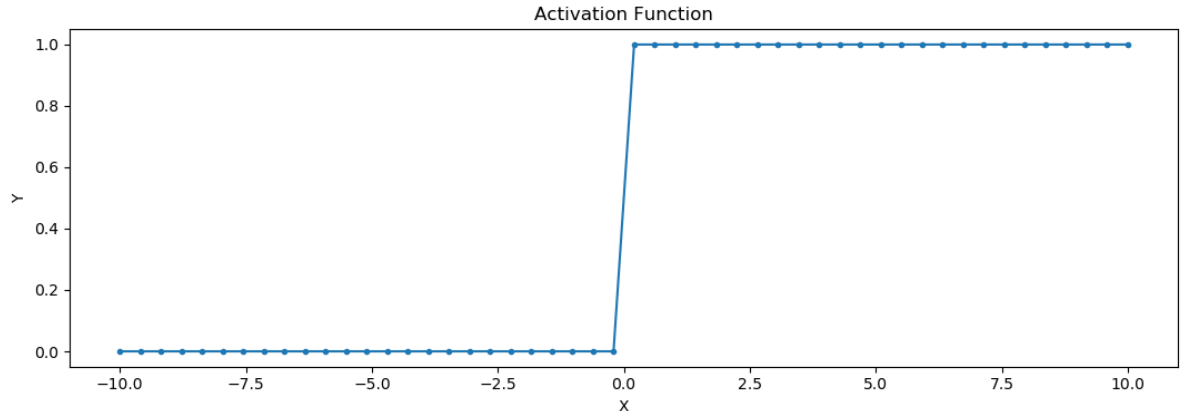
```
import numpy as np
import matplotlib.pyplot as plt


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


in_array = np.linspace(-10, 10, 50)
out_array = []
for i in range(len(in_array)):
  out_array.append(Activation(in_array[i]))
out_array = np.array(out_array)


plt.plot(in_array, out_array, marker = ".")
plt.title("Activation Function")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

נקבל את הפלט הבא:




נכתוב כעת מחלקה ב- python המממשת את הלוגיקה שלמדנו כמחלקה בשם Perceptron.

```
class Perceptron(object):
  def __init__(self, numOfInputs, epochs=100, 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
```

המחלקה Perceptron כוללת פעולה בונה ועוד 2 פעולות נוספות. פעולה בונה המקבלת מספר פרמטרים והם:

- פרמטר בשם numOfInputs המקבל את מספר המבואות שיש להקצות לפרספטרון.
- פרמטר בשם epochs המקבל את מספר הפעמים שבו הפרספטרון יעבור על כלל נתוני האימון בשלב הלמידה. במחלקה זו נגדיר כברירת מחדל את הערך ל-100.
- פרמטר בשם learningRate המייצג מספר שיקבע מה יהיה קצב השינוי המשקלים תוך כדי ביצוע חישובי הקירוב לערכים שאנו מחפשים. במחלקה זו נגדיר כברירת מחדל את הערך ל- 0.01

את 2 הפרמטרים learningRate ו- epochs פגשנו בפעילות מספר 7 כאשר דיברנו על האלגוריתם Gradient Descent שגם בו היה צורך לבצע מספר אינטראקציות כאשר בכל פעם היה צורך לשנות את הפרמטרים בקצב כלשהו.

המחלקה Perceptron כולל גם את פונקציית המעבר שלנו ופעולה נוספת שמחשבת את סכום מכפלות אותות המבוא במשקלים ואז מעבירה את התוצאה לפונקצית העבר. קבענו את שמה של הפעולה ל- predict מפני שזו בעצם אותה פעולה שבהמשך הפעילות שלנו נזמן אותה כדי לספק לה כקלט את נתוני המבוא כדי שתנבא את מוצא המערכת בהתאם למה שלמדה.

הפעולה המרכזית במחלקה Perceptron היא train הממומשת להלן:

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

לפעולה זו חשיבות מרכזית בהפיכת פרספטרון למכונה לומדת. הרעיון המרכזי של מכונה לומדת מבוסס על כך שהאלגוריתם עובר מספר פעמים על נתונים מתויגים, כלומר על סדרה של נתוני מבוא כאשר לכל נתון יש תגית המציינת מה הערך הנכון שאמור להיות במוצא הפרספטרון כאשר אותו נתון נקלט. במקרה שלנו מדובר בטבלת האמת של שער AND כאשר input A ו- input B הם הנתונים הנכנסים לפרספטרון ו- Output היא התגית הנכונה של כל נתון.

| data | | label |
|---|---|---|
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

הפעולה עוברת על כל אחד מהנתונים שבמבוא, משווה בין כל פלט כפי שהתקבל במוצא הפרספטרון ומתקנת את המשקלים בהתאם לשגיאה (ההפרש בין הערך הרצוי כלומר התווית לבין מוצא הפרספטרון).

להלן קטע הקוד הלקוח מתוך הפעולה train המממש לוגיקה זו:

