מור-טק מרכז המורים הארצי למקצועות הטכנולוגיים
הפקולטה לחינוך למדע וטכנולוגיה, קריית הטכניון, חיפה 32000 טל: 04-8293146
E-mail: Moretech@ed.technion.ac.il
http://moretech.technion.ac.il

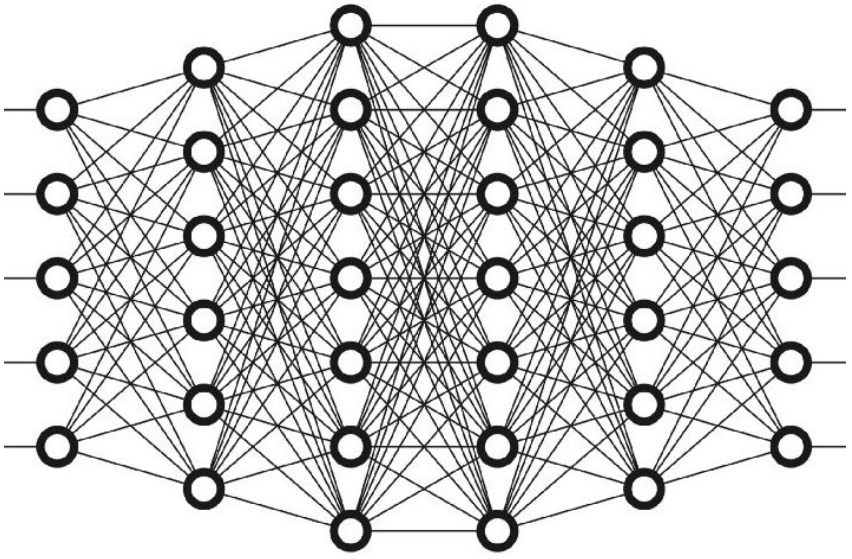
למידת מכונה

(Machine Learning)

בשפת Python

מחבר ומפתח האוגדן: מר גדי הרמן

ייעוץ מדעי ופדגוגי: מר קובי מייק



מדריך למורה - תש"פ

מור-טק מרכז המורים הארצי למקצועות הטכנולוגיים
הפקולטה לחינוך למדע וטכנולוגיה, קריית הטכניון, חיפה 32000 טל: 04-8293146
E-mail: Moretech@ed.technion.ac.il
http://moretech.technion.ac.il

מרכז מורים ארצי למורי מורטק. הפרויקט מבוצע על ידי

מוסד הטכניון עפ"י מכרז 30/8.14

הפרויקט מבוצע עבור המזכירות הפדגוגית, משרד החינוך.

האוגדן יצא לאור במימון האגף למדעים במזכירות הפדגוגית ומינהלת מל"מ המרכז הישראלי לחינוך מדעי טכנולוגי.

מור-טק מרכז המורים הארצי למקצועות הטכנולוגיים
הפקולטה לחינוך למדע וטכנולוגיה, קריית הטכניון, חיפה 32000 טל: 04-8293146
E-mail: Moretech@ed.technion.ac.il
http://moretech.technion.ac.il

תוכן עניינים

מור-טק מרכז המורים הארצי למקצועות הטכנולוגיים
הפקולטה לחינוך למדע וטכנולוגיה, קריית הטכניון, חיפה 32000 טל: 04-8293146
E-mail: Moretech@ed.technion.ac.il
http://moretech.technion.ac.il

מדריך לקורא

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 שחלק מהאלגוריתמים, כמו המחלקה המממשת פרספטרון בודד ואחרת המממשת רשת נוירונים רדודה, הם פחות יעילים אלגוריתמית מפונקציות ספריה מוכנות. אבל הם עובדים. מסגולים לבצע הליך למידה מבוסס נתונים ולבצע סיווג של מידע על סמך דוגמאות.

כשניגשתי לכתוב את המדריך למורה היה צריך להחליט האם למקד את כתיבתו על ספריות קוד מוכנות יעילות ונוחות לשימוש או להתמקד בעקרונות מתמטיים ויישומים בתוך אלגוריתמיים תכנותיים. כפי שכבר הבנתם בפתח דברי צריך למצוא איזון בין השניים. עמדתי שצריך להיות איזון בין בסיס מתמטי הקשור ללמידת מכונה, ידע אלגוריתמי מעשי ושילוב של השניים יחד החל בדוגמאות המבוססות על כתיבת קוד מהיסוד ועד לשימוש בספריית מוכנות. מכאן שמדריך זה מחולק לארבע חלקים. החלק הראשון יוצא מתוך נקודת הנחה שהקורא מכיר את יסודות התכנות בשפת Python, מכאן שהפרק עוסק בהקניית יסודות הקשורים לתחום למידת מכונה כמו שימוש בספריה המתמטית של Python, עבודה על גרפים ותמונות. החלק השני במדריך כולל לימוד אלגוריתמים בסיסיים בלמידת מוכנה כאשר המרכזי שבהם הוא Gradient Descent המוסבר הן בהיבט המתמטי שלו והן בהיבט האלגוריתמי שלו.

החלק השלישי במדריך הוא החלק המרכזי. בחלק זה אנו מפתחים מכונת למידה המבוססת על רשת נוירונים מלאכותיים מהיסוד וזאת ללא שימוש בספריות קוד ללמידת מוכנה. החלק האחרון בספר לוקח את הידע שנצבר בחלקים הקודמים ובונה מספר מכונות למידה העושות שימוש בספריות ייעודיות ללמידת מוכנה כמו scikit-learn , keras ו- tensorflow.

בגלל שמדובר במדריך למורה שעתיד להתפרסם בגרסה דיגיטלית, הרשתי לעצמי להביא את דוגמאות הקוד בשלמותם, גם אם הם מתפרשים על מספר עמודים או קטעים מהם מופיעים מספר פעמים, כל זאת במטרה לאפשר לכם המורים לעשות העתק והדבק ישירות מתוך המדריך לתוך סביבת הפיתוח שלכם.

בסוף המדריך תצאו פרק המתאר את אופן ביצוע התקנות הספרייה הנדרשות להרצת הקודים המובאים. כמו כן תמצאו בו רשימת קישורים שמצאתי אותם רלוונטיים לכל אחד מהפרקים.

במקומות שבהם קיים צורך לחדד ולהבהיר דגשים פדגוגיים הוספתי הסבר בצמוד לסימן הפותח את פסקה זו.

השראה לגישה הפדגוגית של המדריך קיבלתי מסדרה של סרטונים בשם The coding train שהפיק פרופסור Daniel Shiffman המלמד ב- New York University להלן קישור לאתר ההרצאות:

https://thecodingtrain.com

לסיכום חשוב להדגיש מדריך זה אינו ספר לימוד המיועד לתלמידים, מטרתו לשמש אתכם המורים במטרה להכיר את תחום למידת המכונה ולתת לכם כלים ורעיונות פדגוגים כיצד לתווך את הנושא לתלמידים על ידי הכנת חומרי למידה מותאמים לתלמידים שלכם.

לשירותכם מצורף קישור הכולל את דוגמאות והקבצים הנלווים לספר זה:

https://github.com/GadiHerman/machine_learning_book

לסיכום, במידה ומצאתם שגיאה או שיש לכם רעיון פדגוגי בתחום אתם מוזמנים לכתוב לי
gadi.herman@gmail.com

מבוא ללמידת מכונה

למידת מכונה (Machine Learning) היא טכנולוגיה המאפשרת לפתור בעיות תוכנה, שקשה מאוד עד כדי כך שלא ניתן היה לפתור אותם בעזרת תכנות מסורתי. טכנולוגיה זו מבוססת על היכולת של אלגוריתמים ללמוד מתוך דוגמאות ולשפר את ביצועיהם במשימות מורכבות.

להלן תרשים מלבנים המתאר עקרון פעולה של מחשב המבצע אלגוריתם תכנותי מסורתי המקבל במבוא נתונים ואלגוריתם. פלט המחשב יהיה נתונים מעובדים על פי האלגוריתם שסופק למחשב.

גישה זו מתאימה לפיתוח תוכנה בה ברורות הדרישות מהאלגוריתם. לדוגמא ניתוב הודעות אימייל לפי כתובת של משתמש. אולם כיצד יתמודד המחשב עם משימה של ניתוב דברי דואר בהן הכתובת כתובה בכתב יד? כיצד נכתוב אלגוריתם המבחין בין ספרות שונות ואותיות שונות? הגישה של למידת מכונה מתאימה לפתרון בעיות מסוג זה, בהן קשה להגדיר את הצעדים הדרושים על מנת לפתור את הבעיה, אולם קיימות לנו דוגמאות רבות של ספרות בכתב יד מהן ניתן ללמוד.

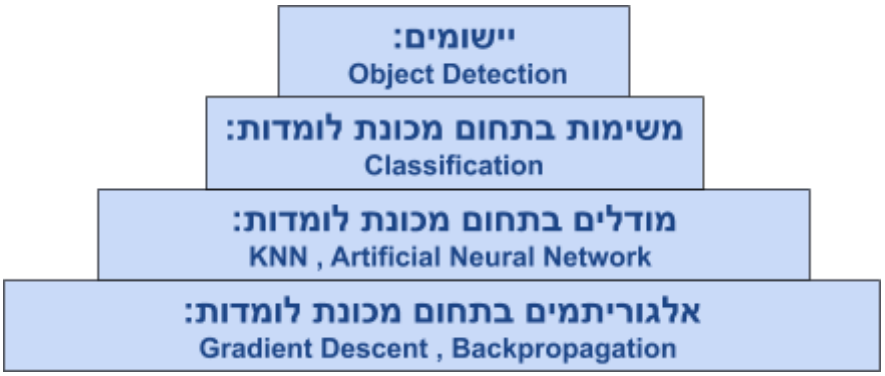
האיור הבא מתאר את עקרון הפעולה של אלגוריתם המבוסס על מכונה לומדת:

השרטוט מעלה הינו סכמטי ומטרתו להציג את ההבדל העקרוני בין מכונה לומדת לתכנות מסורתי. בהמשך הספר נלמד על התהליך המלא של אימון והפעלה של מכונה לומדת.

למראית עין האיור נראה לא הגיוני. כיצד אוסף של נתונים ומידע על הפלט שאנו רוצים לקבל מסופקים במבוא מערכת מפיקים אלגוריתם במוצא? אלגוריתם לומד בונה לעצמו מודל על ידי אימון על דוגמאות שמסופקות לו. לאחר בניית המודל, האלגוריתם משתמש במודל זה על מנת לבצע הפרדה בין הספרות השונות.

בספר זה נלמד על אלגוריתמים לומדים כמו: פרספטרון, רשתות נוירונים, KNN ועוד נעזר באלגוריתמים ומודלים במטרה לממש מכונות לומדות המסוגלות לבצע פעולות סיווג/מיון - Classification. לבסוף נאחד את הידע שצברנו כדי לכתוב יישומים לזיהוי מידע מתוך תמונות לדוגמה זיהוי מספרים ועצמים.

נדגים את הדברים באיור הבא:



הספר מחולק ל- 4 פרקים:

יסודות:

פעילות 1 - מערכים בספריית numpy

פעילות 2 - גרפים וספריית matplotlib

פעילות 3 - עבודה עם מטריצות תוך שימוש ב- NumPy

פעילות 4 - פעולות (פונקציות) ומחלקות ב- Python

פעילות 5 - ייצוג מידע ויזואלי במחשב

אלגוריתמי למידת מכונה בסיסיים:

פעילות 6 - אלגוריתם KNN ככלי לפיתוח מכונה לומדת

פעילות 7 - רגרסיה לינארית בסביבת Python

פעילות 8 - יסודות מתמטיים ל- Gradient Descent

פעילות 9 - יישום רגרסיה לינארית על ידי Gradient Descent

תכנות מכונה לומדת צעד אחר צעד מהיסוד:

פעילות 10 - יישום פרספטרון בודד

פעילות 11 - מימוש שער XOR על ידי מספר נוירונים

פעילות 12 - פיתוח רשת נוירונים מהיסוד

יישומי מכונה לומדת בשפת Python

פעילות 13 - זיהוי ספרות בכתב יד תחת scikit-learn

פעילות 14 - מימוש שער XOR על ידי מכונה לומדת תחת Keras

פעילות 15 - רשת נוירונים מבוססת Keras לסיווג תוכן בתמונות

פעילות 16 - רשת נוירונים מבוססת Keras לסיווג הודעות טקסט

פעילות 17 - סיווג פריטי לבוש בתמונות תוך שימוש keras

פעילות 18 - גיבוי ושחזור מערך המשקלים של רשת נוירונים

יסודות

פעילות 1 - מערכים וספריית numpy

NumPy היא הספרייה המתמטית של Python. ספרייה זו משתמשת במבנה נתונים ייחודי בשם ndarray העוזר לנו לבצע סדרה גדולה של פעולות מתמטיות. בפעילות זו נעשה הכרות עם מבנה נתונים זה. בהמשך המדריך כאשר נפתח רשתות נוירונים המבוססת על פרספטרונים נעשה שימוש משמעותי במבנה נתונים זה ובאוסף הפעולות שהסיפריה NumPy מספקת לנו עבורו. גם ספריות אחרות כדוגמת Matplotlib ו-scikit-learning משתמשות במבנה נתונים זה.

מערך חד מימדי

נכתוב את הקוד הבא:

```
import numpy as np


x = np.array([2, 4, 6, 8, 10])

print("type of x is:",type(x))

print("shape of x is:",x.shape)

print(x[0], x[1], x[4])

x[0] = 5

print(x)
```

נקבל את הפלט הבא:

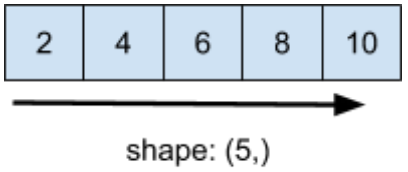
```
<class 'numpy.ndarray'>
(5,)
2 4 10
[ 5  4  6  8 10]
```

נפרש את הוראות התכנית:

הגדרת מערך חד מימדי מטיפוס numpy.ndarray בשם x הכולל 5 איברים.	x = np.array([2, 4, 6, 8, 10])

נציג על המסך את סוג טיפוס הנתונים ואת מבנה מערך הנתונים (כלומר מערך חד מימדי הכולל 5 איברים מטיפוס numpy.ndarray).	print(type(x)) print(x.shape)
נציג על המסך חלק מאיברי המערך (כאשר האיבר הראשון הוא איבר מספר 0 והאחרון הוא 4).	print(x[0], x[1], x[4])
נשנה את ערכו של האיבר הראשון במערך ל- 5 ונציג את כל איברי המערך.	x[0] = 5 print(x)

נתאר גרפית את המבנה של מערך חד מימדי:



הגדרת מערך דו מימדי

נכתוב את הקוד הבא:

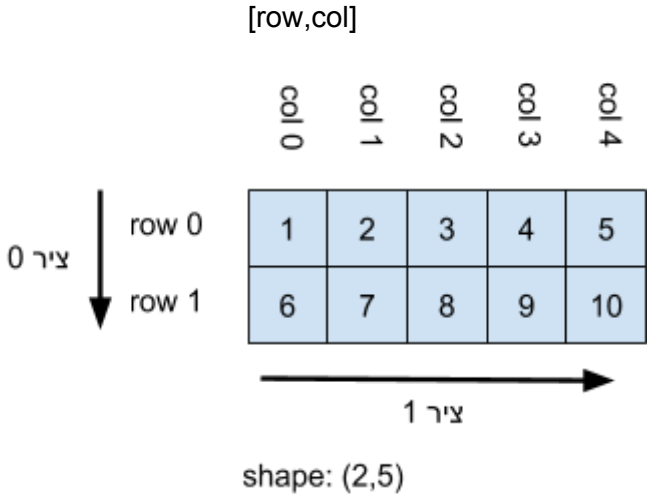
```
import numpy as np
x2 = np.array([[1, 2, 3, 4, 5],[6, 7, 8, 9, 10]])
print("print all:\n",x2)
print("type: ", type(x2))
print("shape: ", x2.shape)
print("data from row0: ",x2[0,0], x2[0,1], x2[0,4])
print("data from row1: ",x2[1,0], x2[1,1], x2[1,4])
print("all row0: ",x2[0])
print("all row1: ",x2[1])
```

```
print("Column 0:",x2[0,:])

print ("Column 1:",x2[1,:])

x2[0,0] = 500

x2[1,4] = 100

print("all new array:\n",x2)
```

נקבל את הפלט הבא:

```
print all:
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
type:  <class 'numpy.ndarray'>
shape:  (2, 5)
data from row0:  1 2 5
data from row1:  6 7 10
all row0:  [1 2 3 4 5]
all row1:  [ 6  7  8  9 10]
Column 0: [1 2 3 4 5]
Column 1: [ 6  7  8  9 10]
all new array:
 [[500   2   3   4   5]
 [  6   7   8   9 100]]
```

בדומה למערך חד מימדי גם כאן ניתן לגשת לכל אחד מאיברי המערך, לשנות אותם ולהציג אותם. אך הפעם כל איבר במערך מקבל 2 אינדקסים במבנה הבא:



ניתן לגשת לשורה במערך על ידי ההוראה:

```
print("all row0: ",x2[0])
```

ניתן לגשת לעמודה במערך על ידי ההוראה:

```
print("Column 0:",x2[0,:])
```

מערך תלת מימדי

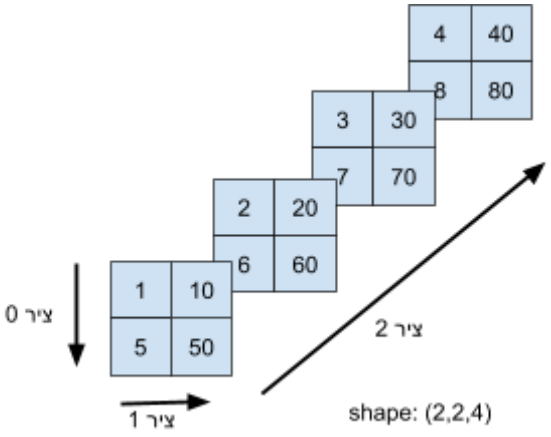
נכתוב את הקוד הבא:

```
import numpy as np
x3 = np.array([[     [1  , 2 , 3 , 4 ],
                     [5 , 6 , 7 , 8 ]],
               [     [10, 20, 30, 40],
                     [50, 60, 70, 80]]])
print("print all:\n",x3[0,0])
print("print all:\n",x3)
print("x3.type: ", type(x3))
print("x3.shape: ", x3.shape)
print("x3[0,0,0]: =",x3[0,0,0])
print("x3[1,1,3]:= ",x3[1,1,3])
print("x3[0,0]=: ",x3[0,0])
print("x3[1,1]=: ",x3[1,1])
print("x3[0,:,0]=",x3[0,:,0])
print("x3[0,:,3]=",x3[1,:,3])
x3[0,0,0] = 500
x3[1,1,3]=100
print("print all:\n",x3)
```

נקבל את הפלט הבא:

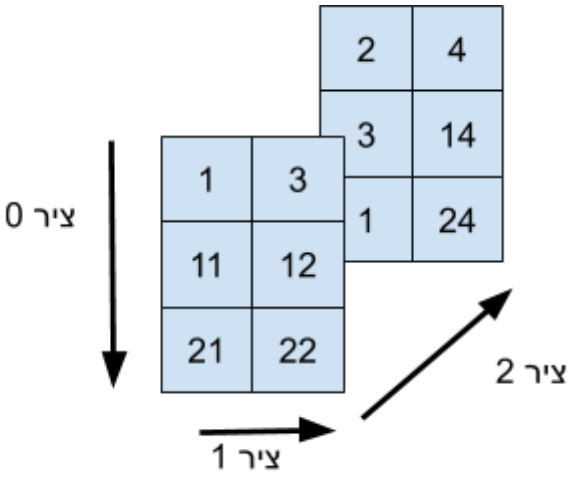
```
print all:
 [1 2 3 4]
print all:
 [[[ 1  2  3  4]
  [ 5  6  7  8]]

 [[10 20 30 40]
  [50 60 70 80]]]
x3.type:  <class 'numpy.ndarray'>
x3.shape:  (2, 2, 4)
x3[0,0,0]: = 1
x3[1,1,3]:=  80
x3[0,0]=:  [1 2 3 4]
x3[1,1]=:  [50 60 70 80]
x3[0,:,0]= [1 5]
x3[0,:,3]= [40 80]
print all:
 [[[500   2   3   4]
  [  5   6   7   8]]

 [[ 10  20  30  40]
  [ 50  60  70 100]]]
```

להלן מבנה המערך X3:



תרגיל

ממש/י את המערך הבא בקוד:



1. הציג/יו את כל אחד מאיברי המערך
2. הציג/יו את איברי המערך כאשר כל איבר מוכפל ב-2
3. הציג/יו את כל איברי המערך כאשר מוסיפים לכל איבר 10

פתרון:

```
import numpy as np


x3 = np.array([[ [1 , 2 ],[3 , 4 ] ],
               [ [11, 3],[12, 14] ],
               [ [21, 1],[22, 24] ] ])


print("\n all:\n",x3)
print("\n all*2: \n",x3*2)
print("\n all+10: \n",x3+10)
print("\n all*2: \n",x3*2)
```

אתחול מערך

פעמים רבות יעלה הצורך ליצור מערכים הכוללים ערכים כבר בשלב האתחול. לא פעם הצורך יהיה ליצור מערך נתונים הכולל מספרים אקראיים או ערכים קבועים שונים לדוגמה מערך שכולו 1 או 0. בחלק של של הפעילות נלמד לעשות זאת.

נכתוב את הקוד הבא:

```
import numpy as np

a = np.zeros((4,4))             # init 4X4 zeros matrix

print("np.zero:\n",a)

b = np.ones((1,8))              # init 1X8 ones matrix

print("np.ones:\n",b)

c = np.full((2,2), 7)           # init 2X2 matrix of value 7

print("np.full:\n",c)

d = np.eye(5)                   # init 5X5 identity matrix

print("np.eye:\n",d)

e = np.random.uniform(size=[3,5]) # init 5X3 random matrix wit uniform 0 to 1 random variable

print("uniform(size=[3,5]):\n",e)

f = np.random.uniform(-1,1,size=[2,4]) # init 2X4 random matrix wit uniform -1 to 1 random variable

print("uniform(-1,1,size=[2,4]):\n",f)
```

נקבל את הפלט הבא:

```
np.zero:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
np.ones:
 [[1. 1. 1. 1. 1. 1. 1. 1.]]
np.full:
 [[7 7]
 [7 7]]
np.eye:
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
uniform(size=[3,5]):
 [[0.76489838 0.96726512 0.26917239 0.50966359 0.40879879]
 [0.50326079 0.10500606 0.89575599 0.52825743 0.76499683]
 [0.97348599 0.4761976  0.09054215 0.96694981 0.55988833]]
uniform(-1,1,size=[2,4]):
 [[-0.07004457  0.77054727  0.29424537 -0.90894882]
 [-0.31196421  0.2049986   0.34126223  0.32111504]]
```

נפרש את הפעולות:

הפעולות zeros ו- ones מקבלות מבנה נתונים מסוג Tuple הכולל את מימדיו הרצויים של המערך. הפעולה מחזירה מבנה נתונים מטיפוס numpy.ndarray הכולל אפסים או אחדים בהתאמה.

הפעולה full מקבלת שני פרמטרים הראשון מבנה נתונים מסוג Tuple הכולל את מימדיו הרצויים של המערך והשני ערך בודד. הפעולה מחזירה מבנה נתונים מטיפוס numpy.ndarray בגודל הרצוי הכולל את ערכו של הפרמטר השני בכל האיברים שוב.

הפעולה eye מקבלת פרמטר אורך (השווה לרוחב) של מערך דו-מימדי ריבועי. הפעולה מחזירה מבנה נתונים מטיפוס numpy.ndarray בגודל הרצוי הכולל ברובו אפסים פרט לתאים שאינדקס העמודה שווה לאינדקס השורה הכוללים את הערך אחד. מטריצה זו מכונה מטריצת יחידה.

הפעולה uniform יכולה לקבל מספר שונה של פרמטרים. דוגמת הקוד מציגה שתי אפשרויות מתוך מספר רב יותר של אופציות. הדוגמה הראשונה מקבלת גודל רצוי למערך דו-מימדי ומחזירה מערך numpy.ndarray בגודל הרצוי הכולל מספרים אקראיים בין 0 ל-1. הדוגמה השנייה מקבלת את גודל מערך הרצוי ו-2 פרמטרים נוספים המייצגים טווח מספרים. הפעולה מחזירה מערך numpy.ndarray בגודל הרצוי הכולל מספרים אקראיים בטווח הנדרש.

חיתוך מערכים חד מימדיים

חיתוך מערכים ב-numpy זה לחיתוך רשימות. נדגים זאת:

```
import numpy as np

x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

print("\nprint all:\n",x)

print("start=1:stop=7:step=no ",x[1:7])

print("start=5:stop=no:step=no ",x[5:])

print("start=no:stop=5:step=no ",x[:5])

print("start=1:stop=7:step=2 ",x[1:7:2])

print("start=no:stop=no:step=-1 ",x[::-1])

print("start=-3:stop=-6:step=-1 ",x[-3:-6:-1])
```

נקבל את הפלט הבא:

```
print all:
 [ 1  2  3  4  5  6  7  8  9 10]
start=1:stop=7:step=no  [2 3 4 5 6 7]
start=5:stop=no:step=no  [ 6  7  8  9 10]
start=no:stop=5:step=no  [1 2 3 4 5]
start=1:stop=7:step=2  [2 4 6]
start=no:stop=no:step=-1  [10  9  8  7  6  5  4  3  2  1]
start=-3:stop=-6:step=-1  [8 7 6]
```

חיתוך מערכים דו מימדיים

נכתוב את הקוד הבא:

```
import numpy as np


x2 = np.array([      [1,2,3,4,5],

                     [6,7,8,9,10],
```

```
                    [11,12,13,14,15],
                    [16,17,18,19,20],
                    [21,22,23,24,25] ])

print("\nprint all:\n",x2)
print("axis 0:start=1:stop=4:step=no axis 1:start=1:stop=4:step=no\n",x2[1:4,1:4])
print("axis 0:start=2:stop=no:step=no axis 1:start=2:stop=no:step=no\n",x2[2:,2:])
print("axis 0:start=no:stop=no:step=2 axis 1:start=no:stop=no:step=2\n",x2[::2,::2])
print("axis 0:start=no:stop=no:step=-1 axis 1:start=no:stop=no:step=-1\n",x2[::-1,::-1])
print("axis 0:start=-2:stop=no:step=-1 axis 1:start=-2:stop=no:step=-1\n",x2[-2::-1,-2::-1])
```

נקבל את הפלט הבא:

```
print all:
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
axis 0:start=1:stop=4:step=no axis 1:start=1:stop=4:step=no
 [[ 7  8  9]
 [12 13 14]
 [17 18 19]]
axis 0:start=2:stop=no:step=no axis 1:start=2:stop=no:step=no
 [[13 14 15]
 [18 19 20]
 [23 24 25]]
axis 0:start=no:stop=no:step=2 axis 1:start=no:stop=no:step=2
 [[ 1  3  5]
 [11 13 15]
 [21 23 25]]
axis 0:start=no:stop=no:step=-1 axis 1:start=no:stop=no:step=-1
 [[25 24 23 22 21]
 [20 19 18 17 16]
 [15 14 13 12 11]
 [10  9  8  7  6]
 [ 5  4  3  2  1]]
axis 0:start=-2:stop=no:step=-1 axis 1:start=-2:stop=no:step=-1
 [[19 18 17 16]
 [14 13 12 11]
 [ 9  8  7  6]
 [ 4  3  2  1]]
```

שינוי צורה (shape) של מערכים

נכתוב את הקוד הבא:

```
import numpy as np
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
a = np.reshape(x,(5, 2))              # change shape of x to a 5X2 array
print("reshape(x,(5, 2)):\n",a)
b = np.reshape(x,(10, 1) )            # change shape of x to a 10X1 array
print("reshape(x,(10, 1)):\n",b)
```

```
c = np.reshape(x,(2, 5))                # change shape of x to a 2X5 array

print("reshape(x,(2, 5)):\n",c)
```

נקבל את הפלט הבא:

```
reshape(x,(5, 2)):
 [[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]]
reshape(x,(10, 1)):
 [[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]
reshape(x,(2, 5)):
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
```

הפעולה reshape מקבלת שני פרמטרים הראשון מערך numpy.ndarray והשני מבנה נתונים מסוג Tuple הכולל את מימדיו הרצויים של המערך שנקלט. הפעולה מחזירה מבנה נתונים חדש מטיפוס numpy.ndarray בגודל הרצוי הכולל את הערכים של המערך שנקלט כפרמטר הראשון.