```
self.weights += (labels[i] - prd) * inputs[i] * self.learningRate
self.bios += (labels[i] - prd) * self.learningRate
```

קוד זה מציג כיצד ערכם של המשקלים weights ו- bios משתנה בהתאם להפרש בין הערך הרצוי labels לבין הערך המופיע במוצא הפרספטרון prd כפול קצב שינוי המשקלים learningRate.

כדי להפעיל את המחלקה נכתוב את הקוד הבא:

```
inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
labels = np.array([0, 0, 0, 1])
perceptron = Perceptron(2)
perceptron.train(inputs, labels)

t1 = np.array([0, 0])
print(t1 , perceptron.predict(t1))
t2 = np.array([0, 1])
print(t2 , perceptron.predict(t2))
t3 = np.array([1, 0])
print(t3 , perceptron.predict(t3))
t4 = np.array([1, 1])
print(t4 , perceptron.predict(t4))
```

נגדיר 2 מערכים האחד בשם input עבור נתוני הקלט והשני labels עבור התגיות.

נגדיר עצם בשם perceptron כמופע של המחלקה Perceptron הכולל 2 מבואות.

נזמן את הפעולה train כדי לאמן את הפרספטרון על הנתונים המתוייגים. לבסוף נבדוק שאפרספטון למד על ידי כך שנזמן את הפעולה predict.

להלן הקוד המלא של תוכנת המממשת הפרספטרון מהיסוד:

```
import numpy as np


class Perceptron(object):
  def __init__(self, numOfInputs, epochs=100, 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 += (labels[i] - prd) * inputs[i] * self.learningRate
        self.bios += (labels[i] - prd) * self.learningRate

inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
labels = np.array([0, 0, 0, 1])
perceptron = Perceptron(2)
perceptron.train(inputs, labels)

t1 = np.array([0, 0])
print(t1 , perceptron.predict(t1))
t2 = np.array([0, 1])
print(t2 , perceptron.predict(t2))
t3 = np.array([1, 0])
print(t3 , perceptron.predict(t3))
t4 = np.array([1, 1])
print(t4 , perceptron.predict(t4))
```

פלט התוכנית יהיה:

| Input A | Input B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

פלט התוכנית:

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

נשנה את המידע שפרספטרון מקבל כדי ללמוד את הפרספטרון לזהות לוגיקה של שער אחר ונבדוק האם הוא מסוגל ללמוד סוגים שונים של שערים. לדוגמה לימוד שער לוגי מסוג NAND

| Inputs | | output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

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

להלן פלט התוכנית:

| Inputs | | output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

פלט

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

**תרגיל**

למדו את פרספטרון לזהות את השערים באים: XOR NOT NOR OR

בדקו האם התוכנה מסוגלת ללמוד כל אחד מהם?

**פתרון**

הפרספטרון לא מסוגל ללמוד את הלוגיקה של שער XOR להלן דוגמה לפלט תוכנית שלמדה לזהות לוגיקה של XOR:

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

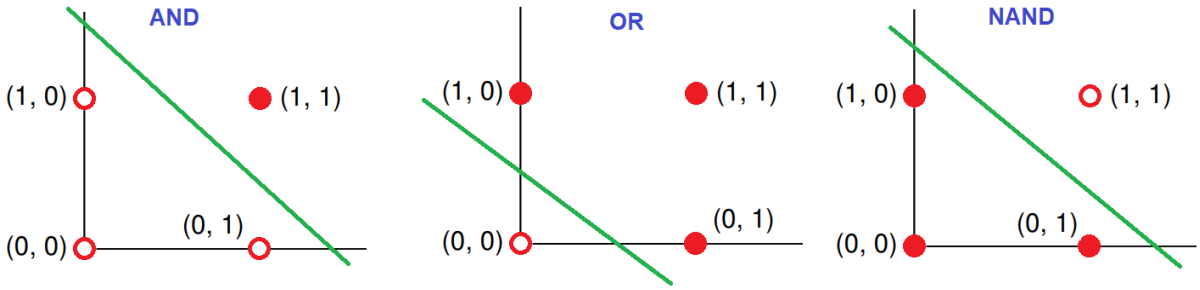
קיבלנו:

| Input | | Output |
|---|---|---|
| A | B | A xor B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

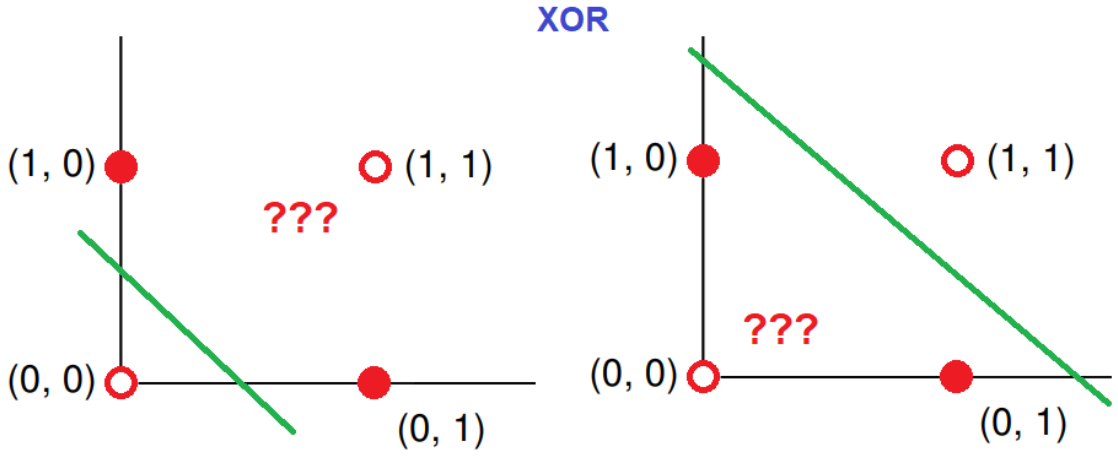
```
[0 0] 1
[0 1] 1
[1 0] 1
[1 1] 0
```

זו טעות !!!! מסתבר שפרספטרון בודד לא מסוגל ללמוד מימוש של שער לוגי מסוג XOR. למה?

פרספטרון מוגדר כמסווג לינארי כלומר מסווג שבו פונקציית הסיווג שלו מממשת משוואת קו ישר. נדגים את הבעייה על ידי האיור הבא:




באיור ניתן לראות סיווג לינארי של שערים לוגיים AND , OR ו- NAND. את כולם ניתן לסווג על ידי העברת קו לינארי אחד המפריד בין מוצא ברמה לוגית 1 לבין רמה לוגית 0. נבחן האם ניתן לעשות זאת גם על שער לוגי מסוג XOR




נראה שלא ניתן !!! בכל מצב של סיווג לינארי של שער XOR על ידי פרספטרון בודד לא נצליח לכסות את כלל מצבי המוצא.

אכן בעייה!

פתרון לכך נלמד בפעילות הבאה שבה נממש רשת של מספר פרספטרונים כדי לבנות מכונה המסוגלת ללמוד שער לוגי מסוג XOR.

**אתגר!**

ניתן לממש שער XOR על ידי פרספטרון בודד תוך כדי החלפת פונקציית המעבר. פרטים ניתן לקרוא בקישור הבא:

https://medium.com/@lucaspereira0612/solving-xor-with-a-single-perceptron-34539f395182

אתם מוזמנים לממש את הפרספטרון המתואר במאמר בקוד תוכנה...

**תרגיל**

היעזרו במחלקה Perceptron שפיתחנו מוקדם יותר בפעילות כדי ללמד פרספטרון בודד לזהות מערכת הכוללת 3 מבואות לוגיים ומוצא אחד. מצאו לפחות 2 צירופים שונים של מוצא: האחד שהרשת מסוגלת לממש ואחד שלא. נמקו את תשובתכם.

**פתרון**

להלן קוד לפתרון עבור מערכת הכוללת פרספטרון בודד מסוגל ללמוד מערכת צירופים של 3 מבואות מוצא אחד:

```
import numpy as np


class Perceptron(object):
  def __init__(self, numOfInputs, epochs=100, 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 += (labels[i] - prd) * inputs[i] * self.learningRate
        self.bios += (labels[i] - prd) * self.learningRate