בהוראה:

```
b = np.reshape(x,(10, 1) )
```

ניתן לראות כיצד להפוך מערך הכולל שורה לוקטור עמודה. כלומר המרה ממבנה (10,) למבנה (10,1).

עיצוב/שינוי מחדש של מבנה מערך דו מימדי

נכתוב את הקוד הבא:

```
import numpy as np
```

```
x2 = np.array([[1,2,3,4], [5,6,7,8]])
d = np.reshape(x2,8)          # change shape of x to one dimensional array
print("reshape(x2,8):\n",d)
e = np.reshape(x2,(8,1) )     # change shape of x to 8X1 array
print("reshape(x2,(8,1)):\n",e)
```

נקבל את הפלט הבא:

```
reshape(x2,8):
 [1 2 3 4 5 6 7 8]
reshape(x2,(8,1)):
 [[1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]]
```

ניתן לכתוב את ההוראה:

```
e = np.reshape(x2,(8,1) )     # change shape of x to 8X1 array
```

להוראה כללית העושה שימוש ב- shape, כך שניתן להמיר כל מבנה מערך לוקטור עמודה נדגים זאת:

```
e = np.reshape(x2,(x2.shape[0]*x2.shape[1], 1))
```

פעולות אריתמטיות ולוגיות בין מערך למספר (broadcast)

כפי שראינו באחד התרגילים בפעילות זו ניתן לבצע פעולות אריתמטיות על מערך שלם ללא הצורך בלולאה .

```
import numpy as np
x = np.array([1, 2, 3, 4, 5 ])
x1 = x+10
print("\n x:\n",x)
print("\n x1: \n",x1)
```

נקבל את הפלט הבא:

```
x:
[1 2 3 4 5]

x1:
[11 12 13 14 15]
```

יכולת זו חוסכת לנו את הצורך לכתב הקוד הכולל לולאה שעוברת על כל אחד מאיברי המערך. קוד שנכתב בגישה זו יעיל בהרבה משימוש בלולאה. נדגים זאת על ידי הקוד הבא:

```
import numpy as np
x = np.array([1, 2, 3, 4, 5 ])
x1 = x+10
print("\n x:\n",x)
print("\n x1: \n",x1)

x2 = np.empty([0])
for i in x:
    x2 = np.append(x2, [i+10])
print("\n x1: \n",x2)
```

פעולות אריתמטיות ולוגיות בין מערך למערך

נדגים פעולות מתמטיות נוספות שניתן לבצע על מערכים:

```
import numpy as np

x1 = np.array([1, 6, 9, 12, 7 ])

x2 = np.array([1, 3, 3, 6 , 2 ])
```

```
x3 = x1 + x2

x4 = x1 - x2

x5 = x1 * x2

x6 = x1 / x2

print("\n x1 + x2: \n",x3)

print("\n x1 - x2: \n",x4)

print("\n x1 * x2: \n",x5)

print("\n x1 / x2: \n",x6)
```

פלט התוכנית יהיה:

```
x1 + x2:
[ 2  9 12 18  9]

x1 - x2:
[0 3 6 6 5]

x1 * x2:
[ 1 18 27 72 14]

x1 / x2:
[1.  2.  3.  2.  3.5]
```

ניתן כמובן לבצע גם פעולות לוגיות כמודגם בקוד הבא:

```
import numpy as np

x1 = np.array([1, 2, 3, 4, 5 ])

x2 = np.array([1, 3, 3, 6, 7 ])

x3 = (x1 == x2)

x4 = (x1 < x2)

x5 = (x1 > x2)
```

```
x6 = (x1 != x2)

print("\n x1 == x2: \n",x3)

print("\n x1 > x2: \n",x4)

print("\n x1 < x2: \n",x5)

print("\n x1 != x2: \n",x6)
```

נקבל את הפלט הבא:

```
x1 == x2:
[ True False  True False False]

x1 > x2:
[False  True False  True  True]

x1 < x2:
[False False False False False]

x1 != x2:
[False  True False  True  True]
```

חיתוך מערך על ידי וקטור בוליאני

מה יהיה פלט הקוד הבא:

```
import numpy as np

x1 = np.array([1, 2, 3, 4, 5 ])

x2 = x1[x1>3]

print("\n x1>3 : \n",x2)
```

פתרון

הפתרון מדגים דרך מהירה לעבור על כל אחד מהאיברים של המערך x1 ולחפש בו את כל האיברים הגדולים מ-3.

```
x1>3 :
[4 5]
```

פעילות 2 - גרפים וספריית matplotlib

Matplotlib היא ספרייה ב-Python המייצרת גרפים דו-מימדים ותלת-מימדים במגוון פורמטים. הספרייה מספקת ממשק תכנותי הדומה ל- MATLAB. בעזרת הספרייה ניתן ליצור בפשטות וביעילות היסטוגרמות וגרפים שונים כמו עמודות ומערכות צירים X,Y. כמו כן Matplotlib מאפשרת הצגת תמונות ונתונים כקובץ תמונה.

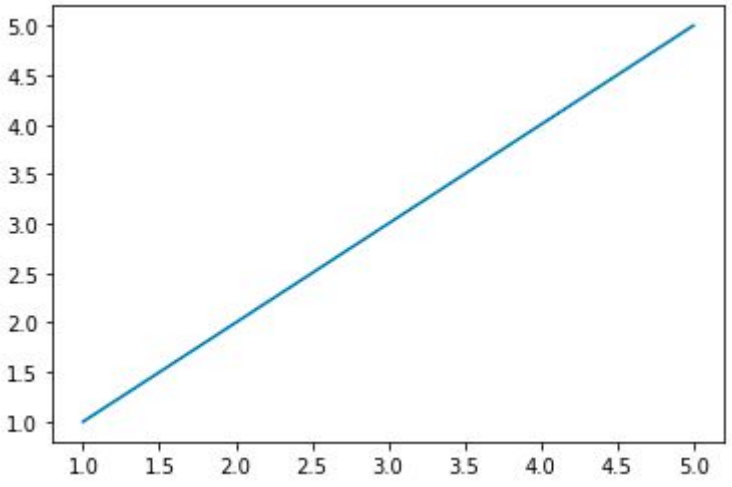
במדריך זה נעשה שימוש משמעותי ב- Matplotlib כדי להבין תהליכים המתבצעים במערכות למידת מכונה שנפתח בהמשך המדריך.

יצירת קו ישר על גבי מערכת צירים

נכתוב את הקוד הבא:

```
import matplotlib.pyplot as plt

x=[1,5]

y=[1,5]

plt.plot(x,y)

plt.show()
```

נקבל את הפלט הבא:



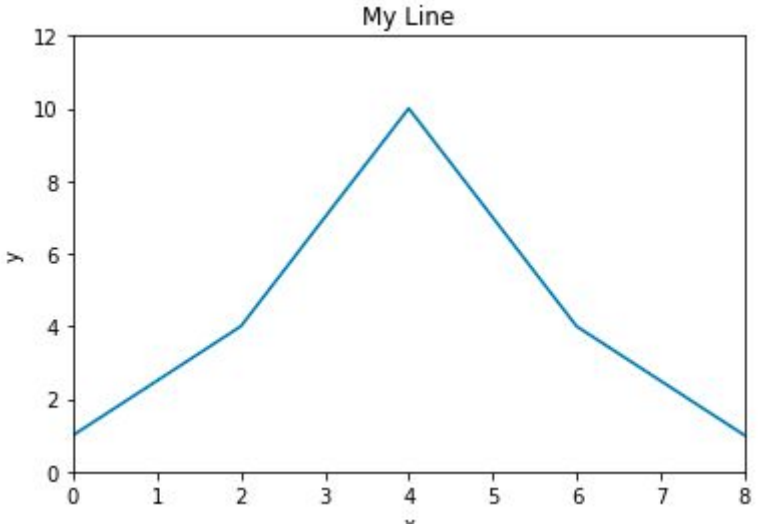
מקרא הוראות:

- הפעולה import מייבאת את הספריה על מנת שנוכל להשתמש בה.
- הפעולה plot קובעת מהם הנתונים שיש להציג על הגרף.
- הפעולה title קובעת את הכותרת של הגרף.
- הפעולה axis קובעת את טווח הערכים של מערכות הצירים לפי הפירוט הבא: [xmin, xmax, ymin, ymax]
- הפעולה xlabel ו- ylabel קובעים את הכותרות לצירים X ו- Y בהתאמה.
- הפעולה show מחוללת את הגרף.

נציג דוגמא מורכבת יותר.

```
import matplotlib.pyplot as plt

x=[1,2,3,4,5]

y=[1,3,5,7,9]

plt.plot(x,y)

plt.title('My Line')

plt.axis([0, 3, 0, 5])

plt.xlabel('x')

plt.ylabel('y')plt.show()
```

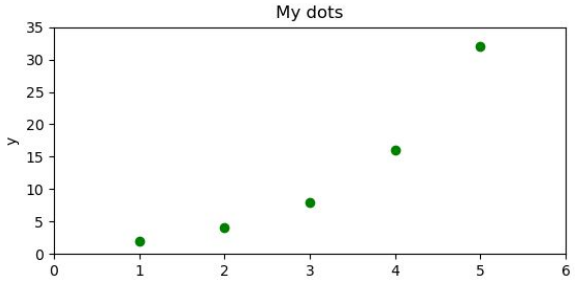
הפלט:



פיזור נקודות על גבי מערכות צירים

נכתוב את הקוד הבא:

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4, 5], [2, 4, 8, 16, 32], 'go')

plt.axis([0, 6, 0, 35])

plt.title('My dots')

plt.xlabel('x')

plt.ylabel('y')

plt.show()
```

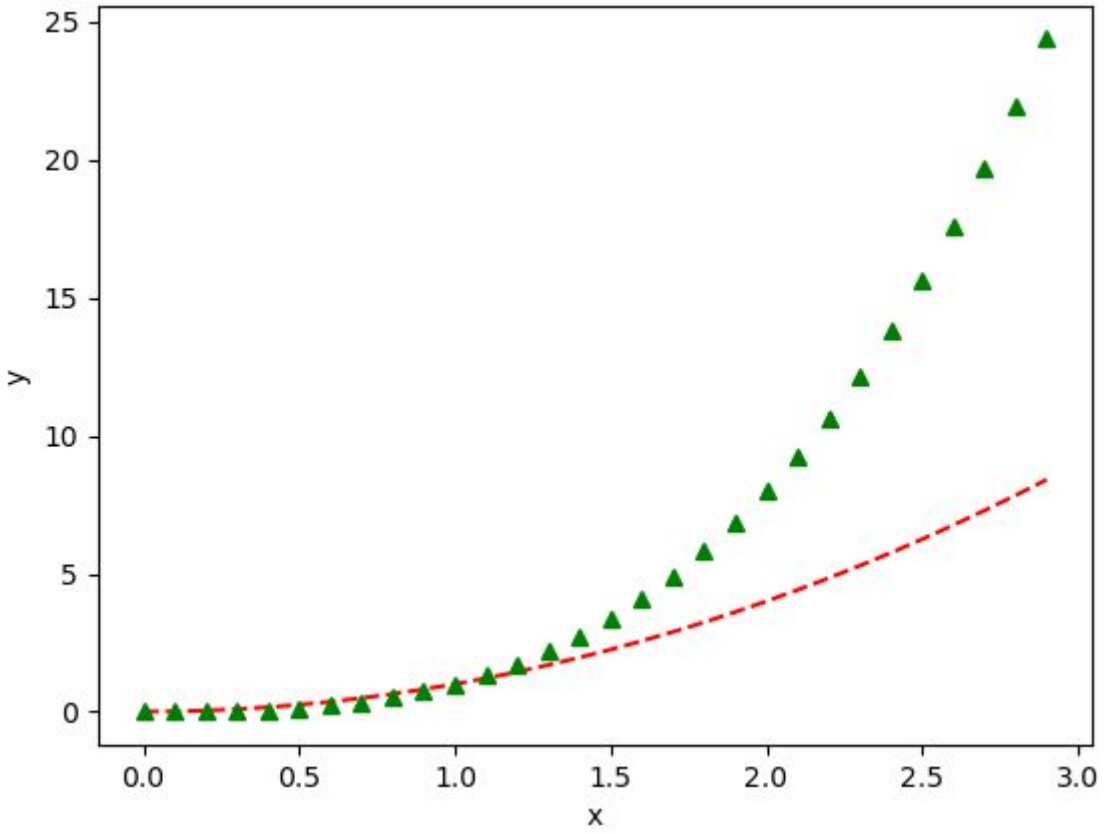


*הכיתוב go מציין שמדובר על נקודות בצורת עיגול 'o' בצבע ירוק 'g'

כמובן ניתן לשרטט מספר גרפים בצבעים שונים להלן דוגמה:

```
import matplotlib.pyplot as plt
x=[]
y1=[]
y2=[]
for i in range(0,30):
    x.append(i/10)
    y1.append(pow(i/10,2))              #pow(x,y) calculates the y power of x
    y2.append(pow(i/10,3))
plt.plot(x, y1, 'r--')
plt.plot(x, y2, 'g^')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

נקבל את הפלט הבא:



בדוגמאות הקוד הבאים בפרק זה נעשה שימוש בפעולות השייכות לספרייה המתמטית numpy של Python.

הפעולה np.arange מייצרת סדרה של מספרים מהערך הראשון שהפעולה מקבלת עד לערך האחרון בקפיצות של הערך השלישי שהפעולה מקבלת.

לדוגמה:

```
import numpy as np

t = np.arange(0.0, 3.0, 0.1)

print(t)
```

הפעולה תחזיר את הרשימה הבא:

```
[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7

 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9]
```

בדרך זו ההוראות שכתבנו עד כה ליצירת רשימה הכוללת את כל המספרים מ- 0 עד 3 בקפיצות של 0.1 כך:

```
x=[]
for i in range(0,30):
    x.append(i/10)
```

יתקצרו להוראה:

```
x = np.arange(0.0, 3.0, 0.1)
```

יתרה מכך. השימוש במבנה נתונים של numpy מאפשר לבצע פעולות מתמטיות ישירות על המבנה ללא צורך לעבור על כל אחד מהנתונים בלולאה.

לדוגמה הלולאה הבא:

```
y=[]
for i in range(0,30):
    y.append(pow(i/10,2))
```

יכולה להתחלף ב:

```
y=x**2
```

* כאשר x הוא מבנה נתונים של numpy.

שרטוט אות סינוס

משוואת אות סינוס נתונה להלן:

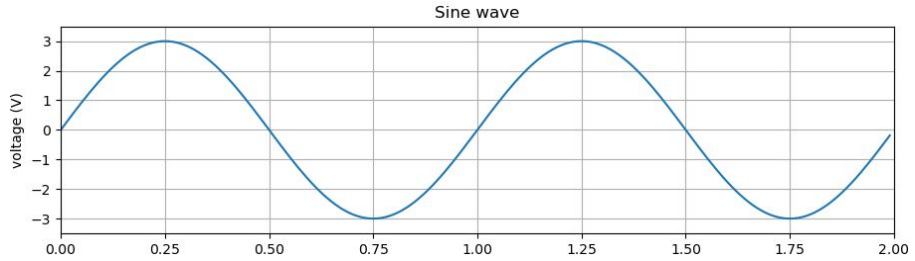
$$y(t) = A \cdot sin\,(2\pi f t)$$

נכתוב את הקוד המשתמש בנוסחה הנ"ל כדי לשרטט גרף של אות סינוס:

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

```
t = np.arange(0.0, 2.0, 0.01)
s = 3*np.sin(2 * np.pi * t)
plt.plot(t, s)
plt.grid()                          # display grid lines
plt.title("Sine wave")
plt.xlabel("time (s)")
plt.ylabel("sine wave")
plt.show()
```

נקבל את הפלט הבא:



תרגיל 2.1:

שרטט/י את האותות הבאים:

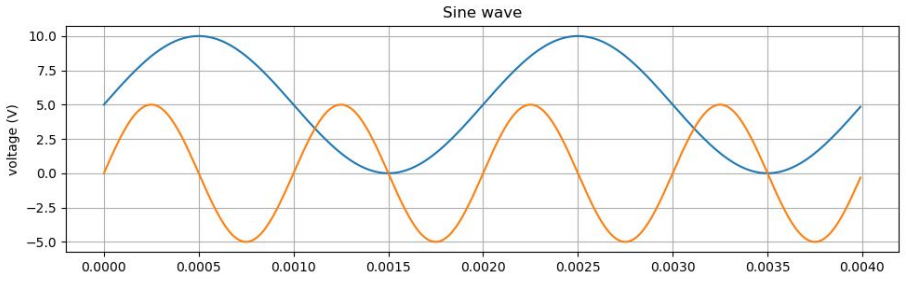
1. $y(t) = 5 + 5 \cdot sin\,(2\pi 500t)$
2. $y(t) = 5 \cdot sin\,(2\pi 1000t)$

פתרון:

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

t = np.arange(0.0, 0.004, 0.00001)
```

```
y1 = 5 + 5*np.sin(2 * np.pi * 500* t)

y2 = 5*np.sin(2 * np.pi * 1000* t)

plt.plot(t, y1)

plt.plot(t, y2)

plt.grid()

plt.title("Sine wave")

plt.xlabel("t")

plt.ylabel("y(t)")

plt.show()
```

נקבל את הפלט הבא:



תרגיל 2.2:

שרטט/י את האותות הבאים:

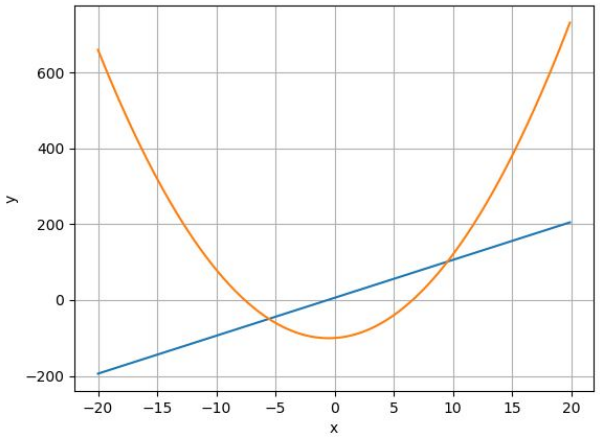
1. $y = 10x + 6$
2. $y = 2x^2 + 2x - 100$

פתרון:

```
import numpy as np

import matplotlib.pyplot as plt


x = np.arange(-20, 20, 0.1)
```

```
y1 = 10*x + 6

y2 = 2*x**2+2*x-100

plt.plot(x, y1)

plt.plot(x, y2)

plt.grid()

plt.xlabel("x")

plt.ylabel("y")

plt.show()
```

נקבל את הפלט הבא:



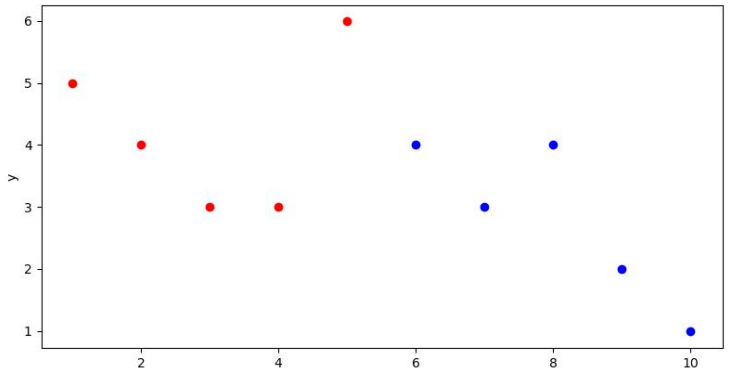
פיזור נקודות בצבעים שונים על גבי מערכות צירים דו-מימדית

למדנו עד כה את הפעולה plot המשרטטת קו מחובר בין נקודות. נלמד כעת את הפעולה scatter המציירת פיזור של נקודות בלי לחבר אותם בקווים. נכתוב את הקוד הבא:

```
import matplotlib.pyplot as plt

x1 = [1,2,3,4,5]

y1 = [5,4,3,3,6]

x2 = [6,7,8,9,10]

y2 = [4,3,4,2,1]
```

```
plt.scatter(x1, y1, c='red')

plt.scatter(x2, y2, c='blue')

plt.show()
```

נקבל את הפלט הבא:

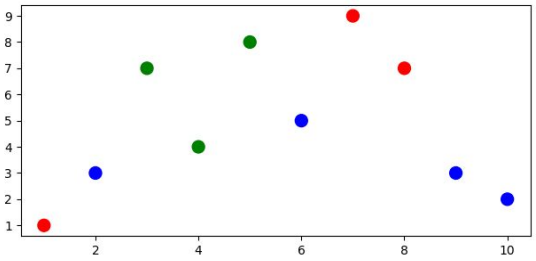


ניתן לשפר את הקוד הקודם בכך שנגדיר מערך ייעודי לצבעים של על אחד מהנקודות. כלומר מערך צבעים שממנו כל נקודה תקבל את הצבע שלה.

נדגים זאת:

```
import matplotlib.pyplot as plt

import numpy as np

a = np.array([      [1,2,3,4,5,6,7,8,9,10],

                    [1,3,7,4,8,5,9,7,3,2]          ])

categories = np.array([0,2,1,1,1,2,0,0,2,2])

colormap = np.array(['r', 'g', 'b'])

plt.scatter(a[0], a[1], s=100, c=colormap[categories])

plt.show()
```

נקבל את הפלט הבא:



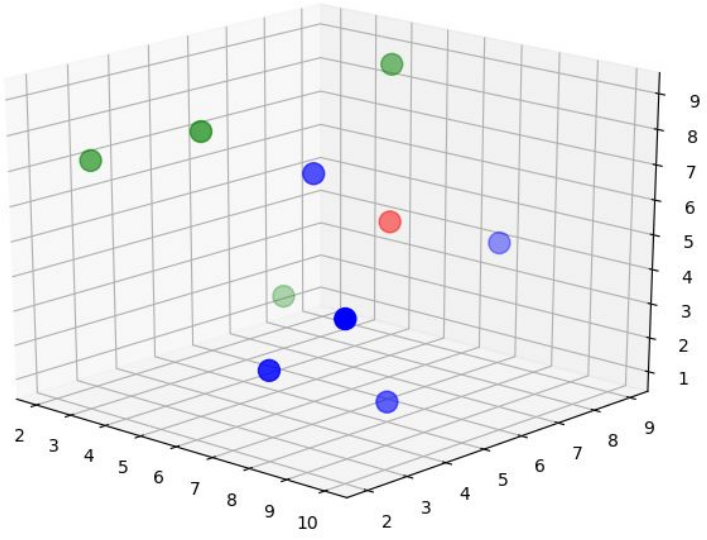
פיזור נקודות בצבעים שונים על גבי מערכות צירים תלת מימדית

נכתוב את הקוד הבא:

```
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

import numpy as np


data = np.array( [  [ 6.0 , 7.0, 5.0],

                    [ 2.0 , 3.0, 7.0],

                    [ 3.0 , 7.0, 2.0],

                    [ 4.0 , 4.0, 8.0],

                    [ 5.0 , 8.0, 9.0],

                    [ 6.0 , 5.0, 7.0],

                    [ 7.0 , 9.0, 4.0],

                    [ 8.0 , 5.0, 1.0],

                    [ 8.0 , 2.0, 3.0],

                    [10.0 , 2.0, 5.0]     ])


categories = np.array([0,1,1,1,1,2,2,2,2,2])

colormap = np.array(['r', 'g', 'b'])
```

```
fig = plt.figure()
ax = Axes3D(fig)

ax.scatter(data[:,0], data[:,1],data[:,2], s=150, c=colormap[categories])
plt.show()
```

נקבל את הפלט הבא:



פעילות 3 - מטריצות ב- NumPy

בפעילות זה נתרגל כתיבת קוד בשפת python לביצוע פעולות מתמטיות על מטריצות מסוג NumPy Arrays

מטריצה היא מערך דו מימדי של ערכים. מטריצה מאפשרת לנו לרכז בצורה יעלה מספר רב של ערכים. במערכות מכונה לומדת משתמשים במטריצות כדי לשמור את ערכי אותות המבוא והמוצא של רשתות נוירונים כמו כן את מערכי המשקלים של פרספטרונים המרכיבים את אבני הבסיס של כל רשת נוירונים.

בפעילות זו נלמד כיצד לבצע פעולות מתמטיות כגון כפל וחיבור על מטריצות במבנה numpy array

פעולות על מטריצות ריבועיות

מטריצה ריבועית היא מטריצה שמספר העמודות שלה שווה למספר השורות.

נכתוב את הקוד הבא:

```
import numpy as np

a = np.array([[1, 2], [3, 4]])

b = np.array([[5, 6], [7, 8]])

x = a + b

print("a + b:\n",x)

x=x+2

print("x+2:\n",x)

x=x/2

print("x/2:\n",x)

x=a*b

print("a*b:\n",x)
```

נקבל את הפלט הבא:

```
a + b:
 [[ 6  8]
 [10 12]]
x+2:
 [[ 8 10]
 [12 14]]
x/2:
 [[4. 5.]
 [6. 7.]]
a*b:
 [[ 5 12]
 [21 32]]
```

כפל מטריצות לא ריבועיות

ניתן לבצע גם כפל של מטריצות בגדלים שונים (כלומר מטריצות לא ריבועיות), בתנאי שמספר העמודות במטריצה הראשונה שווה למספר השורות במטריצה השנייה. במקרה זה נקבל מטריצה חדשה שמספר השורות בה זהה למספר השורות של המטריצה הראשונה ומספר העמודות שווה למספר העמודות של המטריצה השנייה.

פעולה זו שונה מכפל של איבר באיבר כמו שראינו בכפל מטריצות זהות בגודלן. הכפלה זו מתבצעת על ידי הפעולה dot והיא שימושית מאוד. נדגים זאת:

```
import numpy as np

a = np.array([[1, 2, 1], [0, 1, 0]])

b = np.array([[2, 5], [6, 7], [1, 8]])

c = a.dot(b)

print("a:\n",a)

print("b:\n",b)

print("a.dot(b):\n",c)
```

נקבל את הפלט הבא:

```
a:
 [[1 2 1]
 [0 1 0]]
b:
 [[2 5]
 [6 7]
 [1 8]]
a.dot(b):
 [[15 27]
 [ 6  7]]
```

נבדוק את התוצאה:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 5 \\ 6 & 7 \\ 1 & 8 \end{bmatrix} = \begin{bmatrix} 15 & 27 \\ 6 & 7 \end{bmatrix}$$

כמובן שלא ניתן להכפיל כל מטריצה אחת בשנייה. עזרה בנושא ניתן לקבל באתר הבא:

http://matrixmultiplication.xyz/

שחלוף (transpose) מטריצות

שחלוף Transpose מטריצות היא פעולה מתמטית המחליפה בין השורות והעמודות של מטריצה נתונה.

נכתוב את הקוד הבא:

```
import numpy as np

x2 = np.array([[1, 2, 3, 4, 5],[6, 7, 8, 9, 10]])

a = x2.transpose()                    # calculate the transpose matrix

b = x2.T                              # same as transpose()

print("x2:\n",x2)

print("x2.transpose():\n",a)

print("x2.T:\n",b)
```

נקבל את הפלט הבא:

```
x2:
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
x2.transpose():
 [[ 1  6]
 [ 2  7]
 [ 3  8]
 [ 4  9]
 [ 5 10]]
x2.T:
 [[ 1  6]
 [ 2  7]
 [ 3  8]
 [ 4  9]
 [ 5 10]]
```

כפי שניתן לראות אין הבדל בין שימוש בפעולה transpose לבין שימוש ב- T כדי לבצע היפוך מטריצות

שימו לב, לא ניתן לבצע היפוך על מטריצה חד מימדית. אבל בהחלט ניתן לבצע היפוך על מטריצה דו מימדית בגודל של 1 על X (גודל כלשהו)

נסביר את ההבדל על ידי הדוגמה הבא:

```
import numpy as np

x = np.array([2, 4, 6, 8, 10])

x2 = np.array([[2, 4, 6, 8, 10]])

a = x.T

b = x2.T
```

```
print("x.shape=",x.shape)

print("x2.shape=",x2.shape)

print("x:\n",x)

print("x2:\n",x2)

print("x.T:\n",a)

print("x2.T:\n",b)
```

נקבל את הפלט הבא:

```
x.shape= (5,)
x2.shape= (1, 5)
x:
 [ 2  4  6  8 10]
x2:
 [[ 2  4  6  8 10]]
x.T:
 [ 2  4  6  8 10]
x2.T:
 [[ 2]
 [ 4]
 [ 6]
 [ 8]
 [10]]
```

כדי להבין את העניין נתבונן בצורות המטריצות x ו- x2

מטריצה x היא חד מימדית כלומר x היא וקטור:

x.shape= (5,)

מטריצה x2 היא דו מימדית כלומר x2 היא מטריצה:

x2.shape= (1, 5)

שימוש לב להגדרות השונות וקטור למטריצה:

```
x = np.array([2, 4, 6, 8, 10])
x2 = np.array([[2, 4, 6, 8, 10]])
```

פעילות 4 - פעולות (פונקציות) ומחלקות ב- Python

פעולות

פעולות (פונקציות) מאפשרת לנו לארוז קטע קוד המבצע משימה מוגדרת תחת שם, הצורך בפעולות נובע בעיקר מהסיבות הבאות:

- יצירת פעולה המאפשרת לחלק את התוכנית למספר מקטעים בעלי היגיון פנימי.
- ניתן לזמן פעולות ממקומות שונים בתוכנית.
- מימוש פעולות מאפשר לפשט אלגוריתמים מורכבים ולארוז אותם בנפרד.

תכונות פעולה:

- פעולה יכולה לקבל פרמטרים כאשר מזמנים אותה.
- פעולה יכולה גם להחזיר פרמטרים בסיום ההרצה שלה.

הגדרת פעולה בשפת python מתבצעת על ידי ההוראה def נדגים זאת:

```
def SignCheck(x):
    if x > 0:
        print(x,"is positive")
    elif x < 0:
        print(x,"is negative")
    else:
        print(x,"is zero")


SignCheck(-10)


for i in [-2,0,10,-10,0,3]:
    SignCheck(i)
```

נקבל את הפלט הבא:

```
-10 is negative
-2 is negative
0 is zero
10 is positive
-10 is negative
0 is zero
3 is positive
```

הדגמנו פעולה המקבלת כקלט מספר ומציגה על המסך האם הוא חיובי, שלילי או שווה לאפס.

נדגים אותה פעולה אך הפעם היא תחזיר ערך תוך שימוש במשפט return.

```
def SignCheck(x):
    if x > 0:
        return "positive"
    elif x < 0:
        return "negative"
    else:
        return "zero"

for i in [-2,0,10,-10,0,3]:
    print(i, "is", SignCheck(i))
```

מוזמנים לנסות להריץ את הקוד בעצמכם

פעולות ב- python יכולות להחזיר יותר מערך אחד. נדגים זאת:

```
def NewDiv(x,y):
    a=x//y
    b=x%y
    return a , b
```

```
n1,n2 = NewDiv(14,4)

print(n1,n2)


for x in [[4, 2],[7,3]]:

   print(NewDiv(x[0],x[1]))
```

פלט התוכנית יהיה:

```
3 2
(2, 0)
(2, 1)
```

שימוש באופרטור * כדי לארוז ולפרק פרמטרים (Packing and Unpacking)

נדגים את הקוד הבא:

```
def fun(a, b, c):
    print(a, b, c)


my_list = [1, 2, 3]
fun(my_list)
```

נקבל הודעת שגיאה:

```
TypeError: fun() missing 2 required positional arguments: 'b' and 'c'
```

הפעולה מבקשת לקבל 3 פרמטרים ואנו מספקים לה רק אחד. (נכון שמדובר ברשימה הכולל 3 איברים אבל זה לא משנה לפעולה)

כדי לפתור את הבעיה. ניתן להשתמש באופרטור * כדי לפזר (Unpacking) את הרשימה לאיברים שלה. נדגים זאת:

```
def fun(a, b, c):
    print(a, b, c)
```

```
my_list = [1, 2, 3]
fun(*my_list)
```

ניתן להשתמש באופרטור * כדי לכתוב פעולות שמקבלות כמות לא ידוע של פרמטרים באופן הבא:

```
def Sum(*args):
    sum = 0
    for i in range(0, len(args)):
        sum = sum + args[i]
    return sum

print(Sum(10,2))
print(Sum(1,2,3,4,5))
print(Sum(10,20,30))
```

מחלקות

נדגים מחלקה בסיסית המייצגת נקודה המורכבת מ-2 ערכים (x ו- y)

```
class MyPoint(object):
   def __init__(self, x,y):          #constructor
      self.x = x                     # self is a pointer to the object
      self.y = y
   def addX(self, x):
      self.x += x
   def addY(self, y):
      self.y += y
```

המחלקה כוללת פעולה בונה המקבלת 2 ערכים (x ו- y). ערכים אלה מתנהגים כתכונות של הפעולה כלומר ניתן לגשת אליהם לאחר יצירת העצם. כמו כן המחלקה כולל 2 פעולות נוספות הראשונה addX והשנייה

addY. ניתן לראות לפי התחביר self שכל הפעולות במחלקה כולל הפעולה הבונה (constructor) מקבלת הפנייה לעצמה. או במילים אחרות מצביע לעצמה.

להלן קוד מלא של תוכנית הכולל את המחלקה MyPoint ואת השימוש בה:

```
class MyPoint(object):
    def __init__(self, x,y):
        self.x = x
        self.y = y
    def addX(self, x):
        self.x += x
    def addY(self, y):
        self.y += y


p=MyPoint(0,0)
p.x=100
p.y=100
p.addX(-10)
p.addY(10)
print("obj p:  x=",p.x,"y=",p.y)


Points_list = []
for i in range(10):
    p=MyPoint(i,i)
    p.addX(-10)
    p.addY(10)
    Points_list.append(p)
```

```
for obj in Points_list:
    print("x=",obj.x,"y=",obj.y)
```

פלט התוכנית יהיה:

```
obj p:  x= 90 y= 110
x= -10 y= 10
x= -9 y= 11
x= -8 y= 12
x= -7 y= 13
x= -6 y= 14
x= -5 y= 15
x= -4 y= 16
x= -3 y= 17
x= -2 y= 18
x= -1 y= 19
```

ניתן להגדיר בפעולה הבונה פרמטרים אופציונאליים, בדוגמה הבאה הגדרנו את הערך 0 כברירת מחדל עבור x ו- y. באופן זה ניתן ליצור מנגנון הדומה לפעולות בונות מרובות. נדגים זאת:

```
class MyPoint(object):
    def __init__(self, x=0,y=0):
        self.x = x
        self.y = y
    def addX(self, x):
        self.x += x
    def addY(self, y):
        self.y += y


p1=MyPoint(5,5)
p1.x=100
p1.y=100
p1.addX(-10)
```

```
p1.addY(10)

print("obj p1:  x=",p1.x,"y=",p1.y)


p2=MyPoint()

p2.addX(-10)

p2.addY(10)

print("obj p2:  x=",p2.x,"y=",p2.y)


Points_list = []

for i in range(5):

    p=MyPoint(i,i)

    p.addX(-10)

    p.addY(10)

    p.x=100

    Points_list.append(p)

for obj in Points_list:

    print("x=",obj.x,"y=",obj.y)
```

פלט התוכנית יהיה:

```
obj p1:  x= 90 y= 110
obj p2:  x= -10 y= 10
x= 100 y= 10
x= 100 y= 11
x= 100 y= 12
x= 100 y= 13
x= 100 y= 14
```

נשנה את המחלקה point כך שיקבעו ערכים אקראיים עבור x ו- y. כמו כן נעצב את פלט לנתוני המחלקה.

```
import numpy as np
class point(object):
    def __init__(self):
        self.x = np.random.uniform(-1,1)
        self.y = np.random.uniform(-1,1)
        if self.x > self.y:
            self.label = 1
        else:
            self.label = -1
    def __repr__(self):          # implements to_string function
        return "x="+str(self.x)+" y="+str(self.y)+" label="+str(self.label)+"\n"

p= point()
print(p)

Points_list = []
for i in range(10):
    Points_list.append(point())
print(Points_list)
```

התכונה label תשמש אותנו בהמשך לסיווג הנקודות ונבין לעומק את משמעותה.

נקבל את הפלט הבא:

```
x=0.7907591440637343 y=-0.058959954890721145 label=1

[x=0.6876653909738102 y=0.08977168407574498 label=1
, x=0.3428198983546864 y=-0.5145277973129032 label=1
, x=0.10600195016585712 y=-0.14154582078282085 label=1
, x=-0.5499731736537745 y=-0.9320351544322274 label=1
, x=-0.9717897738567731 y=0.7414112851627186 label=-1
, x=0.9078033924939835 y=0.16622836537757957 label=1
, x=-0.43462113690052373 y=-0.14375660869085238 label=-1
, x=0.6873452973067942 y=0.9829363640425661 label=-1
, x=-0.4758858269531585 y=0.7870852594065578 label=-1
, x=0.3853977260543342 y=-0.3552255667708084 label=1
]
```

פעילות 5 - ייצוג מידע ויזואלי במחשב

בפעילות זה נתרגל כתיבת קוד בשפת python לעבודה עם תמונות, נלמד כיצד פותחים קובץ תמונה קיים, כיצד בנוי הקובץ וכיצד לעבד את הנתונים בתוך הקובץ. בפעילות זו נעשה שימוש ב - PIL שהם ראשי התיבות של Python Imaging Library שהיא ספרייה ב- Python המספקת יכולות עיבוד תמונות.

פתיחת קובץ תמונה

נכנס לאתר שיתוף התמונות https://pixabay.com ונוריד למחשב תמונה. שימו לב שמדובר באתר המאפשר להוריד תמונה ממאגר תמונות גדול שנכון להיום התמונות שבו ניתנות בחינם ללא זכויות יוצרים. לדוגמה הורדתי למחשב את התמונה הבא תחת השם train.jpg

Image by David Mark from Pixabay

כדי לפתוח את התמונה תוך שימוש בספריה PIL נכתוב את הקוד הבא:

```
from PIL import Image

img = Image.open("data/train.jpg")

width, height = img.size

print(width, height)

img.show()
```

הפעולה:

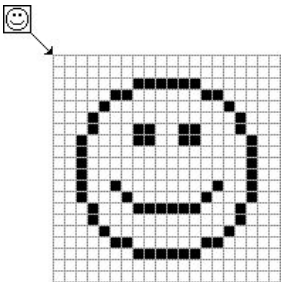
```
Image.open("data/train.jpg")
```

מקבלת את שם התמונה יחד עם שם התיקייה שבה היא נמצאת יחסית לקובץ ההרצה של התכנית, במידה ונכתב למחשב רק את שם התמונה, הוא יחפש אותה באותה תיקייה שקובץ ההרצה נמצא. ניתן כמובן לספק למחשב מיקום תמונה מוחלט, כלומר מיקום תמונה שאינו תלוי במיקום קובץ ההרצה לדוגמה:

```
Image.open("C:/data/train.jpg")
```

פלט התוכנית יהיה התמונה עצמה יחד עם הערכים 1920 1392 כך שהערך 1920 מייצג את רוחב התמונה ו- 1392 מייצג את גובה התמונה. כלומר מדובר על תמונה הכוללת 2,672,640 פיקסלים.

פיקסל היא יחידת מידע גרפית בסיסית במחשב, המתארת נקודה בתמונה דיגיטלית. החלק הקטן ביותר של הקווים, התווים, והצורות שעל צג המחשב או בתמונה המודפסת. כל תמונה בנויה ממאות אלפי עד עשרות מיליוני נקודות קטנות, ונקודה זו היא יחידת תמונה.(מקור: ויקיפדיה ttps://he.wikipedia.org/wiki/)



מקור התמונה: https://commons.wikimedia.org/wiki/File:SMILE_FACE_16x16_PIXEL_EXAMPLE1.PNG

הפעולה open של המחלקה Image מקבלת מחרוזת המייצגת את מיקום התמונה. הפעולה מחזירה עצם בשם img המייצג את התמונה.

הפעולה img.size מחזירה רשימה ובה האורך והרוחב של התמונה.

הפעולה img.show מציגה את התמונה.

הספריה PIL משמשת בפרק זה לטעינה ועיבוד תמונות לאור יכולותיה המורחבות ולצרכים הפדגוגיים. בהמשך הספר נעזר בספריה matplotlib המאפשרת לטעון תמונות ישירות לתוך מערכי numpy.

פעולות על תמונה

המחלקה Image כולל מספר פעולות המאפשרות לנו לבצע עיבוד תמונה. בקוד הבא נדגים את הפעולות סיבוב תמונה (rotate), חיתוך מקטע מתמונה (corp), שינוי גודל של תמונה (resize) והיפוך (transpose) של תמונה :

```
from PIL import Image


img = Image.open("data/train.jpg")

width, height = img.size

img45 = img.rotate(45)

area = (width/2-200, height/2-200, width/2+200, height/2+200)

img_crop = img.crop(area)

newsize = (width//4, height//4)

img_resize = img.resize(newsize)

img_transpose = img.transpose(Image.FLIP_LEFT_RIGHT)


img45.save("data/train45.jpg")

img_crop.save("data/train_cropped.jpg")

img_resize.save("data/train_resized.jpg")

print("Old image size:", img.size)

print("New image size:", img_resize.size)

img45.show()

img_crop.show()

img_resize.show()

img_transpose.show()
```

הפעולה img.rotate מקבלת זווית ומחזירה תמונה הכוללת הטייה בזווית הרצויה

הפעולה img.crop מבצעת חיתוך של קטע מתמונה על פי פרמטר שנקלט דרך המשתנה area

```
area = (width/2-200, height/2-200, width/2+200, height/2+200)
```

הפעולה img.resize מבצעת שינוי בגודל התמונה על פי פרמטר המתקבל דרך המשנה newsize

```
newsize = (width//4, height//4)

img_resize = img.resize(newsize)
```

הפעולה img.transpose מבצעת היפוך לתמונה על פי הפרמטר שהיא מקבלת. בדוגמה שלנו ההפוך יהיה מימין לשמאל (Image.FLIP_LEFT_RIGHT)

```
img_transpose = img.transpose(Image.FLIP_LEFT_RIGHT)
```

הפעולה save שומרת את התמונה כקובץ בשם המסופק לפעולה כפרמטר

```
img45.save("data/train45.jpg")
```

המרת צבעים בתמונה

נדגים כיצד מעבירים תמונה צבעונית לשחור לבן:

```
from PIL import Image

img = Image.open("data/train.jpg")


img1 = img.convert("L")

img2 = img.convert("1")

img1.show()

img2.show()
```

הפעולה img.convert מקבלת פרמטר המגדיר את אופי ההמרה.

הפרמטר "L" מבצע המרה של התמונה לשחור לבן, כלומר כל פיקסל יכול להיות ערך בודד בין 0 ל- 255.
המרת תמונה מצבעונית לשחור לבן או לגווני אפור תתבצע על ידי הנוסחה הבאה:

L = R * 299/1000 + G * 587/1000 + B * 114/1000

כאשר L יהיה ערכו של הפיקסל החדש והערכים R , G , B הם ערכי האדום, ירוק וכחול של הפיקסל המקורי.

הפרמטר "1" מציין המרה לתמונה בינארית, כלומר לתמונה חדשה שבה כל פיקסל יכול להיות אחד מ-2 ערכים 0 או 255.

הפעולה מחזירה עצם תמונה לאחר ההמרה. נדגים את פלט התוכנית לאחר ההמרה:


תמונה בינארית


תמונה שחור/לבן

בהמשך הספר נפתח גם אנו קוד המממש אלגוריתם זה.

פירוק התמונה לצבעים המרכיבים אותה

כל פיקסל בתמונה בנוי משלושה ערכים. הראשון עוצמת האדום, השני עוצמת הירוק והשלישי עוצמת הכחול. כדי להבין עיקרון זה נבחן תוכנית שמפרקת את התמונה לשלוש תמונות שונות כאשר בכל תמונה רואים צבע אחד בלבד. להלן קוד התוכנית:

```
from PIL import Image

img = Image.open("data/train.jpg")

width, height = img.size

data = img.getdata()

R = []

G = []

B = []

for i in data:

    R.append((i[0], 0, 0))

    G.append((0, i[1], 0))

    B.append((0, 0, i[2]))

imgR = Image.new(mode = "RGB", size = (width, height))

imgG = Image.new(mode = "RGB", size = (width, height))
```

```
imgB = Image.new(mode = "RGB", size = (width, height))

imgR.putdata(R)

imgG.putdata(G)

imgB.putdata(B)

imgR.show()

imgG.show()

imgB.show()
```

נקבל את הפלט הבא:

הפעולה img.getdata הופכת את הערכים של כל אחד מהפיקסלים המרכיבים את התמונה למערך של מספרים כאשר כל פיקסל מקבל 3 מספרים המייצגים את שלוש הצבעים.

מבנה הנתונים של תמונה השמור במחשב עם סיומת jpg אינו בנוי כמערך של פיקסלים כאשר לכל פיקסל שלושה צבעים. הסיבה לכך היא שקובץ jpg שומר את המידע על הפיקסלים במבנה דחוס וזאת במטרה לשמור במחשב קובץ קטן ככל הניתן תוך כדי שמירה על איכות התמונה. הפעולה getdata מבצעת הליך המשחזר על הערכים של כל פיקסל.

למידע נוסף על מבנה קובץ jpg ניתן לקבל בקישור הבא:

https://he.wikipedia.org/wiki/JPEG

לאחר פענוח מערך הפיקסלים על ידי הפעולה getdata נגדיר 3 מערכים בשמות R ,G ו- B ונשמור בכל מהם צבע בודד.

```
R = []

G = []

B = []
```

```
for i in data:
    R.append((i[0], 0, 0))
    G.append((0, i[1], 0))
    B.append((0, 0, i[2]))
```

נעזר בקוד הבא כדי להבין טוב יותר כיצד נראה כל פיקסל:

```
from PIL import Image
img = Image.open("data/train.jpg")
data = img.getdata()
for i in range(100):
    print("pixel", i , "=" , data[i])
```

פלט תוכנית זו יציג לנו את 100 הפיקסלים הראשונים מהשורה הראשונה של הקובץ train.jpg מיתוך 2,672,640 הפיקסלים המרכיבים אותה. ניתן לראות כי ערך האדום הינו 0, ירוק 42-42 וכחול 117-118. ערכים אלו הגיוניים מכיוון שהם מייצגים את כחול השמיים.

```
pixel 1 = (0, 41, 117)    pixel 26 = (0, 42, 118)   pixel 51 = (0, 42, 118)   pixel 76 = (0, 42, 118)
pixel 2 = (0, 41, 117)    pixel 27 = (0, 42, 118)   pixel 52 = (0, 42, 118)   pixel 77 = (0, 42, 118)
pixel 3 = (0, 41, 117)    pixel 28 = (0, 42, 118)   pixel 53 = (0, 42, 118)   pixel 78 = (0, 42, 118)
pixel 4 = (0, 41, 117)    pixel 29 = (0, 42, 118)   pixel 54 = (0, 42, 118)   pixel 79 = (0, 42, 118)
pixel 5 = (0, 41, 117)    pixel 30 = (0, 42, 118)   pixel 55 = (0, 42, 118)   pixel 80 = (0, 42, 118)
pixel 6 = (0, 41, 117)    pixel 31 = (0, 42, 118)   pixel 56 = (0, 42, 118)   pixel 81 = (0, 42, 118)
pixel 7 = (0, 41, 117)    pixel 32 = (0, 41, 117)   pixel 57 = (0, 42, 118)   pixel 82 = (0, 42, 118)
pixel 8 = (0, 41, 117)    pixel 33 = (0, 41, 117)   pixel 58 = (0, 42, 118)   pixel 83 = (0, 42, 118)
pixel 9 = (0, 41, 117)    pixel 34 = (0, 41, 117)   pixel 59 = (0, 42, 118)   pixel 84 = (0, 42, 118)
pixel 10 = (0, 41, 117)   pixel 35 = (0, 41, 117)   pixel 60 = (0, 42, 118)   pixel 85 = (0, 42, 118)
pixel 11 = (0, 41, 117)   pixel 36 = (0, 41, 117)   pixel 61 = (0, 42, 118)   pixel 86 = (0, 42, 118)
pixel 12 = (0, 41, 117)   pixel 37 = (0, 41, 117)   pixel 62 = (0, 42, 118)   pixel 87 = (0, 42, 118)
pixel 13 = (0, 41, 117)   pixel 38 = (0, 41, 117)   pixel 63 = (0, 42, 118)   pixel 88 = (0, 42, 118)
pixel 14 = (0, 41, 117)   pixel 39 = (0, 41, 117)   pixel 64 = (0, 42, 118)   pixel 89 = (0, 42, 118)
pixel 15 = (0, 41, 117)   pixel 40 = (0, 41, 117)   pixel 65 = (0, 42, 118)   pixel 90 = (0, 42, 118)
pixel 16 = (0, 41, 117)   pixel 41 = (0, 41, 117)   pixel 66 = (0, 42, 118)   pixel 91 = (0, 42, 118)
pixel 17 = (0, 41, 117)   pixel 42 = (0, 41, 117)   pixel 67 = (0, 42, 118)   pixel 92 = (0, 42, 118)
pixel 18 = (0, 41, 117)   pixel 43 = (0, 41, 117)   pixel 68 = (0, 42, 118)   pixel 93 = (0, 42, 118)
pixel 19 = (0, 41, 117)   pixel 44 = (0, 41, 117)   pixel 69 = (0, 42, 118)   pixel 94 = (0, 42, 118)
pixel 20 = (0, 41, 117)   pixel 45 = (0, 41, 117)   pixel 70 = (0, 42, 118)   pixel 95 = (0, 42, 118)
pixel 21 = (0, 41, 117)   pixel 46 = (0, 41, 117)   pixel 71 = (0, 42, 118)   pixel 96 = (0, 42, 118)
pixel 22 = (0, 41, 117)   pixel 47 = (0, 41, 117)   pixel 72 = (0, 42, 118)   pixel 97 = (0, 42, 118)
pixel 23 = (0, 41, 117)   pixel 48 = (0, 42, 118)   pixel 73 = (0, 42, 118)   pixel 98 = (0, 42, 118)
pixel 24 = (0, 42, 118)   pixel 49 = (0, 42, 118)   pixel 74 = (0, 42, 118)   pixel 99 = (0, 42, 118)
pixel 25 = (0, 42, 118)   pixel 50 = (0, 42, 118)   pixel 75 = (0, 42, 118)
```

תרגיל

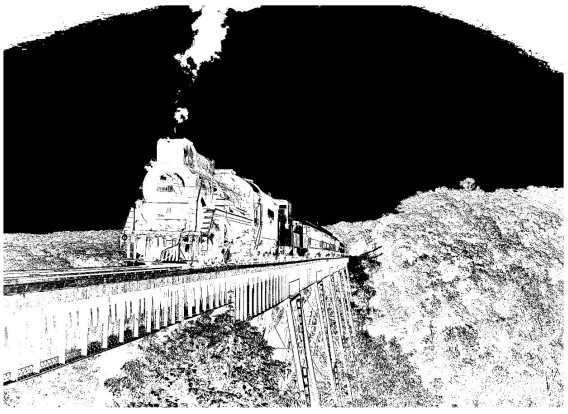
ראינו קודם את הפעולה convert שהופכת תמונה צבעונית לתמונה בינארית (תמונה שבה כל פיקסל מקבל את הערך 1 או 0). כאמור כל פיקסל יכול להיות שחור או לבן בלבד. זאת בניגוד לתמונת גווני אפור שבה יש מגוון ערכים בין שחור ללבן. כתבו קוד בשפת python הממיר תמונה צבעונית לתמונה בינארית.

פתרון

להלן קוד התוכנית:

```
from PIL import Image
img = Image.open("data/train.jpg")
width, height = img.size
data = img.getdata()
BIN = []
for i in data:
    avg = (i[0]+i[1]+i[2])/3
    if avg<60:
        BIN.append((255, 255, 255))
    else:
        BIN.append((0, 0, 0))
imgBIN = Image.new(mode = "RGB", size = (width, height))
imgBIN.putdata(BIN)
imgBIN.show()
```

להלן דוגמה לפלט התוכנית:



תרגיל

שנה את קוד התרגיל הקודם כדי ליצור תמונה בשחור לבן במקום תמונה בינארית.

פתרון

להלן קוד התוכנית:

```
from PIL import Image
img = Image.open("data/train.jpg")
width, height = img.size
data = img.getdata()
BIN = []
for i in data:
    avg = int((i[0]+i[1]+i[2])/3)
    BIN.append((avg, avg, avg))
imgBIN = Image.new(mode = "RGB", size = (width, height))
imgBIN.putdata(BIN)
imgBIN.show()
```

להלן דוגמה לפלט התוכנית:

תרגיל

בחזית הרכבת יש לוחית זיהוי הכוללת מספר. כתבו תוכנית שתציג תמונה הכוללת רק את מספר לוחית הזיהוי לתמונה בינארית חדשה.

פתרון

```
from PIL import Image
img = Image.open("data/train.jpg")
width, height = img.size
area = (500, 580, 640, 650)
img_crop = img.crop(area)
data = img_crop.getdata()
BIN = []
for i in data:
    avg = (i[0]+i[1]+i[2])/3
    if avg<60:
        BIN.append((255, 255, 255))
    else:
        BIN.append((0, 0, 0))
width, height = img_crop.size
imgBIN = Image.new(mode = "RGB", size = (width, height))
imgBIN.putdata(BIN)
newsize = (width*3, height*3)
imgBIN = imgBIN.resize(newsize)
imgBIN.show()
```

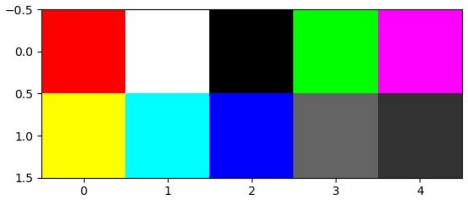
להלן דוגמה לפלט התוכנית:

יצירת תמונה תוך שימוש בכלי numpy

נדגים יצירת תמונה חדשה המופקת ממערך numpy array

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

pixels =[
        [  [255,0,0],[255,255,255],[0,0,0],[0,255,0],[255,0,255]          ],
        [  [255,255,0],[0,255,255],[0,0,255],[100,100,100],[50,50,50]  ]
        ]

array = np.array(pixels, dtype=np.uint8)
plt.imshow(array)
plt.show()
```

נקבל את הפלט הבא:



נעזר בידע שלמדנו בעבודה עם numpy ונייצר מערך תלתמימדי באופן הבא:

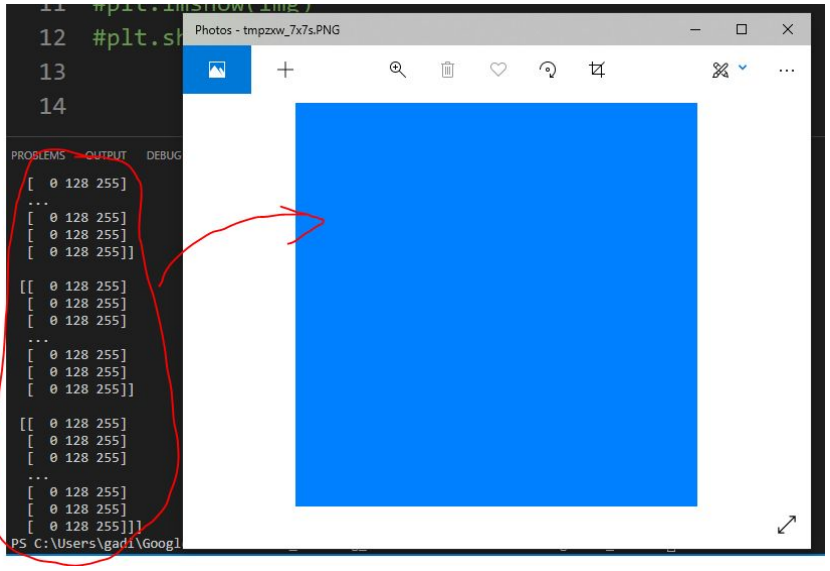
```
pixels = np.zeros([400,400,3],dtype=np.uint8)
```

כלומר 400 שורת על 400 עמודות כאשר כל רשומה כוללת 3 תאים. בכל תא נשמור משתנה מטיפוס uint8 כולמר ערך בין 0 ל- 255 כפי שכבר ראינו באפיון רמת הצבע של כל פיקסל.