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]])

labels = np.array([1, 1, 1, 1, 0, 0, 1, 1])
perceptron = Perceptron(3)
perceptron.train(inputs, labels)

t = np.array([0, 0, 0])
print(t , perceptron.predict(t))
t = np.array([0, 0, 1])
print(t , perceptron.predict(t))
t = np.array([0, 1, 0])
print(t , perceptron.predict(t))
t = np.array([0, 1, 1])
print(t , perceptron.predict(t))
```

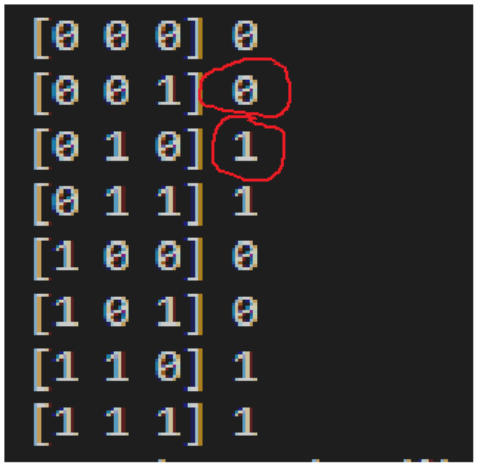
```
t = np.array([1, 0, 0])
print(t , perceptron.predict(t))
t = np.array([1, 0, 1])
print(t , perceptron.predict(t))
t = np.array([1, 1, 0])
print(t , perceptron.predict(t))
t = np.array([1, 1, 1])
print(t , perceptron.predict(t))
```

להלן קוד עבור לוגיקה שבה פרספטרון בודד לא מסוגל ללמוד:

```
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]])

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

במצב זה פלט התוכנית יהיה:



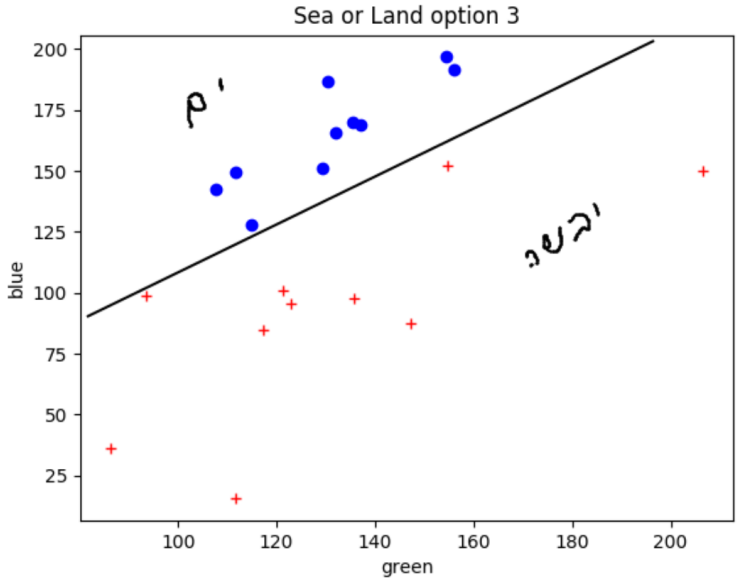

ניתן לראות שיש 2 טעויות ביחס למידע שהמערכת התבקשה ללמוד.

**משימה:** יישום מעשי של מכונה לומדת מבוססת פרספטרון לזיהוי תמונות

טוב מה כבר אפשר לעשות פרספטרון בעולם האמיתי. נבחן את העניין במשימה זו. אך תחילה חזרה קצרה.

- פרספטרון היא היחידה הבסיסית המרכיבה רשת נוירונים מלאכותית ANN.
- פרספטרון מוגדר כמסווג לינארי כלומר מסווג שבו פונקציית הסיווג מממשת משוואת קו ישר.

בפעילות 5 למדנו שניתן לסווג תמונות על פי מאפיינים (Feature) ראינו שניתן לסווג תמונות ים ותמונות יבשה על פי היחס בין צבע ירוק לצבע כחול. למדנו זאת על ידי ניסוי שעשינו על מדגם של 20 תמונות ים ויבשה ואז קיבלנו את הגרף הבא:




האם ניתן להשתמש בפרספטרון כמסווג לינארי ולבנות מכונה לומדת המבוססת על פרסטרון בודד לזהות תמונות נוף ים ויבשה?