הפעם נציג את התמונה עם ספריית matplotlib. להלן קוד התוכנית:

```
import numpy as np

import matplotlib.pyplot as plt


pixels = np.zeros([400,400,3],dtype=np.uint8)

pixels[:,:] = [0, 128, 255]

print(pixels)

plt.imshow(pixels)

plt.show()
```

נקבל את הפלט הבא:



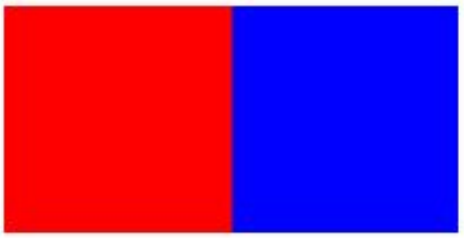
ניתן לראות שקיבלנו מערך numpy הכולל 160,000 פיקסלים שכל אחד מהם מכיל את הערכים: [0 128 255]

ניתן ליצור תמונות הכוללת יותר מצבע אחד. להלן דוגמה:

```
import numpy as np
```

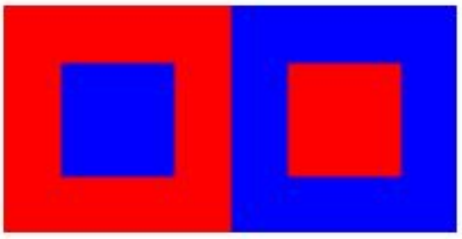
```
import matplotlib.pyplot as plt


pixels = np.zeros([100,200,3],dtype=np.uint8)

pixels[:,:100] = [255, 0, 0]

pixels[:,100:] = [0, 0, 255]


plt.imshow(pixels)

plt.show()
```

נקבל את הפלט הבא:



תרגיל

צור תמונה הכוללת את הצורה הזו:



פתרון

```
import matplotlib.pyplot as plt

import numpy as np
```

```
pixels = np.zeros([100,200,3],dtype=np.uint8)
pixels[:,:100] = [255, 0, 0]
pixels[:,100:] = [0, 0, 255]
pixels[25:75,25:75] = [0, 0, 255]
pixels[25:75,125:175] = [255, 0, 0]
print(pixels.shape,type(pixels))
plt.imshow(pixels)
plt.show()
```

קריאת תמונה לתוך מערך numpy

קריאת קובץ תמונה לתוך מערך תוך כדי עיבוד בסיסי של התמונה:

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

img= imglib.imread("train.jpg")
data = img.copy()
data[0:100,0:100]=[255, 0, 0]
data[100:200,100:200]=[0, 255, 0]
data[200:300,200:300]=[0, 0, 255]
data[300:400,300:400]=[255, 0, 255]
data[400:500,400:500]=[0, 255, 255]
data[500:600,500:600]=[255, 255, 255]
```

```
plt.imshow(data)
plt.show()
```

נקבל את הפלט הבא:

נדגיש שההוראה שבקוד הבא:

```
data[0:100,0:100]=[255, 0, 0]
```

זהה לשלוש ההוראות שבקוד הנ"ל:

```
data[0:100,0:100,0]=255
data[0:100,0:100,1]=0
data[0:100,0:100,2]=0
```

חיפש מאפיינים לזיהוי פריטים בתמונה (הכנות למשחק ים יבשה בגרסת מכונה לומדת)

לצורך זיהוי פריטים בתמונה או בקיצור זיהוי תמונות על ידי מכונה לומדת יש צורך לארגן מערך של תמונות מתויגות. נבחן את הסוגיה על ידי התבוננות על התמונה הבא:

Image by maja7777 from Pixabay

כיצד אם כן נאפיין מי זה הכלב ומי החתול בתמונה? האם כל הלבנים חתולים? האם כל בעלי הפרווה כלבים? האם לכל מי שיש אף בצורת משולש הוא חתול? האם כל מי שיש לו אוזן ורודה הוא כלב?

נראה שיש צורך לחדד את הנושא ולעזור לנו לאפיין את הגורמים המבדילים בין פריטים. תהליך זה נקרא במכונות לומדות: Feature Selection in Machine Learning והוא בא לספק לנו כלים להגדרת המאפיינים שיכולים להבדיל בין דברים שאנו רוצים לסווג.

לא תמיד שלב זה מתקיים, בייחוד ברשתות עמוקות עליהן נלמד בהמשך הספר. ברשתות עמוקות פעמים רבות נבחר לספק לרשת את הנתונים כפי שהם ללא חישובים מקדימים. השכבות הראשונות של הרשת יממשו את חילוץ המאפיינים בעצמם על ידי למידה של המאפיינים החשובים לסיווג.

במקרה שלנו כאשר מדובר בתמונות, אנו מתייחסים ליכולת לאפיין מספר תכונות בתוך תמונה במטרה לסווג פריטים בתוך תמונה ברמת וודאות גבוה.

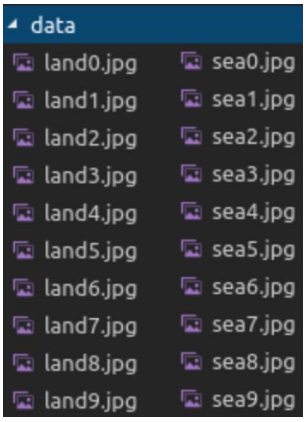
לצורך התנסות בנושא ובמטרה לתרגל עבודה עם תמונות נחפש מאפיינים Features שיכולים להבדיל בין תמונות ים לבין תמונות יבשה. בקיצור משחק ים יבשה בגרסת מכונה לומדת (נו טוב לא ממש דומה למשחק הילדים ששיחקנו פעם).

נוריד מהאינטרנט 10 תמונות של ים ו-10 תמונות של יבשה ונשמור אותם בתיקייה בשם data תחת השמות sea ומספר התמונה עבור כל התמונות המתארות ים ו- land ומספר התמונה עבוד כל התמונות המתארות יבשה.

ניתן להוריד תמונות שאינם מוגנות מזכויות יוצרים מאתר:

https://pixabay.com/

להלן דוגמה למבנה הקבצים בתיקייה data הכולל את התמונות.



לאחר שהורדנו למחשב 20 תמונות נחפש מאפיינים שיכולים להבדיל בין תמונות נוף של ים ליבשה. מתוך אינטואיציה כי תמונות ים יכללו יותר מהצבע הכחול ותמונות יבשה יכללו יותר מהצבע הירוק נבדוק כמאפיין אפשרי את היחס בין ממוצע הצבע הירוק תמונת יבשה לבין יחס ממוצע הצבע הכחול בכל תמונת היבשה. לשם כך נכתוב קטע קוד המציג מתוך תמונה בודדת של ים ומתוך תמונה אחת של יבשה את ממוצע הצבע הירוק ואת ממוצע הצבע הכחול בכל אחת מהם:

```
from PIL import Image

import numpy as np


img = Image.open("data/sea0.jpg")

img.load()

sea0 = np.array(img, dtype=np.uint8)


print ("sea0 - Green =" , sea0[:,:,1].sum() / sea0[:,:,1].size)

print ("sea0 - blue =" , sea0[:,:,2].sum() / sea0[:,:,2].size)


img = Image.open("data/land0.jpg")

img.load()

land0 = np.array(img, dtype=np.uint8)
```

```
print ("land0 - Green =" , land0[:,:,1].sum() / land0[:,:,1].size)

print ("land0 - blue =" , land0[:,:,2].sum() / land0[:,:,2].size)
```

עבור שתי התמונות המוצגות להלן נקבל את הפלט הבא:

Image by Michelle Maria from Pixaba Image by Sasin Tipchai from Pixabay

```
sea1 - Green = 123.1879816955684
sea1 - blue = 153.2496296965318
land1 - Green = 145.8581862745098
land1 - blue = 73.04027328431373
```

שאלה: האם זה הגיוני שהאלגוריתם שלנו מצא צבע ירוק בתמונה של הים?

תשובה: ברור שכן!, כל נקודה על גבי המסך מורכבת מפיקסלים. כל פיקסל מורכב מאדום,ירוק וכחול. אז ברור שבחלק מהפיקסלים בייחוד בלבן יש מרכיב משמעותי של ירוק בתוך כל אחד מהם.

בשלב הבא נבנה 2 רשימות הכולות את ממוצע הצבע ירוק בכל אחד מ- 20 התמונות לפי הפרדה בין תמונות ים לתמונות יבשה:

```
from PIL import Image

import numpy as np


sea_list = []

land_list = []
```

```
for i in range(10):
    img = Image.open("data/sea" + str(i) + ".jpg")
    img.load()
    data = np.array(img, dtype=np.uint8)
    sea_list.append(data[:,:,1].sum() / data[:,:,1].size)

    img = Image.open("data/land" + str(i) + ".jpg")
    img.load()
    data = np.array(img, dtype=np.uint8)
    land_list.append(data[:,:,1].sum() / data[:,:,1].size)

print("\nsea_list:\n",sea_list)
print("\nland_list:\n",land_list)
```

נקבל את הפלט הבא:

```
sea_list:
 [130.4101615341768, 131.96297276468894, 154.34126688346825, 111.7069587628866, 115.01519280118855, 155.91324930555555, 129.21721614583333, 135.44244079172367, 137.0602974487363, 107.63084548611111]

land_list:
 [121.23325928276738, 86.22154513888889, 147.22994865262908, 117.407721875, 122.99220265188997, 206.6399573089777, 154.72347739692003, 111.65416666666667, 135.62204365079364, 93.525663189269751]
```

בשלב הבא נבנה גרפים המציגים את הנתונים שקיבלנו מהשלב הקודם אך הפעם נציג את היחס בין ממוצע כל 2 צבעים בכל תמונה לפי חלוקה בין תמונות ים לתמונות יבשה:

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

sea_colors = []
for i in range(10):
```

```
    img = Image.open("data/sea" + str(i) + ".jpg")
    img.load()
    data = np.array(img, dtype=np.uint8)
    t= []
    for i in range(3):
        t.append(data[:,:,i].sum() / data[:,:,i].size)
    sea_colors.append(t)

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)
    t= []
    for i in range(3):
        t.append(data[:,:,i].sum() / data[:,:,i].size)
    land_colors.append(t)

sea_array = np.array(sea_colors)       #convert the list data type to numpy data type
land_array = np.array(land_colors)     #convert the list data type to numpy data type

# red green
plt.subplot(131)
sea_array0 = sea_array[:,0]
```

```
sea_array1 = sea_array[:,1]

land_array0 = land_array[:,0]

land_array1 = land_array[:,1]

plt.plot(sea_array0, sea_array1, 'bo', land_array0, land_array1, 'r+')

plt.xlabel("red")

plt.ylabel("green")

plt.title("Sea or Land option 1")


# red blue

plt.subplot(132)

sea_array0 = sea_array[:,0]

sea_array1 = sea_array[:,2]

land_array0 = land_array[:,0]

land_array1 = land_array[:,2]

plt.plot(sea_array0, sea_array1, 'bo', land_array0, land_array1, 'r+')

plt.xlabel("red")

plt.ylabel("blue")

plt.title("Sea or Land option 2")


# green blue

plt.subplot(133)

sea_array1 = sea_array[:,1]

sea_array2  = sea_array[:,2]

land_array1 = land_array[:,1]
```

```
land_array2= land_array[:,2]

plt.plot(sea_array1 , sea_array2 , 'bo', land_array1, land_array2, 'r+')

plt.xlabel("green")

plt.ylabel("blue")

plt.title("Sea or Land option 3")

plt.show()
```

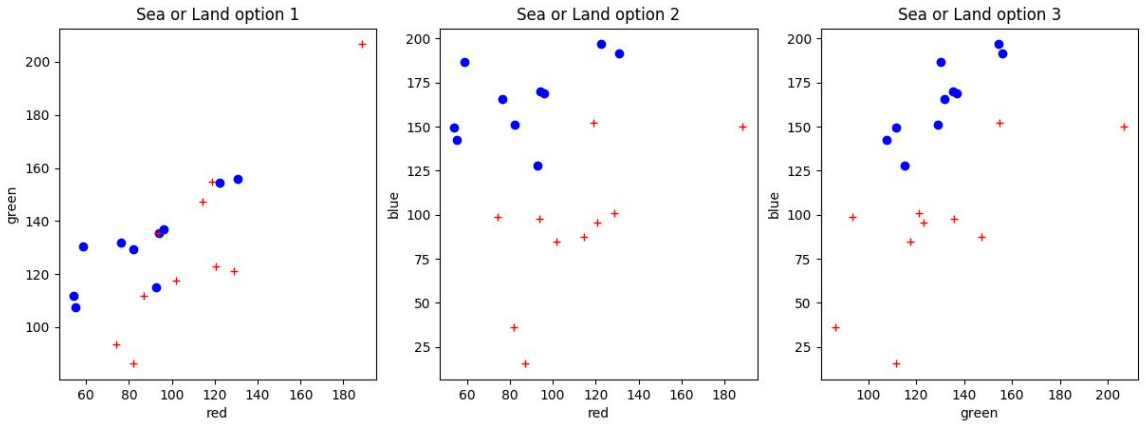
ההוראה:

```
land_colors.append([data[:,:,color].sum() / data[:,:,color].size for color in range(3)
```

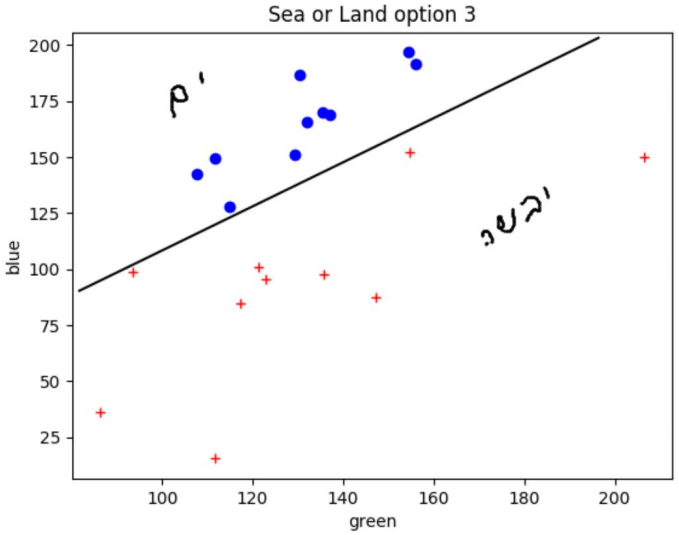
יכולה להיות גרסה מקוצרת של קטע הקוד הבא:

```
t = []

for i in range(3):

    t.append(data[:,:,i].sum() / data[:,:,i].size)

land_colors.append(t)
```

נקבל את הפלט הבא:



ניתן ללמוד מניתוח מאפייני הצבעים שבתמונות מכך שמאפיין היחס בין צבע ירוק לצבע כחול יכול לספק לנו יכולת סיווג גבוה כיוון שעל פי מדגם 20 התמונות שלנו ניתן לסווג תמונות שלא מוכרות למוכנה שלנו על פי יחס זה. נדגים זאת על ידי הגרף הבא:



בפעילות זו למדנו כיצד מייצגים תמונות במחשב, וכיצד ניתן למצוא מאפיינים של תמונות, לדוגמא מאפיין מבדיל בין תמונות נוף ים לבין תמונות נוף יבשתי.

ראינו שעל ידי בדיקת ממוצעהצבע הירוק וממוצעהצבע הכחול בכל התמונות ניתן לזהות בסבירות טובה, יחסית למדגם שעשינו, מה הנוף בתמונה (ים או יבשה). בפעילויות הבאות נמשיך ונפתח את הרעיון כדי לבנות מכונה לומדת המסוגלת לזהות בין תמונות ים לבין תמונת יבשה.

בפעילות הבאה, נתמקד בעקרונות המתמטיים המאפשרים לנו למצוא את הקו הלינארי מייצג את ההפרדה בין 2 סוגית התמונות. נעשה זאת על ידי פעולה מתמטית בשם רגרסיה לינארית.

אלגוריתמי למידת מכונה בסיסיים

פעילות 6 - אלגוריתם KNN ככלי לפיתוח מכונה לומדת

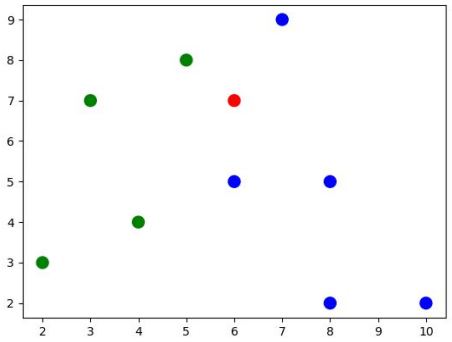
בפעילות הפעם נציג שלא כל המכונות הלומדות מבוססות על רשתות נוירונים מלאכותית. בפעילות זו נדגים אלגורית בשם K-Nearest Neighbors algorithm או בקיצור K-NN. אלגוריתם זה הוא אחד הפתרונות לפיתוח מכונות לומדות. במסגרת הפעילות נלמד ש- KNN מבוסס בין היתר על מדידת מרחק אוקלידי בין נקודות במרחב או בשם הפשוט יותר שלו, שימוש במשפט פיתגורס כדי לחשב את היתר במשולש ישר זווית.

כדי להבין את עקרון הפעולה של אלגוריתם KNN נחקור את מערך הנקודות הבא:

```
import matplotlib.pyplot as plt

import numpy as np


data = np.array( [     [ 6 , 7],
                       [ 2 , 3],
                       [ 3 , 7],
                       [ 4 , 4],
                       [ 5 , 8],
                       [ 6 , 5],
                       [ 7 , 9],
                       [ 8 , 5],
                       [ 8 , 2],
                       [10 , 2]    ])


categories = np.array([0,1,1,1,1,2,2,2,2,2])

colormap = np.array(['r', 'g', 'b'])


plt.scatter(data[:,0], data[:,1], s=100, c=colormap[categories])
```

```
plt.show()
```

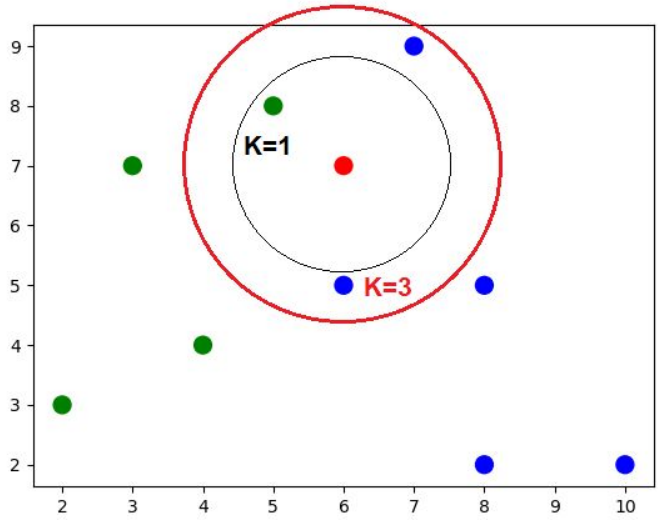
נקבל את הפלט הבא:



קיבלנו אם כן מערך הכולל 3 סוגי נקודות. נקודות כחולות, נקודות ירוקות ונקודה אחת בצבע אדום. בהנחה שיש קשר כלשהו בין כל הנקודות הכחולה וקשר אחר בין כל הנקודות הירוקות. נשאלת השאלה האם הנקודה האדומה שייכת לקבוצת הנקודות הכחולות או לקבוצת הירוקות?

על פי אלגוריתם KNN אנו יכולים לשער לאיזו קבוצה שייכת הנקודה האדומה על ידי כך שנבדוק מי השכנים שלה וכמה שכנים יש לאותה נקודה מכל סוג (כחולים וירוקים) לפי עקרון זה ניתן לשער שככל שיש יותר נקודות מסוג מסוים באותו האזור שאנו בודקים, הנקודה הנבדקת שייכת לאותה קבוצה.

נשאלת השאלה מה מספר השכנים הרצוי? נבחן את הסוגייה על ידי האיור הבא:



כאשר הגדרנו באיור את האזור בעיגול שחור מצאנו שבתוך אותו השטח קיימת רק נקודה אחת בצבע ירוק. כאשר הגדרנו את האזור בעיגול אדום קיבלנו בשטח שיש 2 נקודות כחולות ונקודה ירוקה אחת.

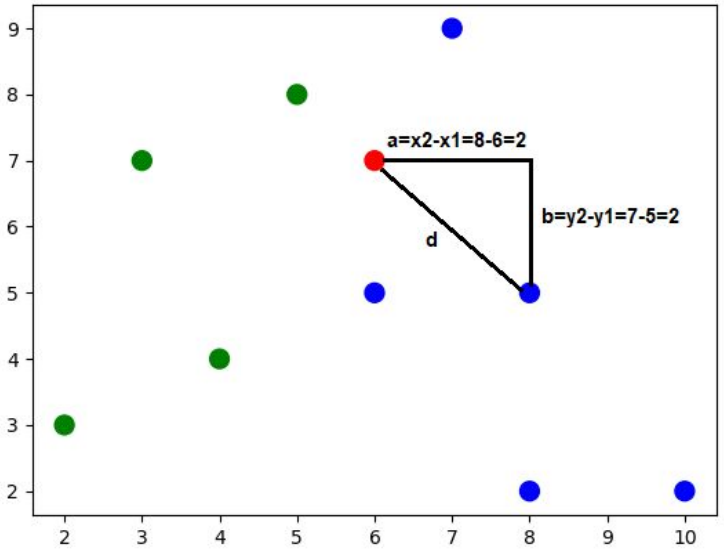
K המייצג את מספר הפריטים הכי קרובים לאותה נקודה שאנו בודקים. כאשר K=1 אנו בודקים מי הנקודה הכי קרובה לנקודה הנבדקת. כאשר K=3 אנו בודקים מי שלוש הנקודות הכי קרובות לנקודה הנבדקת. באופן זה ניתן לחזות האם הנקודה שאנו בודקים שייכת לקבוצה הכחולה או לקבוצה הירוקה. ניתן לקבוע ל-K גם ערכים גדולים יותר מ-3.

מהדוגמה שבאיור אנו כבר מקבלים בעייה בסיווג הנקודה האדומה. כי כאשר K=1 הנקודה הנבדקת שייכת לקבוצה הירוקה. אך כאשר K=3 הנקודה הנבדקת שייכת לקבוצה הכחולה.

מדוגמה זו ניתן להבין לצד הפשטות של השיטה גם את הקושי שלה. כי שינוי מספר השכנים הקרובים K משפיע על קביעת הסיווג.

מרחק אוקלידי בין נקודות במרחב

כדי לממש את האלגוריתם KNN אנו זקוקים לאלגוריתם לחישוב מרחק בין נקודות במרחב. נדגים זאת על ידי מציאת המרחק בין 2 נקודות על פני מישור דו-מימדי.



המרחק d בין 2 הנקודות באיור מחושב על ידי המשוואה הבא:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

נדגים את עקרון מדידת מרחק דרך דוגמת הקוד הבא:

```
import numpy as np


data = np.array([      [ 6.0 , 7.0],
                       [ 2.0 , 3.0],
                       [ 3.0 , 7.0],
                       [ 4.0 , 4.0],
                       [ 5.0 , 8.0],
                       [ 6.0 , 5.0],
                       [ 7.0 , 9.0],
                       [ 8.0 , 5.0],
                       [ 8.0 , 2.0],
                       [10.0 , 2.0]])


def euclidean_distance(p1, p2):
      dx = float(p1[0]-p2[0])
      dy = float(p1[1]-p2[1])
      d = np.power(dx,2) + np.power(dy,2)
      return np.sqrt(d)


print("\nAll Euclidean Distance from point: ",data[0], "\n")
for point in data:
      distance = euclidean_distance(data[0], point)
      print(distance)
```

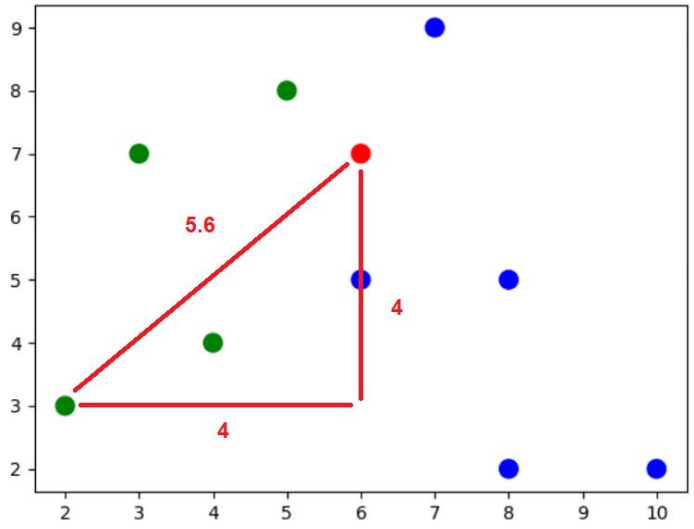
נקבל את הפלט הבא:

```
All Euclidean Distance from point:  [6. 7.]

0.0
5.656854249492381
3.0
3.605551275463989
1.4142135623730951
2.0
2.23606797749979
2.8284271247461903
5.385164807134504
6.4031242374328485
```

נבחן את שלוש התוצאות הראשונות בפלט:

הערך הראשון הוא 0 כי המרחק בין נקודה לעצמה הוא אפס.

הערך השני הוא 5.656 כי המרחק בין נקודה (6,7) לנקודה (2,3) הוא שורש של 4 בריבוע ועוד 4 בריבוע.



הערך השלישי הוא 3 כי המרחק בין נקודה (6,7) לנקודה (3,7) הוא פשוט 3 כי יש שינוי רק על ציר x.

מרחק אוקלידי על גבי מערכת מרובת צירים

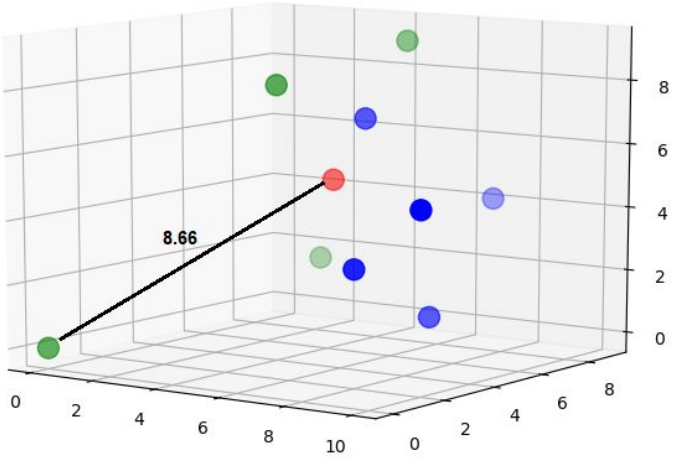
נכתוב מחדש את הפעולה euclidean_distance כדי שתתאים לחישוב מרחק אוקלידי ביותר מ-2 מימדים. להלן דוגמת קוד המחשבת מרחקים מ-9 נקודות על פני מערכת צירים בעלת 3 מימדים, כאשר המרחק מחושב מהנקודה (5,5,5):

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

data = np.array( [      [ 5.0 , 5.0, 5.0],
                        [ 0.0 , 0.0, 0.0],
                        [ 3.0 , 7.0, 2.0],
                        [ 4.0 , 4.0, 8.0],
                        [ 5.0 , 8.0, 9.0],
                        [ 6.0 , 5.0, 7.0],
                        [ 7.0 , 9.0, 4.0],
                        [ 8.0 , 5.0, 1.0],
                        [ 8.0 , 2.0, 3.0],
                        [10.0 , 2.0, 5.0]    ])

def euclidean_distance(p1, p2):
    d = 0.0
    for i in range(len(p1)):
        a = float(p1[i])
        b = float(p2[i])
        d += np.power((a-b),2)
    d = np.sqrt(d)
    return d

print("\nAll Euclidean Distance from point: ",data[0], "\n")
for point in data:
        distance = euclidean_distance(data[0], point)
```

```
    print(distance)


categories = np.array([0,1,1,1,1,2,2,2,2,2])
colormap = np.array(['r', 'g', 'b'])
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(data[:,0], data[:,1],data[:,2], s=150, c=colormap[categories])
plt.show()
```

נקבל את הפלט הבא:

```
All Euclidean Distance from point:  [5. 5. 5.]

0.0
8.660254037844387
4.123105625617661
3.3166247903554
5.0
2.23606797749979
4.58257569495584
5.0
4.69041575982343
5.830951894845301
```



ניתן לראות שהפעולה חישבה נכון את המרחק בין נקודה (0,0,0) לבין הנקודה (5,5,5) כ- 8.66 כי:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2}$$

$$d = \sqrt{(5 - 0)^2 + (5 - 0)^2 + (5 - 0)^2} = 8.66$$

ניתן לחשב את המרחק על ידי הפעולה np.norm של ההפרש בין הוקטורים.

השלב הבא בכתיבת קוד יהיה לכתוב פעולה בשם KNN. הפעולה תקבל מערך הנקודות כפרמטר בשם train, את הנקודה שאליה אנו רוצים לחפש את הנקודות הקרובות ביותר בשם test ופרמטר נוסף בשם K שקובע את מספר הנקודות אותה הפעולה תחזיר.

פעולה זו תבצע את ההוראות הבאות:

- תעבור בלולאה על כל הערכים במערך train ובכל פעם תזמן את הפעולה euclidean_distance
- הערך המוחזר מפעולה euclidean_distance נכנס למערך פנימי הכולל את המרחק ואת ערכי הנקודה שנבדקה.
- לאחר סיום מדידת כל הנקודות נמיין את המערך על פי המרחקים.
- לבסוף נעבור על הלולאה הממוינת ונחלץ ממנה את K האיברים שאותם אנו רוצים לקבל.

הלהן מימוש הקוד:

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

data = np.array( [     [ 6.0 , 7.0],
                       [ 2.0 , 3.0],
                       [ 3.0 , 7.0],
                       [ 4.0 , 4.0],
                       [ 5.0 , 8.0],
                       [ 6.0 , 5.0],
                       [ 7.0 , 9.0],
                       [ 8.0 , 5.0],
                       [ 8.0 , 2.0],
                       [10.0 , 2.0]   ])

def euclidean_distance(p1, p2):
```

```
    d = 0.0
    for i in range(len(p1)):
        a = float(p1[i])
        b = float(p2[i])
        d += np.power((a-b),2)
    d = np.sqrt(d)
    return d


def KNN(train, test, K):
        distances = []
        NN = []
        for p in train:
                dist = euclidean_distance(test, p)
                distances.append((p, dist))
        distances.sort(key=lambda dist: dist[1])
        distances = np.delete(distances,[0], axis=0)
        for i in range(K):
                NN.append(distances[i][0])
        return NN


neighbors = KNN(data, data[0], 3)
for neighbor in neighbors:
        print(neighbor)
```

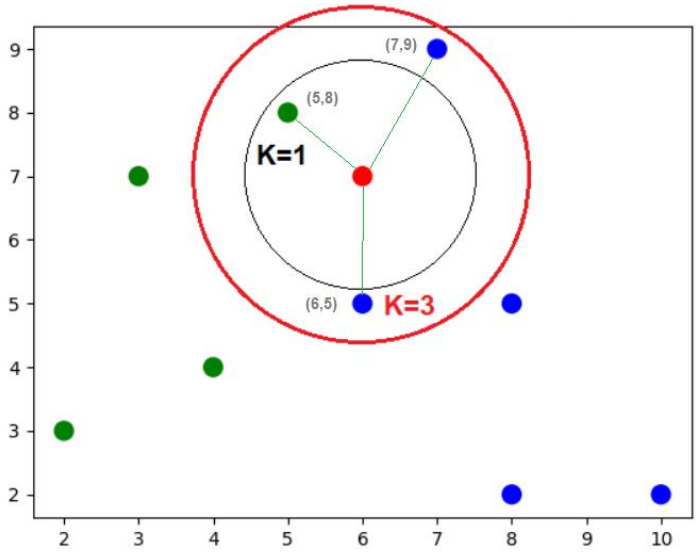
נקבל את הפלט הבא עבור K=3:

```
[5. 8.]
[6. 5.]
[7. 9.]
```

ואת פלט הבא עבור K=1:

```
[5. 8.]
```

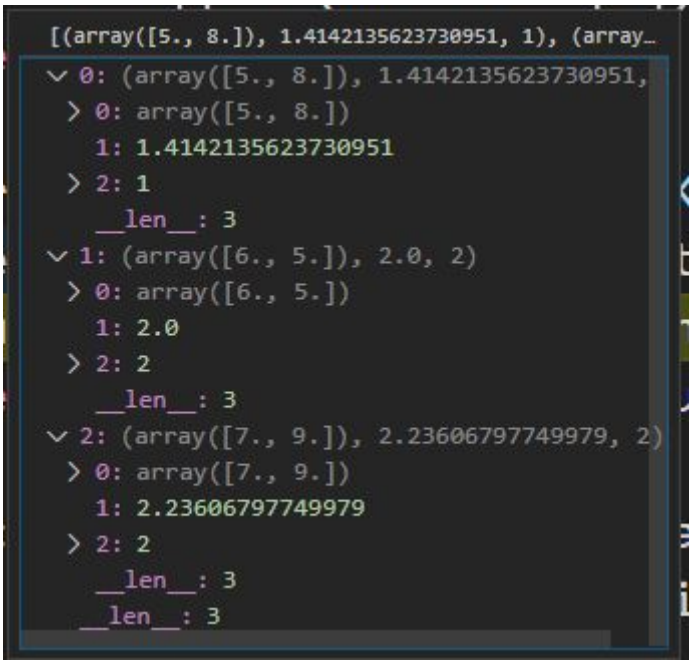
נבחן את התוצאות מול התרשים הבא:



השלב האחרון בכתיבת קוד ליישום אלגוריתם KNN במכונה לומדת יהיה לכתוב את הפעולה predict.
הפעולה תקבל מערך נקודות כפרמטר בשם train, נקודה בודדת כפרמטר בשם test שאליה אנו רוצים לחפש את הנקודות הקרובות ביותר, מערך תגיות בשם lbl המציין לאיזו קבוצה שייכת כל נקודה במערך train ופרמטר נוסף בשם K שקובע את מספר הנקודות שעל פיהם נקבע לאיזו קבוצה שייכת הנקודה test. הפעולה תחזיר פרמטר אחד שהוא מספר הקבוצה.

פעולה זו תבצע את השלבים הבאים:

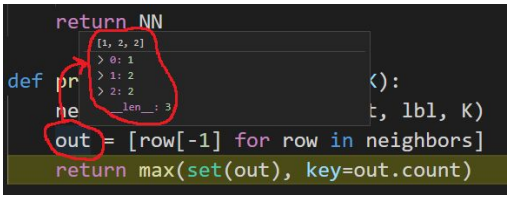
נזמן את הפעולה KNN כדי לקבל את שלוש הנקודות הכי קרובות לנקודה test להלן צילום מסך של הערכים שמקבלים מהפעולה KNN:



על פי האיור ניתן לראות שאנו מקבלים מערך הכולל 3 איברים שבו כל איבר בנוי ממערך נוסף הכולל את ערכי הנקודה, המרחק ומספר הקבוצה שאליה שייכת הנקודה.

השלב הבא יהיה לספור כמה נקודות יש לכל קבוצה ולהציג את הקבוצה שיש לה הכי הרבה נקודות:

```
out = [row[-1] for row in neighbors]
return max(set(out), key=out.count)
```



ניתן לראות שערך המערך out שווה לערכים [1,2,2]

בשלב האחרון הפעולה max תחזיר את הספרה שמופיעה הכי הרבה פעמים, במקרה שלנו 2.

כדי לקצר את קוד הפעולה השתמשתי בלולאה מקוצרת העוברת על האיבר האחרון (-1) של הרשימה neighbor. כמו כן השתמשתי בפעולה max הכוללת פרמטר בשם key=out.count הסוכם את כמות האיברים במערך.

נוסף על כתיבת הפעולה predict יש צורך לשנות את הפעולה KNN כך שתקבל גם את מערך התגיות lbl.
כמו כן נעשה שינוי במבנה המערך distances שיכיל גם את התווית של כל אחד מהנקודות. להלן מימוש הפעולה KNN לאחר השינויים הנדרשים בתרגיל:

```
def KNN(train, test, lbl, K):
    distances = []
    for t, l in zip(train,lbl):
        dist = euclidean_distance(test, t)
        distances.append((t, dist, l[0]))
    distances.sort(key=lambda dist: dist[1])
    NN = []
    for i in range(K):
        NN.append(distances[i])
    return NN
```

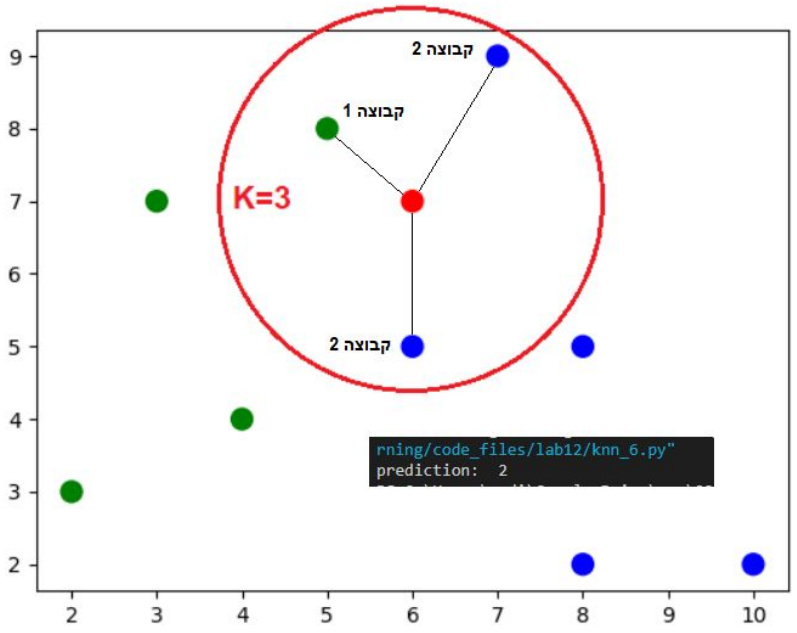
להלן קוד התוכנית המלא:

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

train_data = np.array( [
        [ 2.0 , 3.0 ],
        [ 3.0 , 7.0 ],
        [ 4.0 , 4.0 ],
        [ 5.0 , 8.0 ],
        [ 6.0 , 5.0 ],
        [ 7.0 , 9.0 ],
        [ 8.0 , 5.0 ],
        [ 8.0 , 2.0 ],
        [10.0 , 2.0 ]   ])
```

```
train_lbl = np.array( [
            [ 1 ],
            [ 1 ],
            [ 1 ],
            [ 1 ],
            [ 2 ],
            [ 2 ],
            [ 2 ],
            [ 2 ],
            [ 2 ]   ])

test_data = np.array( [[ 6.0 , 7.0 ]])

def euclidean_distance(p1, p2):
    d = 0.0
    for i in range(len(p1[0])):
        a = float(p1[0,i])
        b = float(p2[i])
        d += np.power((a-b),2)
    d = np.sqrt(d)
    return d

def KNN(train, test, lbl, K):
        distances = []
        for t, l in zip(train,lbl):
                dist = euclidean_distance(test, t)
```

```
            distances.append((t, dist, l[0]))
    distances.sort(key=lambda dist: dist[1])
    NN = []
    for i in range(K):
            NN.append(distances[i])
    return NN


def predict(train, test, lbl, K):
    neighbors = KNN(train, test, lbl, K)
    out = [row[-1] for row in neighbors]
    return max(set(out), key=out.count)


prediction = predict(train_data, test_data, train_lbl, 3)
print("prediction: " , prediction)
```

נקבל את הפלט הבא:

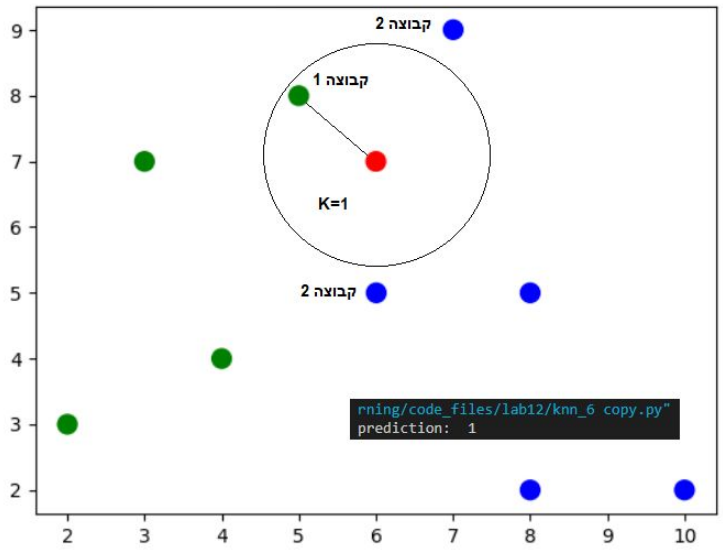


קיבלנו שהנקודה האדומה שייכת לקבוצה מספר 2.

תרגיל

שנו את הפרמטר K=3 ל- K=1 ובדקו האם גם במצב זה הנקודה האדומה שייכת לקבוצה 2. נמקו את תשובתכם.

פתרון

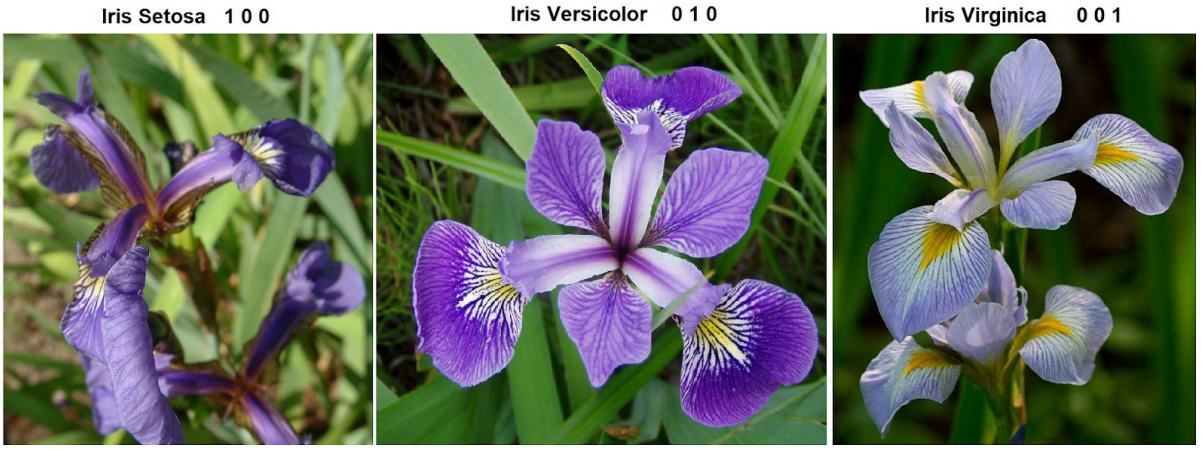


קיבלנו הפעם שיוך לקבוצה 1 כי עבור K=1 יש רק נקודה אחת מקבוצה 1 שהיא בעצם הנקודה הכי קרובה.

משימה - מכונה לומד לסיווג פרחים תוך שימוש באלגוריתם KNN.

בפעילות הקודמת כתבנו קוד העושה שימוש ברשת נוירונים מלאכותית ANN במטרה לבצע סיווג של תמונות פרחים ל- 3 סוגי אירוסים. למדנו שתצאות הסיווג לא היו ממש מדוייקות והיה צורך לבצע מספר אימונים לרשת כדי לקבל תוצאות סבירות. על כן כדאי לנסות אלגוריתם אחר שיכול בצורה פשוטה יותר לבצע את משימת סיווג הפרחים. במשימה זו נבחן את אלגוריתם KNN כדי לסווג פרחים.

מערך הנתונים הכולל מידע מתויג על 3 סוגים שונים של אירוסים. להלן תמונות של שלשות האירוסים:



מקור התמונה: https://commons.wikimedia.org/wiki/Iris_Iridaceae (לשימוש חוזר עם אפשרות לשינויים)

מאגר הנתונים כולל נתונים מתוקפים מחקרית המסווגים שלושת פרחים על פי 4 מאפיינים שהם:

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

להלן תמונה שתסביר את המאפיינים כל גבי תמונה של הפרח

ניתן להוריד את קובץ הנתונים ישירות מהקישור הבא:

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 חמישית הכוללת את שם הפרח שאותו מדדו. מכאן שיש לנו 150 שורות של מידע מתויג.

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. ]
```

בשלב הבא יש צורך להתאים את מערך הנתונים כדי שיכנס לתוך אלגוריתם KNN באופן הבא:

- נשנה את שם הפרח למספרים 1 עד 3 בהתאמה.
- שנערבב את סוגי הפרחים.
- 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.

להלן מימוש הדברים בקוד:

```
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:]

print("\nrandom_iris_data: \n",colored(random_iris_data, 'red'),"\n")

print("\ntrain_data: \n",colored(train_data, 'blue'),"\n")

print("\ntrain_lbl: \n",colored(train_lbl, 'blue'),"\n")

print("\ntest_data: \n",colored(test_data, 'green'),"\n")

print("\ntest_lbl: \n",colored(test_lbl, 'green'),"\n")
```

נבחן את הנתונים שקיבלנו:

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

נשלב את קוד הכנת מבנה הנתונים יחד עם שלוש הפעולות שכתבנו למימוש אלגוריתם KNN ונקבל את הקוד הבא:

```
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:]

def euclidean_distance(p1, p2):
    d = 0.0
```

```
    for i in range(len(p1)):
        a = float(p1[i])
        b = float(p2[i])
        d += np.power((a-b),2)
    d = np.sqrt(d)
    return d


def KNN(train, test, lbl, K):
        distances = []
        for t, l in zip(train,lbl):
                dist = euclidean_distance(test, t)
                distances.append((t, dist, l[0]))
        distances.sort(key=lambda dist: dist[1])
        NN = []
        for i in range(K):
                NN.append(distances[i])
        return NN


def predict(train, test, lbl, K):
        neighbors = KNN(train, test, lbl, K)
        out = [row[-1] for row in neighbors]
        return max(set(out), key=out.count)


for i in range(len(test_data)):
        p = predict(train_data, test_data[i], train_lbl, 1)
        print("test_lbl:" , colored(test_lbl[i], 'green')  ,"prediction: " , colored(p, 'green') )
```

נקבל את הפלט הבא:

```
test_lbl: [1.] prediction:  1.0
test_lbl: [2.] prediction:  2.0
test_lbl: [3.] prediction:  3.0
test_lbl: [1.] prediction:  1.0
test_lbl: [3.] prediction:  3.0
test_lbl: [1.] prediction:  1.0
test_lbl: [2.] prediction:  2.0
test_lbl: [1.] prediction:  1.0
test_lbl: [3.] prediction:  3.0
test_lbl: [1.] prediction:  1.0
```

קיבלנו התאמה מלאה בין מערך התגיות של הפרחים שבדקנו לבין פלט האלגוריתם עבור כל אחד מהם.

קיימות שיטות מתקדמות יותר לבחינת הביצועים של מסווג כגון שיטת cross validation וכן מדדים מתקדמים יותר לבחינת הביצועים כגן confusion matrix. הסברים בדבר שיטות אלו אינם מפורטים בספר זה.

KNN הוא אלגוריתם עצלן לימודית כי בניגוד לאלגוריתמים נוספים שנלמד בהמשך, אלגוריתם KNN לומד ומסווג רק כאשר מתקבלת בקשה לסיווג. זהו אחד החסרונות של KNN. אלגוריתם זה יתקשה לתפקד כאשר מדובר על כמויות גדולות של נתונים לאימון (למידה). בקיצור עצלן! לומד תוך כדי הבחינה!?

פעילות 7 - רגרסיה לינארית בסביבת Python

לפי ההגדרה רגרסיה ליניארית היא שיטה מתמטית למציאת הפרמטרים של הקשר בין משתנה בלתי תלוי X למשתנה תלוי Y, בהנחה שהקשר ביניהם ליניארי, כלומר מהצורה $y = mx + b$. (מקור: ויקיפדיה)

בפעילות זו נלמד ונתרגל את היסודות המתמטיים של רגרסיה לינארית ויישומיה בשפת python. בפעילויות ההמשך נעשה שימוש בעקרונות הרגרסיה הליניארית כדי לכוון את המשקלים של רשתות נוירונים שנפתח בעתיד.

בפעילות זו נתמקד ברגרסיה לינארית בסיסית כדי למצוא את ערכי m ו- b של משוואת קו ישר:

$$y = mx + b$$

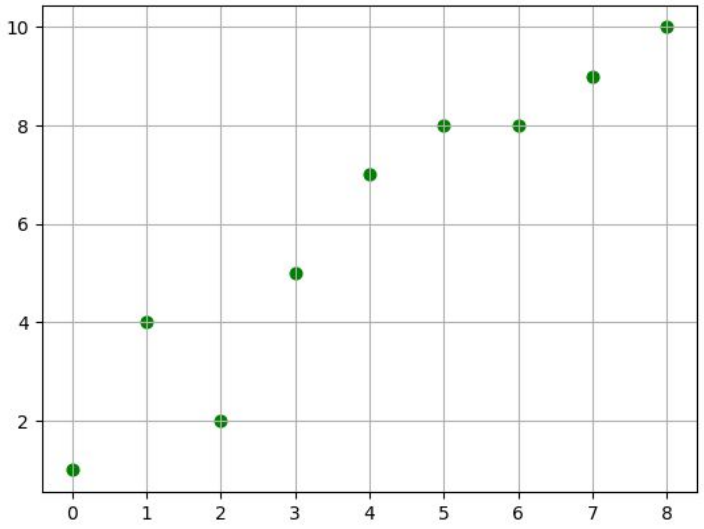
כלומר אנו מנסים למצוא פונקציה לינארית שמנבאת את ערך של y בצורה מדויקת ככל האפשר כפונקציה של משתנה בלתי תלוי x.

כדי להבין את הנושא נגדיר 2 רשימות נתונים: האחת מייצגת את הערכים של x והשנייה את הערכים של y. בפעילות זו ננתח את 2 הרשימות כדי ליצור מהם את משוואת הקו הישר המייצגת באופן אופטימלי את אוסף הנקודות שהגדרנו (כלומר כל 2 ערכים x ו- y מייצגים נקודה על מערכת צירים דו-מימדית).

נדגים זאת בקוד הבא:

```
import numpy as np

import matplotlib.pyplot as plt


x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])

y = np.array([1, 4, 2, 5, 7, 8, 8, 9, 10])


plt.scatter(x, y, color = "g", marker = "o", s = 40)

plt.grid()

plt.show()
```

נקבל את גרף הנקודות הבא:



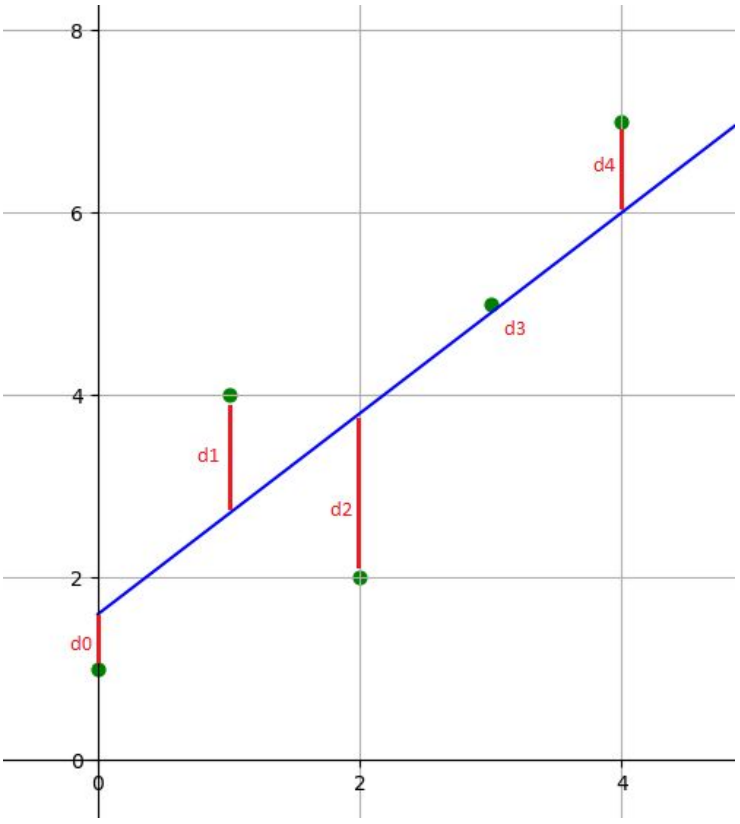
הגדרנו בקוד 2 וקטורים האחד בשם x השווה ל:

$$x = [x1, x2, x2.....xn]$$

והשני וקטור בשם y שווה ל:

$$y = [y1, y2, y3.....yn]$$

כדי למצוא את קו המגמה המתאים ביותר למערך הנקודות נעזר בכלי המתמטי Ordinary least squares המאפשר למצוא פרמטרים המביאים למינימום את סכום הפרשי המרחקים בריבוע. נדגים זאת:



נציג את שגיאת המודל המחושבת על ידי סכום ההפרשים בין הניבוי לבין הערך האמיתי בריבוע בעזרת הנוסחה הבא:

$$(d0)^2 + (d1)^2 + (d2)^2 + (d3)^2 + ... + (dn)^2$$

המטרה היא למצוא קו מגמה שבו שגיאת המודל תהייה קטנה ביותר האפשרית.

כדי להגיע לערך זה עלינו לדעת מה יהיה הערך של m ו- b (כלומר השיפוע של הקו וערך נקודת החיתוך שלו עם ציר y).