נבחן את הקוד הבא:

```
from PIL import Image
import numpy as np


#--------start Perceptron--------------------------------
class Perceptron(object):
  def __init__(self, numOfInputs, epochs=100, 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 += (labels[i] - prd) * inputs[i] * self.learningRate
        self.bios += (labels[i] - prd) * self.learningRate
#--------end Perceptron----------------------------------

#--------start get data from images----------------------------------
sea_colors = list()
for i in range(10):
    img = Image.open("data/sea" + str(i) + ".jpg")
    img.load()
    data = np.array(img, dtype=np.uint8)
    sea_colors.append([data[:,:,color].sum() / data[:,:,color].size for color in range(3)])

land_colors = list()
for i in range(10):
    img = Image.open("data/land" + str(i) + ".jpg")
    img.load()
    data = np.array(img, dtype=np.uint8)
    land_colors.append([data[:,:,color].sum() / data[:,:,color].size for color in range(3)])
```

```
test_colors = list()
for i in range(6):
    img = Image.open("test/test" + str(i) + ".jpg")
    img.load()
    data = np.array(img, dtype=np.uint8)
    test_colors.append([data[:,:,color].sum() / data[:,:,color].size for color in range(3)])
#--------end get data from images----------------------------------

#--------Preparing the data for machine learned--------------------
sea_array = np.array(sea_colors)
land_array = np.array(land_colors)
test_array = np.array(test_colors)
x1 = sea_array[:,1]
y1 = sea_array[:,2]
x2 = land_array[:,1]
y2 = land_array[:,2]
xtest = test_array[:,1]
ytest = test_array[:,2]
x = list()
y = list()
x = np.append(x1,x2)
y = np.append(y1,y2)
data = np.stack((x, y), axis=-1)
test_data = np.stack((xtest, ytest), axis=-1)
lbl=[0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1]
#--------Preparing the data for machine learned--------------------

#--------start run machine learned---------------------------------
perceptron = Perceptron(2)
perceptron.train(data, lbl)

for i in test_data:
```

```
    if perceptron.predict(i)==1:
        print(i , "SEA Picture !!!")
    else:
        print(i , "LAND Picture !!!")
#--------end run machine learned---------------------------------
```

בקוד זה לקחנו את אותו הקוד שבתחילת פעילות זו השתמשנו בו כדי לסווג שער לוגי מסוג AND ו- OR וסיפקנו לו את מערך הנתונים שהפקנו בפעילות 5 שבה חקרנו אפשרויות לסיווג תמונות ים ויבשה.

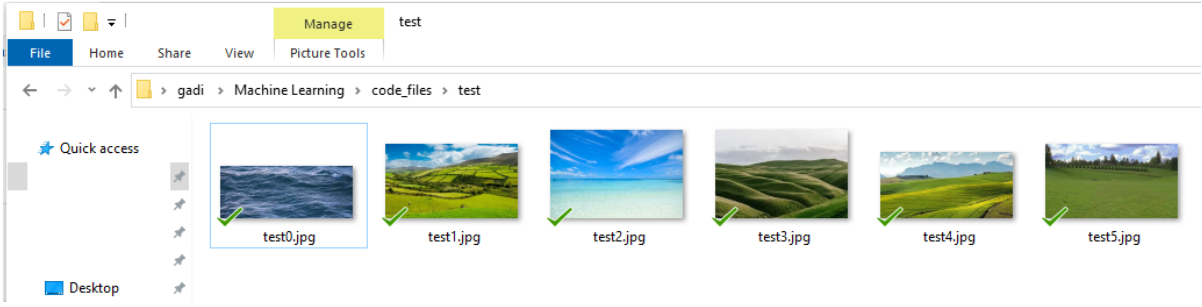
הקוד מחולק ל-4 חלקים:

- בחלק הראשון הגדרנו את המחלקה perceptron
- בחלק השני יצרנו את מערך הנתונים המבוסס על התמונות.
- בחלק השלישי ביצענו עיבוד לנתונים כדי לספק אותם במבנה מתאים ל- perceptron
- בחלק האחרון הפעלנו את תהליך הלמידה של ה- perceptron ובדקנו את איכות הלמידה על ידי מערך נתוני בדיקה.

קיבלנו את התוצאות הבאות:

```
[101.40009948 133.04162356] SEA Picture !!!
[142.22322273  70.25658182] LAND Picture !!!
[184.54570305 228.31970038] SEA Picture !!!
[133.08846498 105.62144064] LAND Picture !!!
[155.4776512   96.15760848] LAND Picture !!!
[132.79631621  86.55477991] LAND Picture !!!
```

נבדוק את התוצאות מול מערך תמנות הבדיקה:




קיבלנו התאמה מלאה בין פלט המכונה לבין סדר התמונות המסופק למכונה.
הצלחנו לממש מכונה לומדת המבוססת על פרסטרון בודד כדי לזהות תמונות נוף של ים ויבשה!

# 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.