כדי למצוא את ערכי m,b הנ"ל, ניתן לגזור את נוסחת השגיאה ולהשוות לאפס את הנגזרת. התוצאה המתקבלת מוצגת בנוסחאות הבאות:

$$m = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\overline{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

נפרש את 3 הנוסחאות הבאות:

- הסימן $\sum_{i=1}^{n}$ מציין שמדובר בסכום כל האיברים החל מ-1 עד n.
- הסימנים $\overline{x}$ ו- $\overline{y}$ מציינים שמדובר בממוצע כל האיברים החל מ-1 עד n.

כדי לחשב את הערך של b נשתמש בנוסחה הבא:

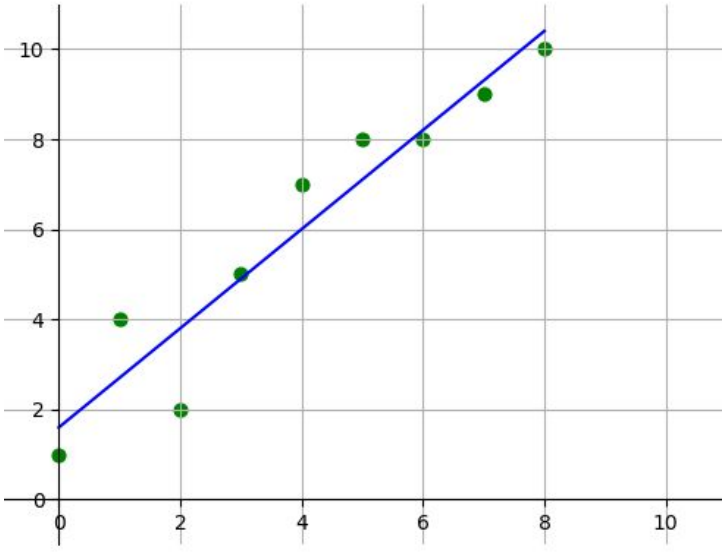
$$b = \overline{y} - m \cdot \overline{x}$$

נכתב את קוד התוכנית למציאת קו הרגרסיה הלינארית המתאימה ביותר למערך נתונים הכולל נקודות:

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

ax = plt.gca()
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
plt.xlim([-1, 11])
plt.ylim([-1, 11])
plt.grid()

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
y = np.array([1, 4, 2, 5, 7, 8, 8, 9, 10])


x_mean= np.mean(x)

y_mean= np.mean(y)

m = (np.sum((x_mean)*(y-y_mean)))/(np.sum((x_mean)*(x_mean)))

b = y_mean - m*x_mean


print("m = ",m," b = ",b)

x_line = x

y_line = m*x + b

plt.plot(x_line, y_line, color = "b")

plt.scatter(x, y, color = "g", marker = "o", s = 40)

plt.show()
```

נקבל את הפלט הבא:



שימו לב לאופן שבו אנו ממירים את המשוואה הבאה לקוד בשפת Python:

$$m = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}$$

```
x_mean = np.mean(x)

y_mean = np.mean(y)

m = (np.sum((x-x_mean)*(y-y_mean)))/(np.sum((x-x_mean)*(x-x_mean)))
```

הפעולה np.mean מחזירה את ממוצע כל האיברים שמקבלים כפרמטר לפעולה. כלומר המשתנים x_mean ו- y_mean מייצגים בקוד את המשתנים $\bar{x}$ ו- $\bar{y}$ בהתאמה.

הפעולה np.sum מחזירה את סכום כל האיברים שמקבלים כפרמטר לפעולה.

על פי אותו עיקרון הפונקציה $b = \bar{y} - m \cdot \bar{x}$ תמומש בקוד באופן הבא:

```
b = y_mean - m*x_mean
```

משימה: יישום רגרסיה לינארית לצורך חיזוי מספרי לידות בקרב בני נוער בגילאי 15 עד 17.

במשימה זו נשתמש בעקרונות הרגרסיה הלינארית כפי שלמדנו כדי לשער את מספר הלידות בקרב בני נוער בגילאי 15 עד 17 על פי מדד העוני באזור המגורים שלהם. לצורך התרגיל נשתמש במערך נתונים הכולל 51 דגימות נתונים אמיתיים שנאספו בשנת 2002 בארה"ב. להלן קישור למקור הנתונים:

https://online.stat.psu.edu/stat462/sites/onlinecourses.science.psu.edu.stat462/files/data/poverty/index.txt

Data source: Mind On Statistics, 3rd edition, Utts and Heckard

קובץ הנתונים כולל 2 עמודות הראשונה שיעור לידות בקרב בנות 16 עד 17 לכל 1000 לידות. העמודה השנייה היא אחוז משקי הבית באותה מדינה שרמת ההכנסה שלהם מתחת לרמת העוני כפי שהגדיר הממשל הפדרלי בארה"ב. להלן דוגמה לחלק מהנתונים:

שם מדינה	מספר לידות בקרב בנות בגילאי 16 עד 17 לכל 1000 לידות	אחוז משקי הבית שרמת ההכנסה שלהם היא מתחת לרמת העוני
Washington	16.8	12.5
Alaska	73.7	18.9
New_York	15.7	16.5
Arkansas	31.6	14.9
California	22.6	16.7
Texas	104.3	38.2
Mississippi	37.6	23.5

כלומר המטרה במשימה זו היא למצוא פונקציה מתמטית המסוגלת לבצע את הקירוב הטוב ביותר לנתוני הטבלה. מציאת הפונקציה המתמטית תאפשר לנו לבצע ניבוי על מקרים חדשים שאין לנו נתונים עבורם בטבלה. אמינות הניבוי תלוי באמינות מדגם הנתונים וגודלו.

עקרון זה הוא בבסיסו של כל מכונה לומדת! כלומר ההכללות הללו הם למעשה הדרך שבה המחשב לומד מדוגמאות שנותנים לו. בהמשך המדריך נגלה שגם מילים ותמונות מיוצגים במחשב בעזרת מספרים כך שלמעשה העיקרון הזה נכון גם במקרים שרוצים לסווג משפטים ותמונות.

העובדה שיש קשר לינארי בין שיעור העוני לבין שיעור לידות בקרב נערות אינו מצביע על כך שיש סיבתיות בין שני נתונים אלו, אלא רק על כך שיש קשר ביניהם.

לצורך מימוש התרגיל נבצע את המשימות הבאות:

- בשלב הראשון נקלוט את נתוני הטבלה לתוך מערך numpy.
- נפעיל את האלגוריתם למציאת קו המגמה הלינארי.
- נבחן על ידי גרף שאכן קיים קירוב בין נתוני המקור לבין קו המגמה.
- נבצע חיזוי לנתון שלא קיים בטבלה.

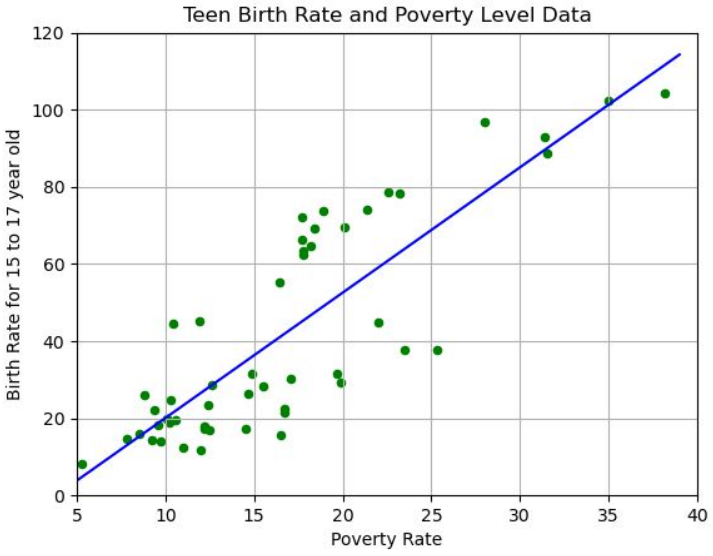
להלן מימוש הקוד:

```
import numpy as np

import matplotlib.pyplot as plt


# Preparing the data for machine learned
```

```
dataset = np.loadtxt("lab6\data.csv", delimiter=',')

x = dataset[:,0]

y = dataset[:,1]


#Learning

def LinearRegression(x,y):

    avgx = np.mean(x)

    avgy = np.mean(y)

    m = (np.sum((x-avgx)*(y-avgy)))/(np.sum((x-avgx)*(x-avgx)))

    b = avgy - m*avgx

    return m,b

m,b =LinearRegression(x,y)


#Generate the graph

y_line = m*x + b

plt.plot(x, y_line, color = "b")

plt.scatter(x, y, color = "g", marker = "o", s = 20)

plt.title('Teen Birth Rate and Poverty Level Data')

plt.xlabel('Poverty Rate')

plt.ylabel('Birth Rate for 15 to 17 year old')

plt.xlim([5, 40])

plt.ylim([0, 120])

plt.grid()

plt.show()
```

```
#Predict data
pov=float(input('Enter poverty rate: '))
brth15to17 = m*pov + b
print('Birth Rate for 15 to 17 year old is:', int(brth15to17))
```

נקבל את הפלט הבא:




```
Enter poverty rate: 15
Birth Rate for 15 to 17 year old is: 36
```

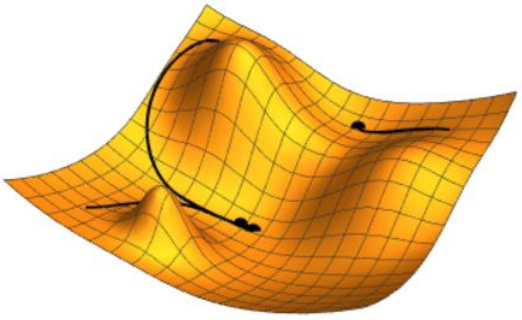
בפעילות הבאה נלמד את אלגוריתם Gradient Descent ויישומו בשפת python. הנושאים רגרסיה ליניארית שלמדנו לממש בפעילות זו יחד עם Gradient Descent הם חלק מרכזי ביכולת ההבנה שלנו את נושא התכנות של רשתות נוירונים מלאכותיות ANN - Artificial Neural Network שנלמד בהמשך מדריך זה.

פעילות 8 - אלגוריתם Gradient Descent

בפעילות זו נלמד את היסודות המתמטיים של Gradient descent ויישומו בשפת python. נושא זה יחד עם רגרסיה ליניארית הם חלק מרכזי בעבודה עם רשתות נוירונים מלאכותיות שנפתח בהמשך מדריך זה.

Gradient descent היא שיטה למציאת נקודות מינימום ומקסימום מקומיים תוך כדי ביצוע סדרה של קירובים ההולכים ומתכנסים לנקודות שאנו מחפשים.

כדי להבין את הרעיון שבבסיס Gradient descent נדמיין שאותנו עומדים בנקודה כלשהי על הר באמצע הלילה ומנסים לפלס את דרכנו למטה כאשר אין לנו יכולת לראות את הסביבה. הדרך היחידה שלנו לחוש את הסביבה היא על ידי התחושה ברגליים כלומר לרגיש בשרירי הרגליים האם אנו עולים לנקודה גבוהה יותר או יורדים לנקודה נמוכה יותר מהמקום שאנו נמצאים ברגע זה. מציאת הדרך למטה מההר תתבצע על ידי החלטה אקראית להזיז את הרגליים כאשר בכל פעם נרגיש האם אנו עולים או יורדים. נחזור על הפעולה הזו מספר רב של פעמים כאשר בכל פעם נבדוק מחדש האם אנו עולים או יורדים ונזוז בצעד אחד לכיוון המקום הנמוך יותר. בסוף תהליך הכולל מספר רב של ניסיונות אנו צפויים להיות במקום הנמוך ביותר שמצאנו כלומר הצלחנו לרדת מההר.

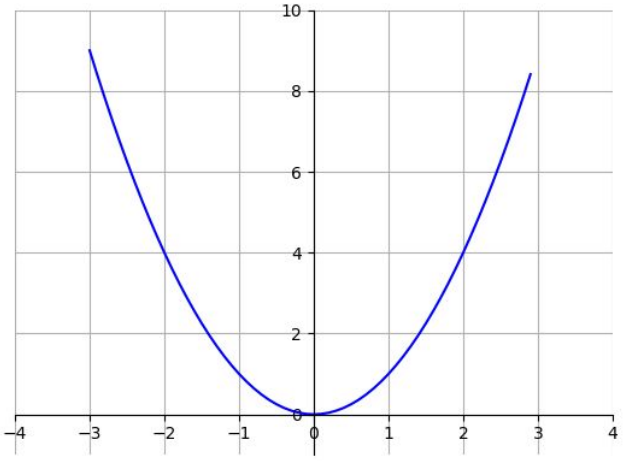


מקור התמונה: https://upload.wikimedia.org/wikipedia/commons/a/a3/Gradient_descent.gif

נתרגם את אנלוגיות הירידה מההר למושגים בתחום למידת מכונה:

- הר הכולל שיפועים - פונקציית שגיאה של אלגוריתם למידת מכונה
- מציאת הכיוון עליה או ירידה - שיפוע הפונקציה.
- הצעד שעושה האיש בכל פעם - קצב הלמידה של המכונה: LearningRate
- מספר רב של צעדים - משתנה Epochs

נדגים את פעולת gradient descent על פולינום ממעלה שניה. פונקציה זו שימושית מאוד למכונות לומדות מכיוון שכפי שנראה בהמשך, פונקציית השגיאה של מכונה לומדת היא פולינום ממעלה שניה. גרף פונקציה כזו נראה באופן הבא:



נגזרת של פונקציה פולינומית

אחת השיטות לחפש את נקודת המינימום היא לחשב את הנגזרת הפונקציה. נדגים את הרעיון בתרגיל הבא.

נתונה הפונקציה הבאה:

$$y = 2x^2 + 4x + 3$$

נגזרת של הפונקציה תכתב כך:

$$y' = 2 \cdot 2x^1 + 1 \cdot 4x^0 + 0$$

$$y' = 4x + 4$$

באופן כללי נגזרת של פונקציה פולינומית מחושבת כך (Power Rule)

$$f_{(x)} = x^n$$

$$f'_{(x)} = nx^{n-1}$$

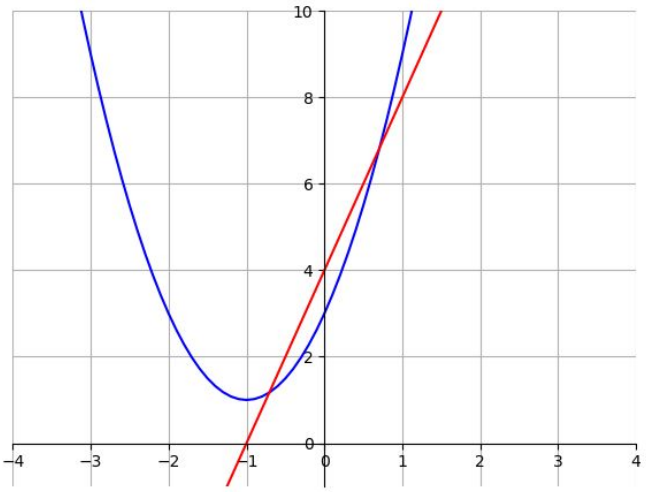
נכתב קוד בשפת python כדי לראות את 2 הפונקציות (הפונקציה y ו- y') להלן קוד התוכנית:

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


ax = plt.gca()
ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')
plt.xlim([-4, 4])
plt.ylim([-10, 10])
plt.grid()


x = np.arange(-4.0, 3.0, 0.1)
y= 2*x**2 + 4*x + 3
y_tag = 4*x+4
plt.plot(x, y, color = "b")
plt.plot(x, y_tag, color = "r")
plt.show()
```

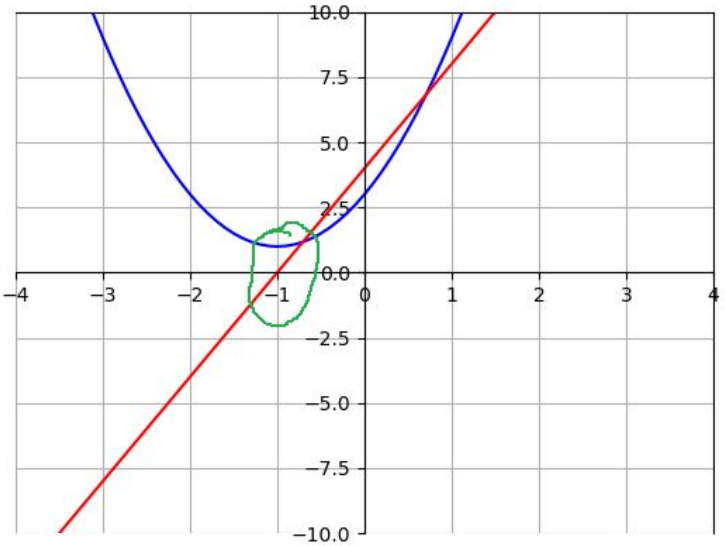
שימו לב לאופן כתיבת משוואת פונקציה בשפת python:

```
y= 2*x**2 + 4*x + 3
y2 = 4*x+4
```

נקבל את הפלט הבא:



מהגרף ניתן לראות את הפונקציה y בצבע כחול והנגזרת שלה $y^{\prime}$ בצבע אדום. הקו הישר מתאר את אוסף השיפועים של הפונקציה y זאת אומרת שנקודת החיתוך של הנגזרת עם ציר X היא נקודת המינימום של הפונקציה y כלומר נקודת המינימום היא מינוס אחד.



חקירת פונקצית הנגזרת מלמדת אותנו שנקודת המינימום היא נמצאת בנקודה שבה y=0 כלומר x=-1. אך חשוב מכך, מצאנו דרך יעילה לבדוק את שיפוע פונקציית המחיר על ידי הנגזרת. ניתן לראות שכאשר x בפונקציית הנגזרת שלילי הפונקציה יודרת כלומר מתקרבת לנקודת המינימום וכאשר x חיובי הפונקציה עולה כלומר מתרחקת מנקודת המינימום. עקרון זה הוא הבסיס שעליו מבוסס Gradient descent.

חקירת פונקציה פולינומית היא חלק מתוכנית הלימודים ברמה של 4 ו- 5 יח"ל במתמטיקה. מכאן שלפי הערכה שלי חשוב ללמד עקרון במטרה להבין כיצד נגזרת של פונקציה מסוגלת לעזור להגיע לנקודת המינימום של פונקציית המחיר COST.

לתרגול הנושא מומלץ לבקר באתר הבא: https://www.derivative-calculator.net

אלגוריתם gradient descent

ניישם את רעיון gradient descent למציאת נקודת המינימום של פונקציה. נרצה עתה למצוא את נקודת המינימום של הפונקציה y על ידי הגרלה של x התחלתי וביצוע איטרציות בהן נגדיל את נקטין את ערכו של x בהתאם לכיוון הנגזרת. אם הנגזרת חיובית סימן שאנחנו מעל נקודת המינימום ולכן נקטין את x. אם הנגזרת שלילית סימן שאנחנו מתחת לנקודת המינימום ולכן נגדיל את x.

באופן כללי משוואת העדכון של x היא:

```
x_new = x - (y') * learning rate
```

נדגים זאת בקוד הבא:

```
learning_rate=0.01
epochs=1000
x=0                          # initialize x as 0
for _ in range(epochs):      # learning iterations loop
  y_tag = 4*x+4
  x = x - y_tag * learning_rate
print ("found minimum at:",x)
```

נקבל את הפלט הבא:

```
found minimum at: -0.9999999999999987
```

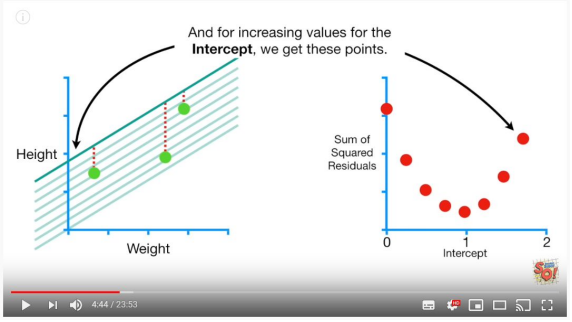
לסיכום

Gradient descent היא שיטה למציאת נקודות מינימום ומקסימום מקומיים תוך כדי ביצוע סדרה של קירובים ההולכים ומתכנסים לנקודות שאנו מחפשים.

בפעילות זו אנו מתרגלים את העקרונות המתמטיים למציאת מינימום בפונקציה פולינומית שבה קיימת נקודת מינימום אחת. אך בפועל מכונות לומדות צריכות לחפש נקודות מינימום מקומי בפונקציות מורכבות הרבה יותר. כמו כן לשם הנוכחות אנו מתייחסים לפונקציה שבה 2 מישורים (x ו- y) בלבד. בפועל מכונות לומדות צריכות לטפל בפונקציות הכוללות החל מ- 2, ו- 3 מימדים עד לעשרות ומאות מימדים. העקרונות המתמטיים שאנו לומדים כאן בהקשר ל- Gradient descent מתאימים גם למספר מימדים גדול יותר. אך חקירת פונקציות אלה חורג מגבולות מדריך זה.

ניתוח מתמטי מורחב ניתן למצוא בסרטון הבא:

https://www.youtube.com/watch?v=sDv4f4s2SB8&feature=youtu.be



צפייה בסרטון זה מגלה לנו את הקשר המתמטי בין איסוף השגיאות תוך כדי הזזת קו המגמה לבין פונקציה פולינומית.

פעילות 9 - יישום רגרסיה לינארית על ידי Gradient Descent

בפעילות זו נתבסס על העקרונות המתמטיים של Gradient descent כדי לממש רגרסיה לינארית. פעילות זו כמו גם הפעילות הקודמת בנושא רגרסיה ליניארית (פעילות 6) הם הבסיס האלגוריתמי לעבודה עם רשתות נוירונים שנפתח בהמשך מדריך זה.

Gradient descent היא שיטה למציאת נקודות מינימום ומקסימום מקומיים תוך כדי ביצוע סדרה של קירובים ההולכים ומתכנסים לנקודות שאנו מחפשים.

בפעילות זו כמו בפעילות הקודמת בנושא רגרסיה לינארית בסיסית נחפש את ערכי m ו- b של משוואת קו ישר:

$$y = mx + b$$

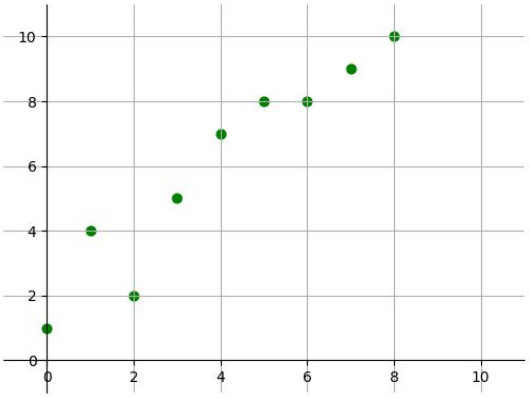
כלומר אנו מנסים למצוא פונקציה לינארית שמנבאת את הערך של y בצורה מדויקת ככל האפשר כפונקציה של המשתנה הבלתי תלוי x. הדוגמאות מהן נלמד נתונות בשתי רשימות נתונים, האחת מייצגת את הערכים של x והשנייה את הערכים של y.

נדגים זאת בקוד הבא:

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


ax = plt.gca()
ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')
plt.xlim([-1, 11])
plt.ylim([-1, 11])
plt.grid()


x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])          # given samples X values
y = np.array([1, 4, 2, 5, 7, 8, 8, 9, 10])         # given samples y values
```

```
plt.scatter(x, y, color = "g", marker = "o", s = 40)

plt.show()
```

נקבל את גרף הנקודות הבא:



כלומר הגדרנו בקוד 2 וקטורים האחד בשם x השווה ל:

$$x = [x1, x2, x2.....xn]$$

והשני y שווה ל:

$$y = [y1, y2, y3.....yn]$$

באופן כללי במערכות של מכונה לומדת נרצה לנבא את ערך של y לפי ערך ידוע של x. אך קודם לכך נרצה לכוון, כלומר לאמן, את המכונה להגיע לרמת דיוק גבוהה על ידי מציאת ערכים ל-m ו-b. לשם כך נגדיר בהתחלה ערכים אקראיים, או אפסים ל-m ו-b ואז ניקח נקודה x כלשהי שקיבלנו נציב בנוסחה ונקבל ערך של y לפי המשוואה. מכאן שיהיו לנו בשלב זה 2 ערכים של y האחד אמיתי (הפלט הרצוי שאנו מצפים שהמכונה תפלוט) והשני הניחוש guess (הערך שהמכונה פלטה על בסיס מספר אקראי שהיא קיבלה במבוא תוך כדי שלב הלמידה).

נגדיר את error כהפרש בין שני ערכי ה- y עבור נקודה כלשהי x_i.

$$error_i = guess_i - y_i$$

$$guess_i = mx_i + b$$

נגדיר פונקציית מחיר באופן הבא:

$$COST = \sum_{i=1}^{n}(guess_i - y_i)^2$$

בהתבסס על האנלוגיה שלנו לגבי הירידה מההר הערך של COST הוא הגובה הנוכחי של האדם על ההר מכאן שהמטרה שלנו היא להקטין ככל שניתן את COST כדי להגיע לתחתית ההר על פי עולם המושגים של האנלוגיה שלנו. על פי עקרונות Gradient descent אנו צריכים להקטין את הערך של COST כדי להגיע לערכי m ו- b של קו ישר היוצרים את השגיאה הקטנה ביותר.

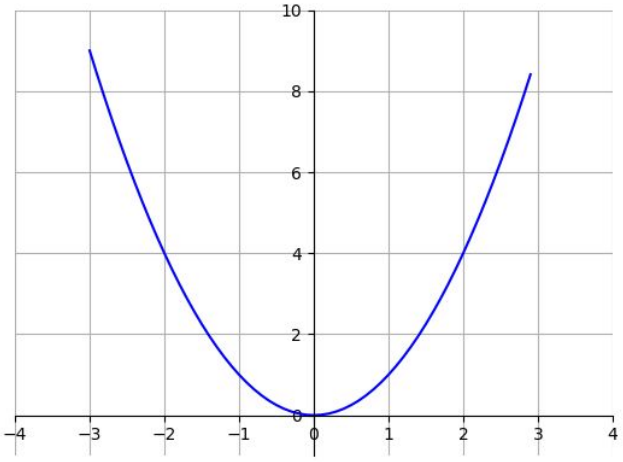
נבחן את המשוואה שכתבנו ונקבל שהיא לא רחוקה מהפונקציה הפולינומית הבא:

$$y = x^2$$

או במושגים שלנו אנו צריכים לחפש את הערך של b המספק לנו את הערך הנמוך ביותר של COST. כלומר לחפש את נקודת המינימום בגרף ובמקביל לחפש את הערך של m המספק לנו את הערך הנמוך ביותר של COST. כלומר לחפש את נקודת המינימום בשני הפולינומים הבאים:

$$COST = b^2 \qquad COST = m^2$$

פונקציית השגיאה היא פולינום ממעלה שניה. גרף פונקציה כזו נראה באופן הבא:



מכאן שעלינו לחפש את הערך של m,b המספק לנו את הערך הנמוך ביותר של cost. כלומר לחפש את נקודת המינימום בגרף.

חקירת Cost Function תוך שימוש בנגזרות

עד כה פגשנו כבר את 2 המשוואות הבאות:

$$error = guess_i - y$$

$$COST = \sum_{i=1}^{n}(guess_i - y_i)^2$$

נסמן,

$$error = guess_i - y_i = (mx_i + b) - y_i$$

כמו כן נסמן איבר בודד של פונקציית ה- COST ב- L בצורה הבאה:

$$L_{(m,b)} = error^2$$

מחפשים את הערכים האופטימליים של משוואת הקו הישר m ו- b שייתנו את השגיאה הקטנה ביותר.

$$L_{(m,b)} = error^2 = (mx_i + b - y_i)^2$$

נעשה זאת על ידי כך שנשנה בקצת את הערכים m ו- b עד לקבלת השגיאה המזערית. נתאר זאת בנוסחה:

$$m = m + \triangle m$$

$$b = b + \triangle b$$

מכאן שאנו צריכים למצוא את השיפוע בכל אחד מ-2 הצירים (m ו- b). חישוב זה מכונה נגזרת חלקית. נעשה זאת על ידי שימוש בנגזרת שבה בכל פעם נבדוק שיפוע על ציר אחד. מכאן שנגזור את הפונקציה שלנו כך שבפעם הראשונה נחשב את הנגזרת של J לפי m כאשר b קבוע ובפעם השנייה נחשב את הנגזרת של J לפי b כאשר m קבוע. אך תחילה יש להבין כיצד גוזרים את הפונקציה הבא:

$$L_{(m,b)} = (mx_i + b - y_i)^2$$

נגזרת של פונקציה מורכבת

כדי לגזור פונקציה מסוג $(mx_i + b - y_i)^2$ נשתמש בכלל השרשרת (Chain rule) המאפשר לנו למצוא את הנגזרות של פונקציות מורכבת.

נדגים את כלל השרשרת תוך כדי גזירה של הפונקציה הבאה:

$$y = (x^2 + 8)^3$$

$$y' = 3(x^2 + 8)^2 \cdot 2x$$

על פי כלל השרשרת בשלב הראשון אנו גוזרים את הפונקציה $(x^2 + 8)^3$ תוך שימוש נגזרת של פולינום שלמדנו. באופן זה קיבלנו את הנגזרת $3(x^2 + 8)^2$. אך זה לא מספיק כי אז אנו צריכים להכפיל את התוצאה בנגזרת הפנימית $x^2 + 8$ שאז נקבל $2x$.

נחזור לפונקציה שלנו $L_{(m,b)} = (mx_i + b - y_i)^2$ וניממש עליה את כלל השרשרת. אך גם כאן עולה קושי נוסף. יש לנו 2 פרמטרים שעל פיהם אנו נדרשים לגזור כדי להגיע לערכי m ו- b של קו ישר היוצרים את השגיאה הקטנה ביותר.

נגזרת של L לפי m

ראינו כבר ש:

$$error = guess_i - y_i = mx + b - y$$

מכאן ניתן לתאר את הנגזרת של L לפי m כך:

$$L_{(m)} = (error)^2$$

$$L'_{(m)} = 2(error)^1 \cdot error'$$

נגזרת של $error$ לפי m תראה כך:

$$error = mx + b - y$$

$$error'_{(m)} = 1 \cdot x \cdot m^0 + 0 - 0 = x$$

מכאן ש:

$$L'_{(m)} = 2 \cdot error \cdot x$$

בפועל כשנעבוד עם Gradient descent נרצה לשנות את התוצאה בקצת לכן נכפיל את התוצאה ב- learning rate כלשהו.

$$L'_{(m)} = (2 \cdot error \cdot x) \cdot Learning\ Rate$$

בגלל שכפלנו את התוצאה בערך קבוע כמו learning rate כבר אין משמעות לכפל ב- 2 ואז ניתן לסכם את כל מה שחקרנו עד כה במשוואה הבא:

$$m = m - \triangle m$$

$$L'_{(m)} = error \cdot x \cdot Learning\ Rate$$

$$\boxed{m = m - error \cdot x \cdot Learning\ Rate}$$

נגזרת של L לפי b

ניתן לתאר את הנגזרת של L לפי b כך:

$$L_{(b)} = (error)^2$$

$$L'_{(b)} = 2(error)^1 \cdot error'$$

נגזרת של $error$ לפי b תראה כך:

$$error = mx + b - y$$

$$error'_{(b)} = 0 + 1 \cdot b^0 - 0 = 1$$

מכאן ש:

$$L'_{(b)} = 2 \cdot error \cdot 1$$

$$L'_{(b)} = (2 \cdot error \cdot 1) \cdot Learning\ Rate$$

$$b = b - \Delta b$$

$$L'_{(b)} = error \cdot x \cdot Learning\ Rate$$

$$b = b - (2 \cdot error \cdot 1) \cdot Learning\ Rate$$

$$b = b - error \cdot Learning\ Rate$$

נשתמש ב-2 הנוסחאות שנלמדו כדי לממש פעולה בשפת python למציאת הערכים של m ו- b תוך שימוש בעקרונות Gradient Descent. נממש פעולה בשם Gradient Descent המקבלת את הפרמטרים הבאים:

- מערך נתונים בשם x המכיל אוסף ערכים של וקטור x.
- מערך נתונים בשם y המכיל אוסף ערכים של וקטור y.
- משתנה בשם learning_rate המכיל ערך שיקבע מה יהיה קצב השינוי של המקדמים m,b תוך כדי ביצוע חישובי הקירוב לערכים שאנו מחפשים. בפעולה זו נגדיר כברירת מחדל את הערך 0.01
- משתנה בשם epochs המכיל ערך שיקבע כמה איטרציות נעשה תוך כדי חיפוש הערכים שלנו. בפעולה זו נגדיר כברירת מחדל את הערך 1000

הפעולה תחזיר לנו 2 ערכים m ו- b המייצגים קירוב בשיטת Gradient Descent עבור משוואת הקו שאנו מחפשים.

להלן מימוש הפעולה:

```
def GradientDescent(x,y,learning_rate=0.01, epochs=1000):
    m=0                                 # initialize m as 0
    b=0 # initialize b as 0
    for _ in range(epochs):             # learning iterations loop
        for i in range(len(x)):         # loop over all data in the dataset
            xi = x[i]
            yi = y[i]
            y_hat = m * xi + b          # calculate the estimated value
            d= y_hat - yi               # calculate the derivative
            m = m - d * xi * learning_rate
            b = b - d * learning_rate
    return m,b
```

הפעולה מבוססת על לולאה מקוננת העוברת על כל אחד מערכי הווקטורים כמספר הפעמים שהוגדר במשתנה epochs. בכל מפגש נקודה על הוקטור הפעולה תבצע את ההוראות הבאות:

```
y_hat = m * xi + b
d = y_hat - yi
m = m - (d * xi) * learning_rate
b = b - d * learning_rate
```

הפעולה תחשב במשתנה y_hat את פונקציית המטרה שלנו כלומר משוואת הקו הישר שאנו מחפשים, בהמשך הפעולה תחשב במשתנה d את הנגזרת הנוכחית ואז תכוון את ערכי m ו- b לערכים חדשים על פי קצב השינוי שנקבע במשתנה learning_rate.

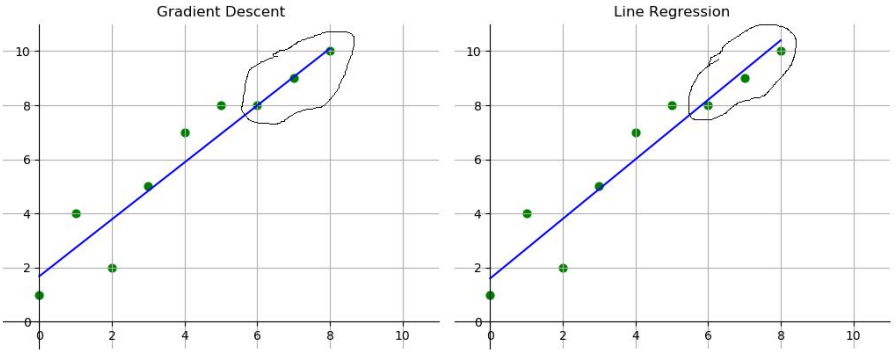
להלן קוד לבדיקת הפעולה שכתבנו:

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

```
def GradientDescent(x,y,learning_rate=0.01, epochs=1000):
    m=0                         # initialize m as 0
    b=0                         # initialize b as 0
    for _ in range(epochs):     # learning iterations loop
        for i in range(len(x)):     # loop over all data in the dataset
            xi = x[i]
            yi = y[i]
            y_hat = m * xi + b      # calculate the estimated value
            d = y_hat - yi          # calculate the derivative
            m = m - d * xi * learning_rate
            b = b - d * learning_rate
    return m,b

ax = plt.gca()
ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')
plt.xlim([-1, 11])
plt.ylim([-1, 11])
plt.grid()

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([1, 4, 2, 5, 7, 8, 8, 9, 10])
m,b=GradientDescent(x,y)
```

```
print("\nm = ",m," b = ",b)

x_line = x

y_line = m*x + b

plt.plot(x_line, y_line, color = "b")

plt.scatter(x, y, color = "g", marker = "o", s = 40)

plt.show()
```

נבחן זה לצד זה את 2 קווי המגמה שקיבלנו. בצד שמאל את הקו המתקבל על ידי הפעולה Gradient Descent ובצד ימין את קו המגמה שקיבלנו בפעילות 6 כאשר חיפשנו קו מגמה תוך שימוש ב- Line Regression.



ניתן לראות שקיבלנו קווי מגמה דומים מאוד אחד לשני. ניתן לזהות הבדלים קטנים בעיקר ב-3 הנקודות העליונות שבגרף.

נבחן כעת את התנהגות הפעולה Gradient Descent עבור ערכים שונים של learning_rate שמייצגים את קצב השינוי תוך כדי ביצוע חישובי הקירוב ושל משתנה epochs הקובע את מספר הקירובים שנעשה.

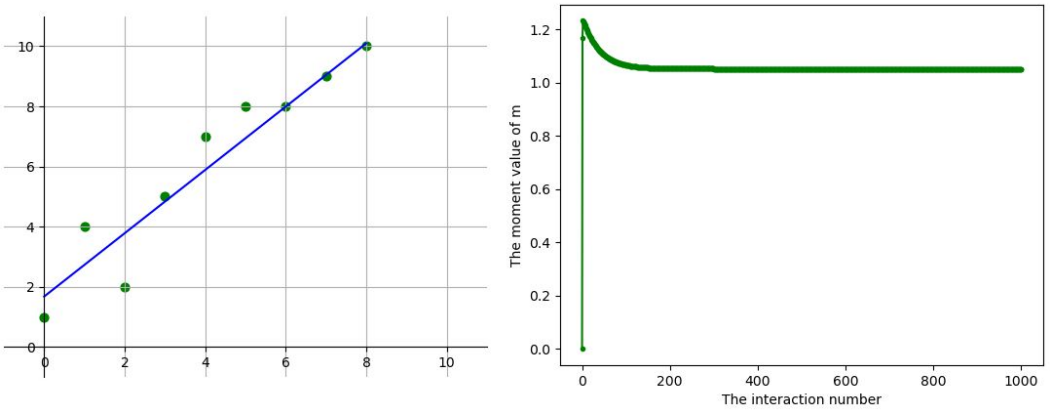
נתחיל בהערכת מספר האיטרציות הדרוש epoch. להלן קוד תוכנית שיעזור לנו בביצוע המשימה:

```
import numpy as np

import matplotlib.pyplot as plt

def GradientDescent(x,y,learning_rate=0.001, epochs=1000):

    m=0                    # initialize m as 0

    b=0                    # initialize b as 0
```

```
    for _ in range(epochs):     # learning iterations loop
        for i in range(len(x)):     # loop over all data in the dataset
            xi = x[i]
            yi = y[i]
            y_hat = m * xi + b     # calculate the estimated value
            d = y_hat - yi          # calculate the derivative
            m = m - d * xi * learning_rate
            b = b - d * learning_rate
        all_m = np.append(all_m,m)
    return m,b,all_m


ax = plt.gca()
ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')
plt.xlim([-1, 11])
plt.ylim([-1, 11])
plt.grid()
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([1, 4, 2, 5, 7, 8, 8, 9, 10])
m,b,all_m=GradientDescent(x,y)
print("\nm = ",m," b = ",b)
x_line = x
y_line = m*x + b
plt.plot(x_line, y_line, color = "b")
```

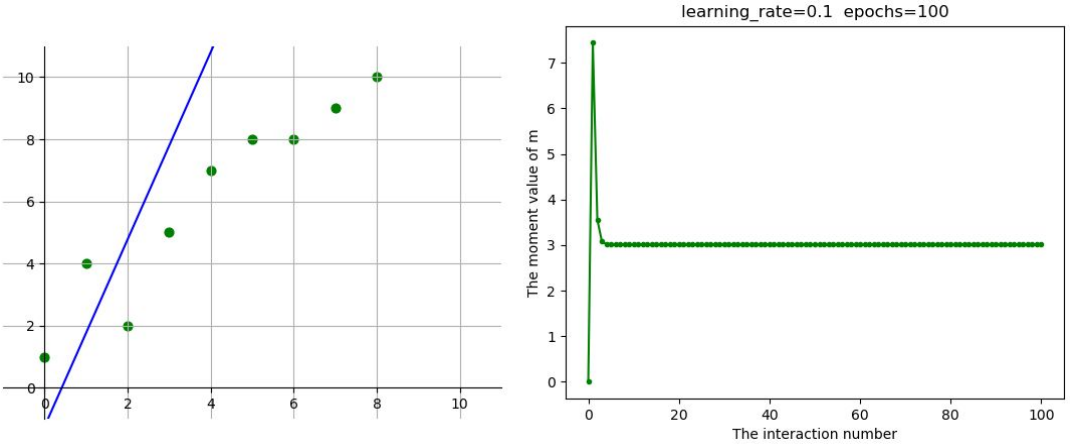
```
plt.scatter(x, y, color = "g", marker = "o", s = 40)

plt.show()

plt.plot(all_m, color = "g", marker = ".")

plt.xlabel('x')

plt.ylabel('y')

plt.show()
```

פלט התוכנית יהיה 2 הגרפים הבאים:

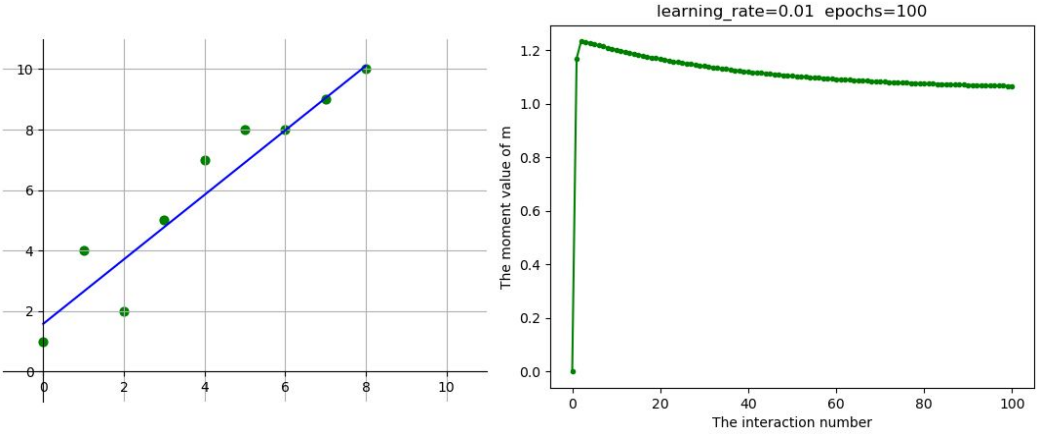


כבר כעת ניתן ללמוד שמספר האינטראקציות שקבענו ל- 1000 גדול מדי כי לאחר כ- 200 פעולות ערכו של m נשאר יציב וקובע.

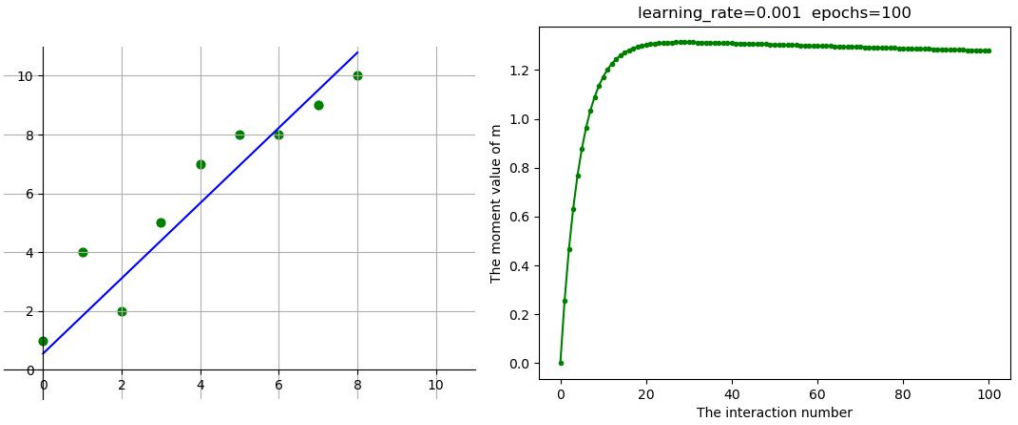
נבחן את השפעתו של המשנה learning_rate על משתנה m תוך כדי שינוי ערכו לערכים הבאים: 0.1 0.01 ו- 0.001. הרצה ראשונה learning_rate שווה ל-0.1:



הרצה שניה learning_rate שווה ל-0.01:



הרצה שלישית learning_rate שווה ל-0.001:



ניתן לראות את ההשפעה שיש ל- learning_rate על יכולת המערכת להגיע לערך מדויק. כאשר ערכו של המשתנה היה 0.1 קצב שינוי האינטראקציה לא אפשר למערכת להתקרב לתוצאה הרצוייה. הדבר דומה לנהג המנסה לעקוב אחר מסלול תוך כדי תנועות הגה חזקות וחדות.

לסיכום נממש זה לצד זה קוד למציאת קו מגמה תוך שימוש Line Regression וב- Gradient Descent כך שניתן יהיה להשוות בין 2 השיטות. התוכנה שלהלן מפיקה גרף המתאר את השגיאה בין הערכים שמחשבת כל שיטה.

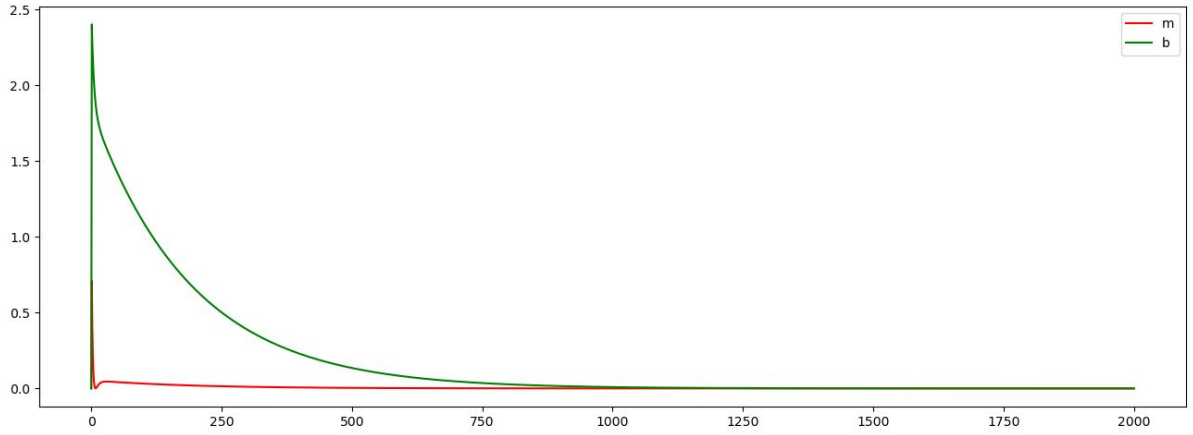
להלן קוד התוכנית:

```
import numpy as np
import matplotlib.pyplot as plt
def LineRegression(x,y):
    avgx = np.mean(x)
    avgy = np.mean(y)
    lin_m = (np.sum((x-avgx)*(y-avgy)))/(np.sum((x-avgx)*(x-avgx)))
    lin_b = avgy - lin_m*avgx
    return lin_m,lin_b


def GradientDescent(x,y,learning_rate=0.001, epochs=2000):
    m=0
    b=0
    err_m = np.array([0])
    err_b = np.array([0])
    lin_m,lin_b=LineRegression(x,y)
    for _ in range(epochs):                    # learning iterations loop
        for i in range(len(x)):                # loop over all data in the dataset
            xi = x[i]
            yi = y[i]
```

```
        y_hat = m * xi + b                  # calculate the estimated value
        d = y_hat - yi                      # calculate the derivative
        m = m - d * xi * learning_rate
        b = b - d * learning_rate
    err_m = np.append(err_m,pow(m-lin_m,2))
    err_b = np.append(err_b,pow(b-lin_b,2))
  return m,b,err_m,err_b

ax = plt.gca()
ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')
plt.xlim([-1, 11])
plt.ylim([-1, 11])
plt.grid()

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([1, 4, 2, 5, 7, 8, 8, 9, 10])
m,b,err_m,err_b=GradientDescent(x,y)

print("\nm = ",m," b = ",b)
x_line = x
y_line = m*x + b
plt.plot(x_line, y_line, color = "b")
plt.scatter(x, y, color = "g", marker = "o", s = 40)
```

```
plt.show()

plt.plot(err_m, '-r', label='m' )

plt.plot(err_b, '-g', label='b')

plt.legend(loc='upper right')

plt.show()
```

נקבל את הפלט הבא:



מור-טק מרכז המורים הארצי למקצועות הטכנולוגיים
הפקולטה לחינוך למדע וטכנולוגיה, קריית הטכניון, חיפה 32000 טל: 04-8293146
E-mail: Moretech@ed.technion.ac.il
http://moretech.technion.ac.il

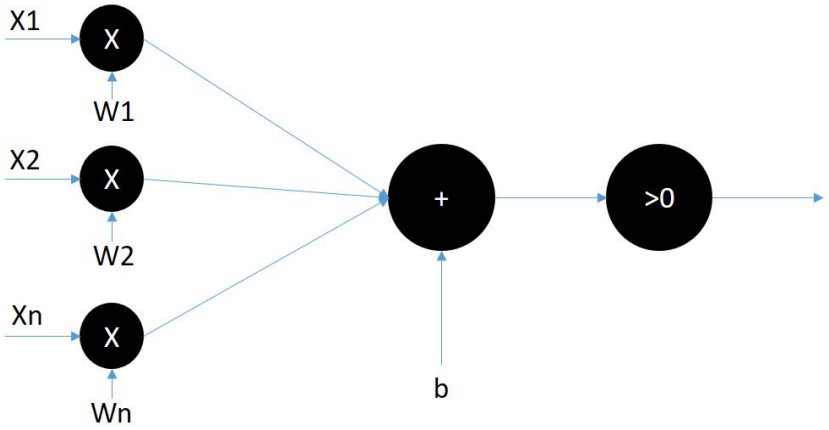
תכנות מכונה לומדת צעד אחר צעד מהיסוד

פעילות 10 - יישום פרספטרון בודד

פרספטרון Perceptron היא היחידה הבסיסית המרכיבה רשת נוירונים. בפעילות זו נלמד לתכנת מהיסוד (כלומר ללא שימוש בספריות קוד מוכנות) 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 מלבנים 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 bias 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) + bias
```

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 שקיימים אלגוריתמים נוספים דומים לפרספטרון שונים הכוללים פונקציות מעבר שונות המתאימות לצרכים שונים. בפעיליות ההמשך נלמד לעבוד עם פונקציות מעבר כדוגמת Sigmoid ו- TanH.

לפרטים נוספים ניתן לקבל בקישור הבא:

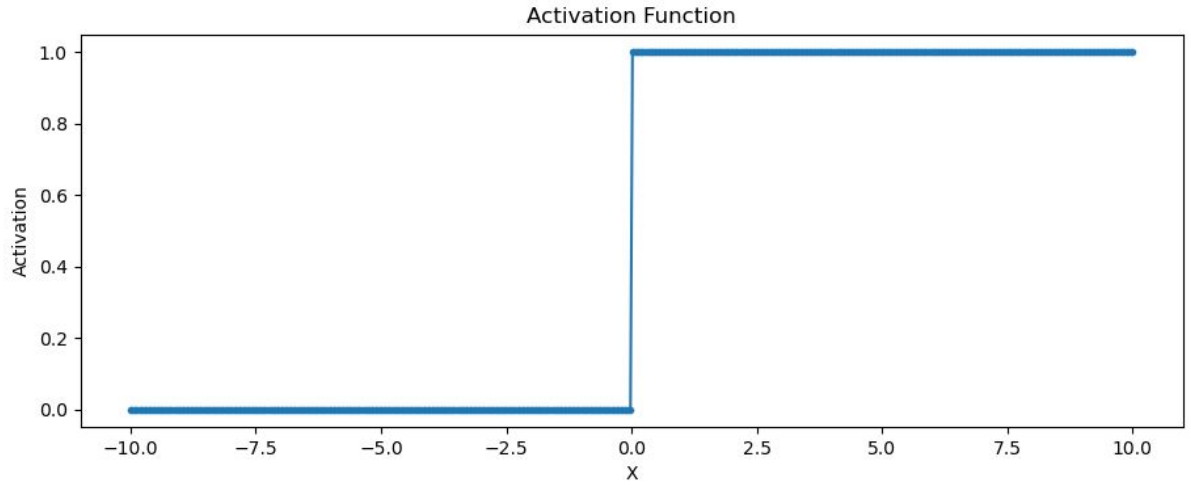
https://en.wikipedia.org/wiki/Activation_function

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, 500)
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("Activation")
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.bias = 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.bias
  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 כוללת 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 לנבא 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.bias += (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 הממומש לוגיקה זו:

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

קוד זה מציג כיצד ערכם של המשקלים weights ו- bias משתנה בהתאם להפרש בין הערך המופיע במוצא הפרספטרון prd לבין הערך הרצוי labels 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 פרספטרון 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.bias = 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.bias
    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.bias += (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))
```

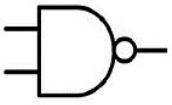
פלט התוכנית יהיה:

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])
```

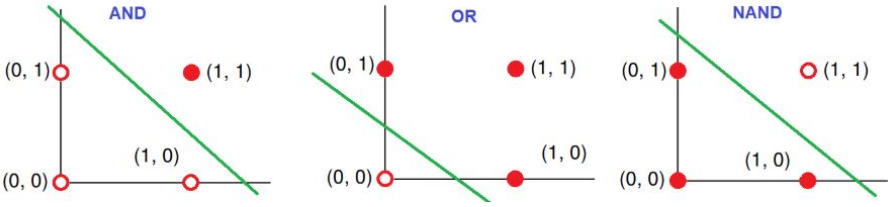
קיבלנו:

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

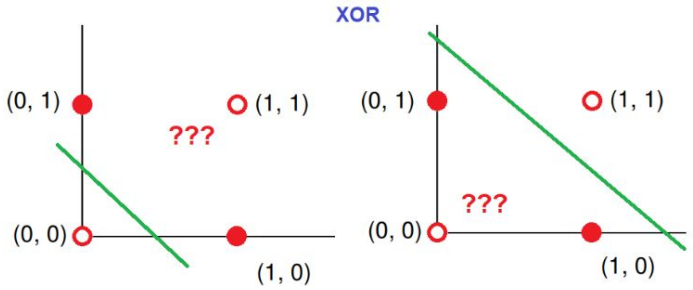
```
[0 0] 1
[0 1] 1
[1 0] 1
[1 1] 0
```

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.bias = 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.bias
    out = self.Activation(sum)
    return out


  def train(self, inputs, labels):
    for _ in range(self.epochs):
```

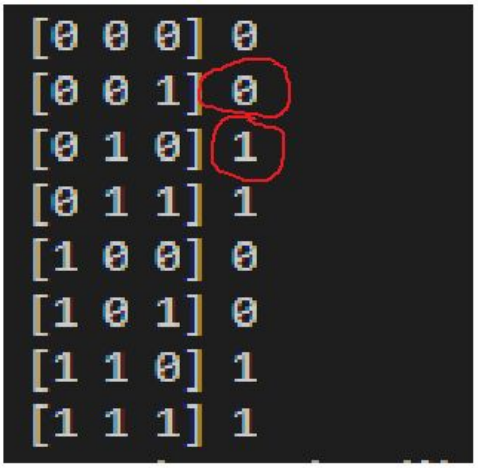
```
        for i in range(len(inputs)):
            prd = self.predict(inputs[i])
            self.weights += (labels[i]-prd) * inputs[i] * self.learningRate
            self.bias += (labels[i]-prd) * self.learningRate


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:



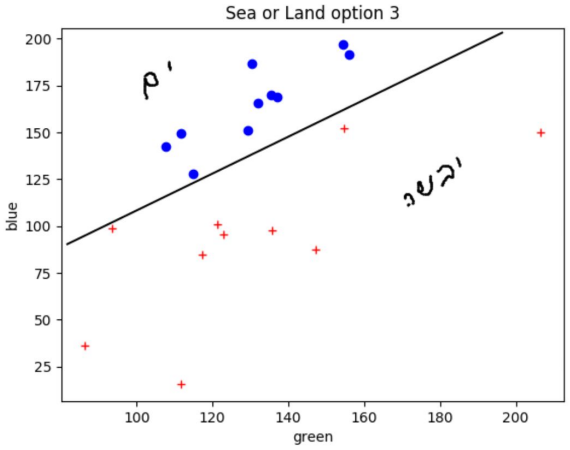
ניתן לראות שיש 2 טעויות ביחס למידע שהמערכת התבקשה ללמוד.

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.bias = 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.bias
    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.bias += (labels[i]-prd) * self.learningRate


#--------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)])


#--------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 = 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]


#--------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 !!!")
```

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.

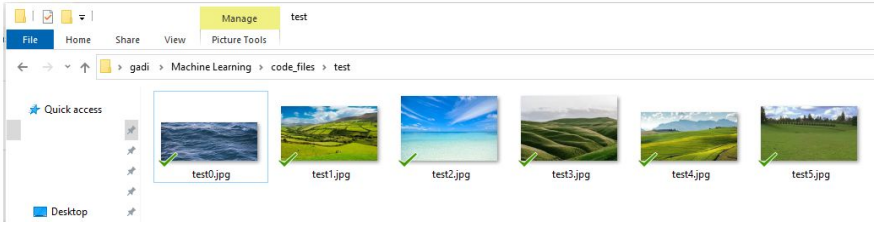
הקוד מחולק ל-4 חלקים:

- 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:



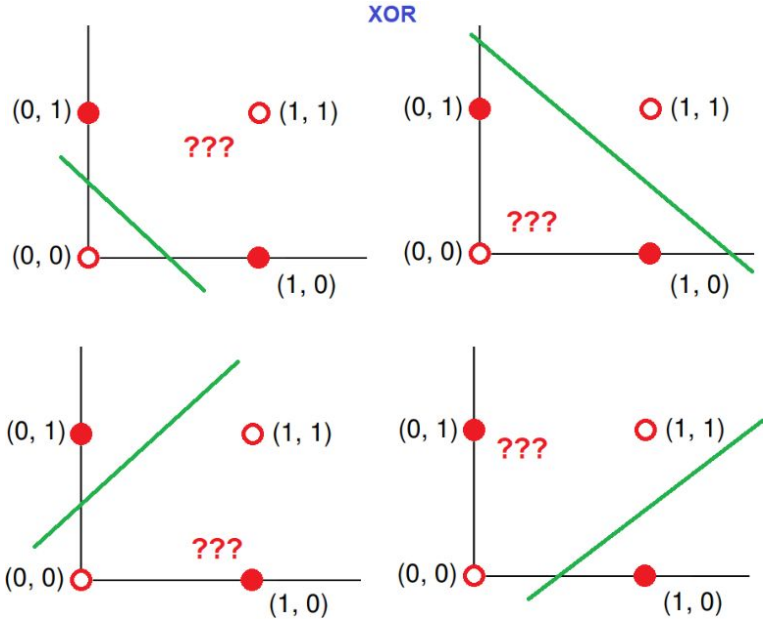
קיבלנו התאמה מלאה בין פלט המכונה לבין סדר התמונות המסופק למכונה.

הצלחנו לממש מכונה לומדת המבוססת על פרסטרון בודד כדי לזהות תמונות נוף של ים ויבשה!

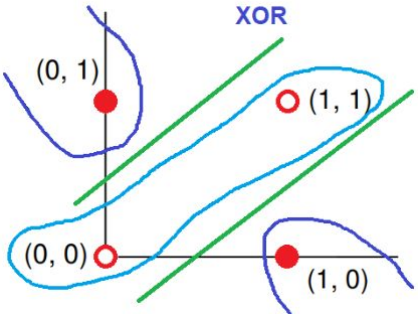
עלינו לזכור כי אימנו את הפרספטרון על בסיס נתונים קטן מאוד וניתן להפרדה לינארית. לא תמיד זהו המצב ולכן בהמשך הספר נלמד אלגוריתמים מתוחכמים יותר המסוגלים להתמודד עם בעיות הפרדה קשות יותר.

פעילות 11 - מימוש שער XOR על ידי רשת נוירונים

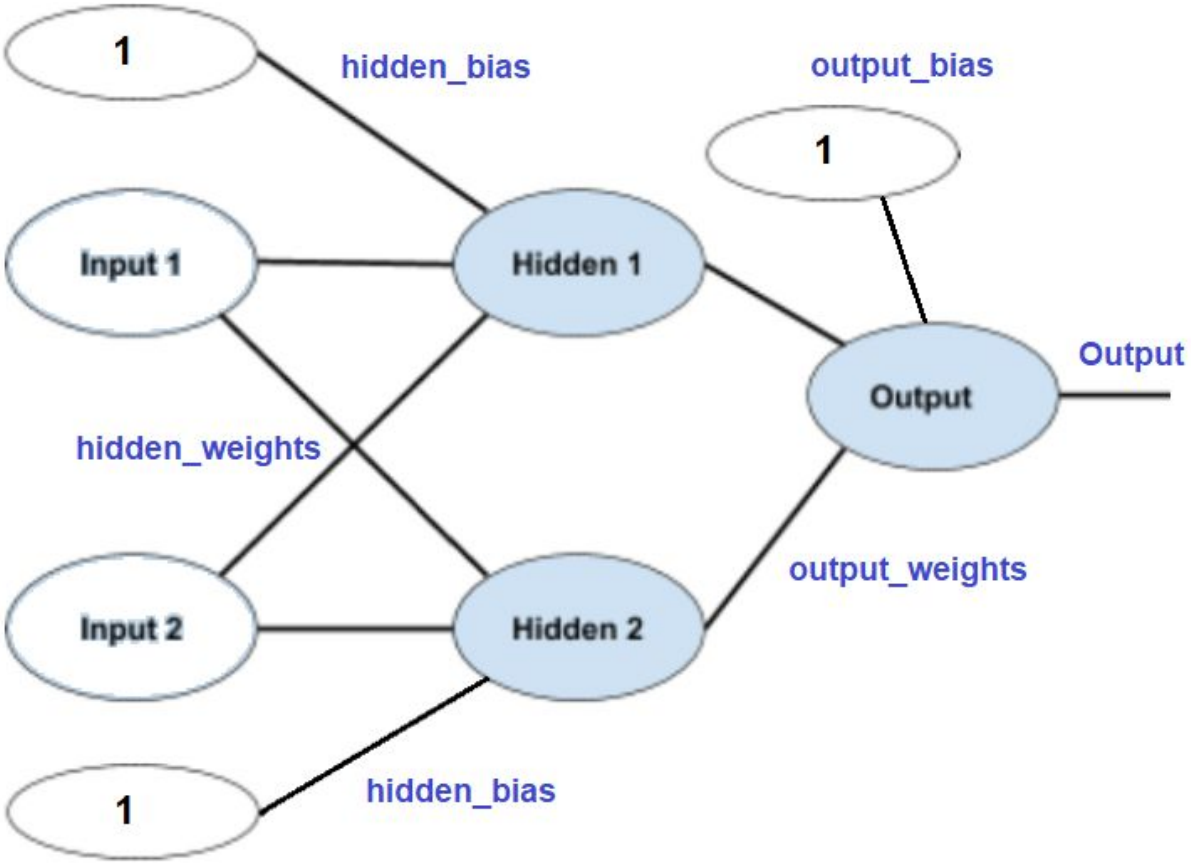
בפעילות 9 כתבנו קוד המממש פרספטרון בודד במטרה להבין ולתרגל את עקרון הפעולה של מכונה לומדת העושה שימוש בנוירון בודד. ראינו שפרספטרון מוגדר כמסווג לינארי כלומר רכיב תוכנה שבו פונקציית הסיווג מממשת משוואת קו ישר. מכאן שהצלחנו ללמד פרספטרון בודד לסווג אותות בינאריים במבוא כשער לוגי מסוג AND, OR ו- NAND ולא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



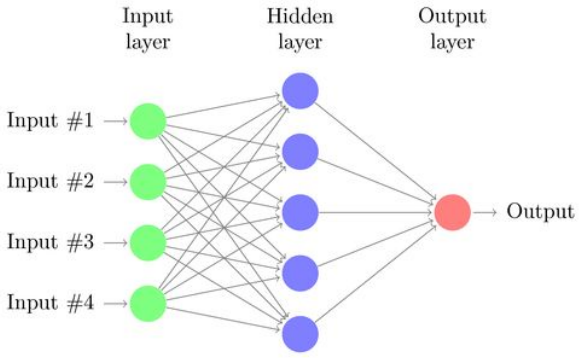
פתרון לכך יהיה מימוש רשת נוירונים במטרה לבנות מכונה המסוגלת ללמוד שער לוגי מסוג XOR.



נבנה רשת של 3 נוירונים על פי המודל הבא:



המודל מבוסס על 2 מבואות , 2 נוירונים בשכבה הפנימית ועוד נוירון אחד במוצא. מערך נוירונים בנוי בארכיטקטורה שבה כל נוירון מחובר לכל שאר הנוירונים בשכבה הבא. עקרון זה מכונה Fully connected neural network, ניתן לראות את עקרון החיבור באיור הבא:



מקור התמונה: http://www.texample.net/tikz/examples/neural-network

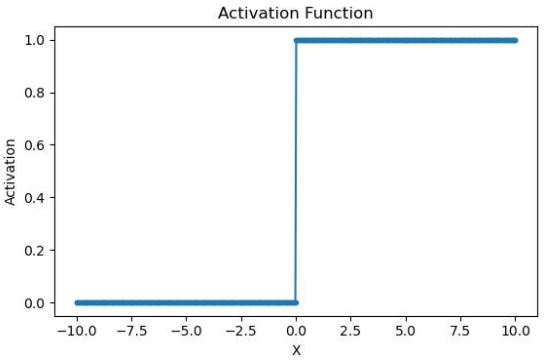
עדכון פונקציית האקטיבציה

בפעילות 10 כשפיתחנו קוד המממש פרספטרון ראינו שהוא בנוי מ-2 מרכיבים. הראשון פונקציית סכום של מכפלות בין כל אחד מהאותות במבוא כפול המשקל שלהם והשני היא פונקציית מעבר activation function המקבלת את סכום מכפלות המבוא במשקלים ומוציאה במוצא הפרספטרון אות בודד בטווח ערכים רצוי.

לצורך מימוש פונקציית המעבר השתמשנו בקוד הבא:

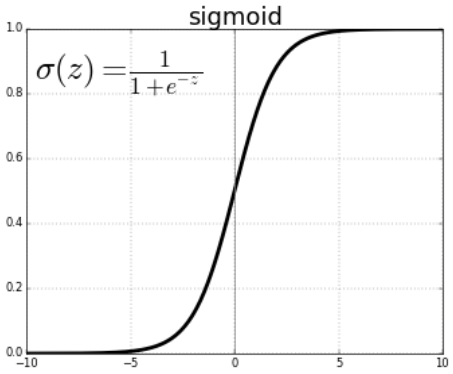
```
def Activation(s):
  if s > 0:
    activation = 1
  else:
    activation = 0
  return activation
```

ראינו גרף המתאר את אות המוצא ביחס לאות המבוא באופן הבא:



לצורך מימוש פרספטרון כחלק מרשת הכוללת מספר נוירונים נחליף את פונקציית המעבר לפונקציה מורכבת יותר בשם sigmoid. מצד אחד פונקציה זו אינה שונה באופן מהותי מהפונקציה שהשתמשו בה עד כה אך מצד שני היא ניתנת לגזירה, תכונה הכרחית על מנת שנוכל למצוא פרמטרים אופטימליים על ידי אלגוריתם gradient descent ובכך ניתן לשלב אותה ברשת הכוללת מספר נוירונים.

באיור הבא ניתן לראות גרף המתאר את המשוואה לפונקציה sigmoid.



בפעילות 9 למדנו שניתן להיעזר בנגזרת של פונקציית ה- COST של פרספטרון במטרה לחפש את נקודת המינימום בגרף ובדרך זו לקבל את הערכים האופטימליים של m ו- b. גם הפעם כשמדובר על רשת הבנוייה ממספר פרספטרונים המשימה נשארה זהה. אנו צריכים להשתמש בנגזרת של פונקציית ה- COST כדי לקבוע עבור כל הערכים של כל אחד מהנוירונים ברשת. משוואות הנגזרות החלקיות עבור כל אחד מהפרמטרים של הרשת חורג מהיקף ממדריך זה. יתרונה של פונקצית סיגמואיד היא בכך שחישוב הנגזרת יעיל חישובית.

האלגוריתם שנכתב עובד לפי השלבים הבאים:

1. נאתחל את כל המשקולות 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. נשנה את המשקולות של נוירון המוצא (Output) בהתאם לשגיאה הכללית באמצעות אלגוריתם gradient descent.
5. נשנה את המשקולות של 2 הנוירונים בשכבה הפנימית Hidden2 , Hidden1 בהתאם לייחס השגיאות כפי שעשינו בשלב 4.
6. נחזור לבצע את כל התהליך מסעיף 2 עד להתכנסות לנקודת מינמום של פונקציית השגיאה

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 = np.dot(d_predicted_output, self.output_weights.T)
```

```
        d_hidden_layer = error_hidden_layer * self.sigmoid_derivative(hidden_layer_output)


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

בשלב הבא נבנה עצם בשם nn מטיפוס 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 לתיוג 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 weights:\n",colored(nn.hidden_weights, 'green'))
print("\nFinal output bias:\n",colored(nn.hidden_bias, 'green'))
print("\nFinal output weights:\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'))
```

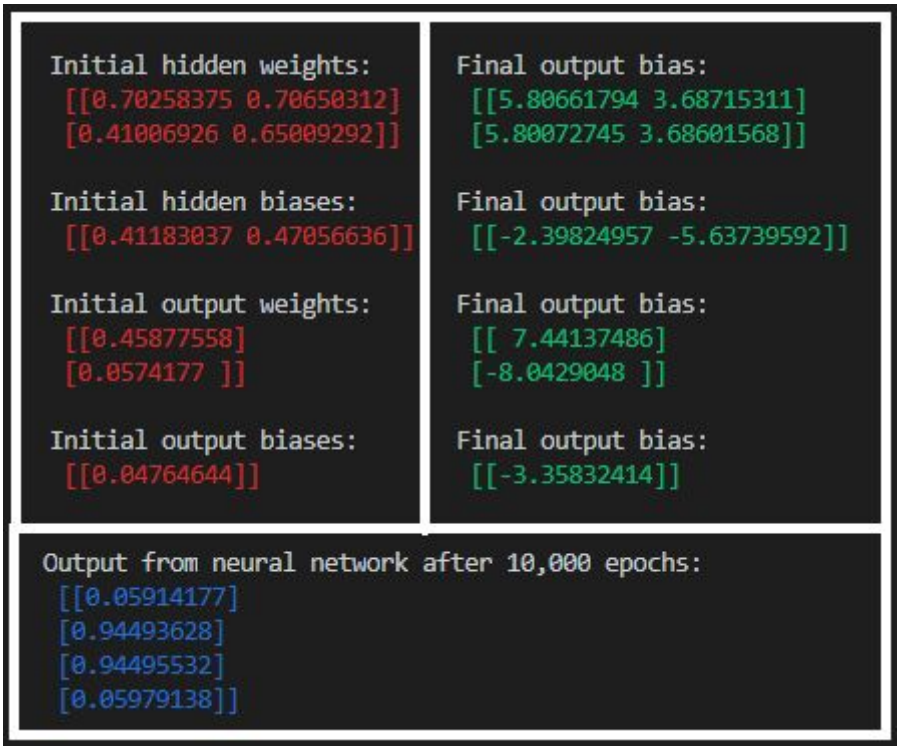
להלן מימוש הקוד המלא של רשת נוירונים ללימוד לוגיקה של שער XOR:

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

            #Updating Weights and Biases
            self.output_weights += np.dot(hidden_layer_output.T,d_predicted_output) \
             * learningRate
            self.output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * learningRate
            self.hidden_weights += np.dot(inpt.T,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 weights:\n",colored(nn.hidden_weights, 'green'))
print("\nFinal output bias:\n",colored(nn.hidden_bias, 'green'))
print("\nFinal output weights:\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:



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

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

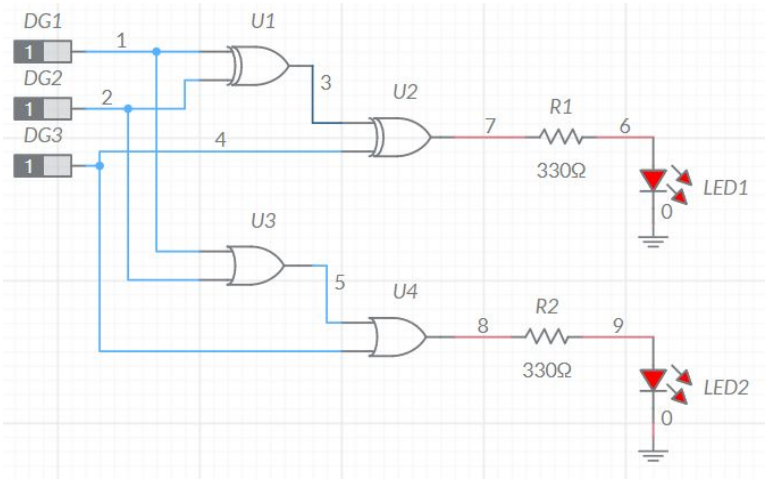
פלט התוכנית יהיו הערכים 0.05914177 ו- 0.05979138 המייצגים אפס לוגי והערכים 0.94493628 ו- 0.94495532 המייצגים אחד לוגי (המספרים הכחולים).

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

רשת הכוללת 2 פרספטרונים בשכבה החבויה לא תמיד תתכנס לפתרון הנכון בגלל בעיית מינימום מקומי. על מנת לפתור בעיה זו ניתן להגדיל את מספר הפרספטרונים בשכבה החבויה ולהגדיל את הסיכוי להתכנסות הרשת.

תרגיל

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



הנוריות LED1,LED2 פועלות על פי הערך הלוגי במוצא: מאירות עבור ערך "אמת" - 1 וכבויות עבור ערך "שקר" - 0.

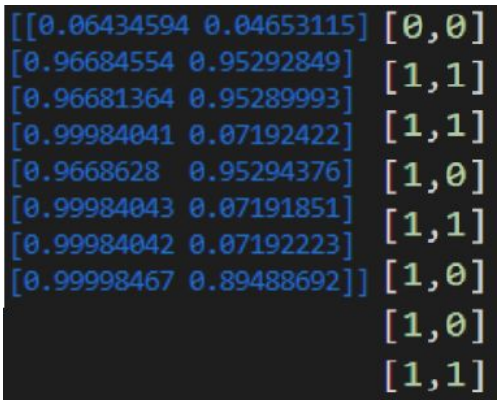
פתרון

```
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],
                                [1,1]])
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]]
```



תרגיל

הוסיפו לקוד המחלקת NeuralNetwork פעולה בשם predict. הפעולה תקבל ערכים המייצגים מידע לוגי המסופק לרשת הנוירונים. הפעולה תחזיר את פלט הרשת לאותם ערכים. הפעילו את הפעולה על הרשת מהתרגיל הקודם. על התוכנית לקלוט את 3 הערכים הבינאריים מהמשתמש. התוכנה תפסיק לקלוט כאשר אחד הערכים הנקלט מהמשתמש אינו מייצג רמה לוגית.

פתרון

```
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
```

פעילות 11 - רשת נוירונים לסיווג תמונות

בפעילות 9 כתבנו קוד המממש פרספטרון בודד במטרה להבין ולתרגל את עקרון הפעולה של מכונה לומדת בעלת נוירון בודד. ראינו שפרספטרון הינו מסווג לינארי כלומר אלגוריתם 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 ANN - Artificial neural networks. השתמשנו במחלקה NeuralNetwork כדי לבנות מכונה לומדת המממשת שער לוגי מסוג XOR.

בפעילות זו נשתמש באותה מחלקה שכתבנו בפעילות 10 כדי לסווג נתונים אמיתיים. בפרק זה נסווג פרחים..

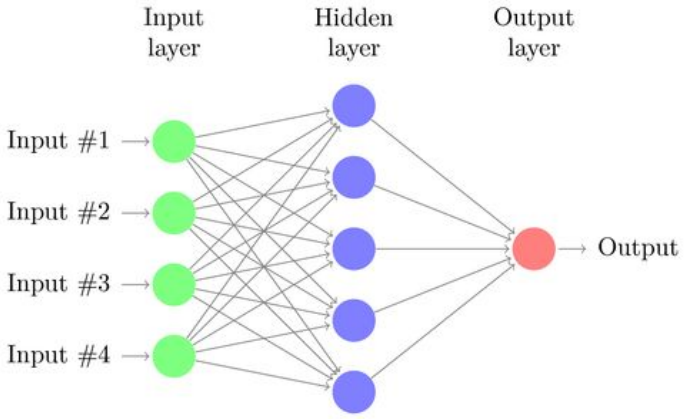
נחלק את הפעילות ל-3 שלבים:

- בשלב הראשון נכתוב מכונה לומדת לסיווג דוגמאות במרחב דו מימדי הניתנות להפרדה לינארית..
- בשלב השני נתרגל כתיבת מכונה לומדת לסיווג דוגמאות במרחב דו מימדי הניתנות להפרדה על ידי פולינום ממעלה 2
- בשלב השלישי נבנה 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 הפרספטרונים בשכבה הבא. עקרון זה מכונה Fully connected neural network, ניתן לראות את עקרון החיבור באיור הבא:

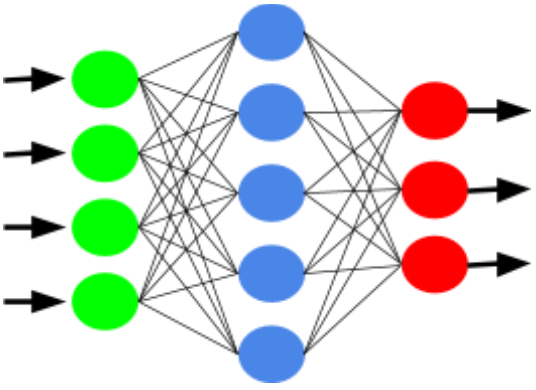


מקור התמונה: http://www.texample.net/tikz/examples/neural-network

למחלקה זו יכולת להגדיר שכבת מבוא (input layer)הכוללת מספר , 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: מכונה לומדת לסיווג נקודות של פונקציה לינארי.

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

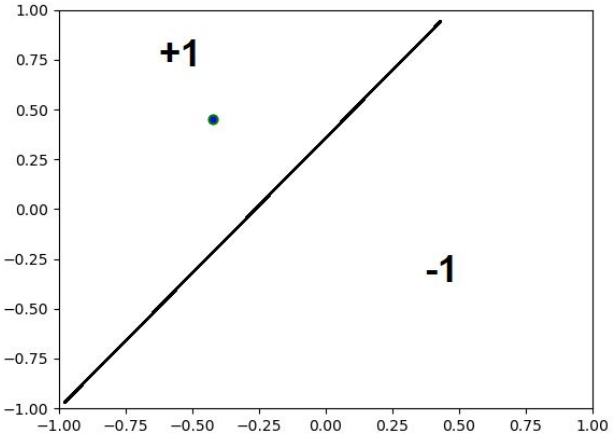
```
class listOfpoint:
    def __init__(self,numberOfPoints,m=1,b=0):
        self.points=np.random.uniform(-1,1,size=(numberOfPoints,2))
        self.labels=np.sign(self.points[:,1] - (m*self.points[:,0] + b))
```

```
def __repr__(self):
    s=''
    for point,label in zip(self.points,self.labels):
        s += "\tx="+str(point[0])+"\ty="+str(point[1])+"\tlabel="+str(label)+"\n"
    return s
```

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 המחלקה listOfpoint מקבלת פרמטר נוסף בשם numberOfPoints הקובע את מספר הנקודות הרצוי.

לסיכום: המחלקה listOfpoint כולל פעולה בונה המקבלת 3 פרמטרים: הראשון מספר הנקודות הרצוי ושני הפרמטרים הנוספים הם הערכים m ו- b של הקו הלינארי. המחלקה מייצרת מערך של נקודות. 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 listOfpoint:
    def __init__(self,numberOfPoints,m=1,b=0):
        self.points=np.random.uniform(-1,1,size=(numberOfPoints,2))
        self.labels=np.sign(self.points[:,1] - (m*self.points[:,0] + b))


    def __repr__(self):
        s=''
        for point,label in zip(self.points,self.labels):
            s += "\tx="+str(point[0])+"\ty="+str(point[1])+"\tlabel="+str(label)+"\n"
        return s


#------Test The Point Class----------------
MyPoints = listOfpoint(10,2,0.5)
print(MyPoints)
```

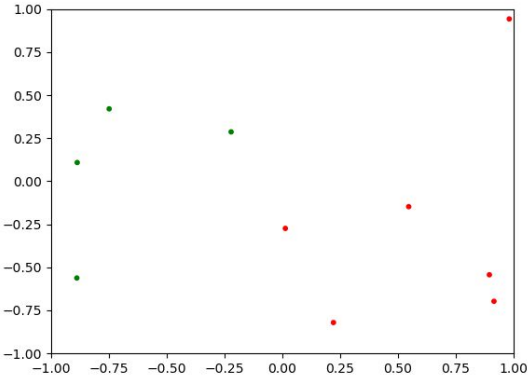
נקבל את הפלט הבא:

```
x=0.7572543535618854    y=-0.6806320391571195   label=-1.0
x=-0.10376648223938534  y=0.1852178752956779    label=-1.0
x=-0.47614512100946316  y=0.9939880049777705    label=1.0
x=-0.4743649797992755   y=-0.12961783958404882  label=1.0
x=0.28297045321633774   y=-0.7306279687259116   label=-1.0
x=-0.4426431910591835   y=0.08588649679370253   label=1.0
x=0.6013185511122008    y=-0.4180946595663779   label=-1.0
x=0.14014744903513643   y=-0.08104518811472672  label=-1.0
x=0.7863252231920219    y=0.5038575266965719    label=-1.0
x=0.5187599301102017    y=0.3005386586304113    label=-1.0
```

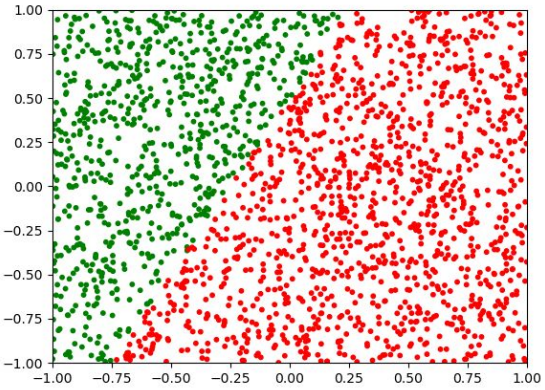
בגלל שקשה להבין כמות כזו של נתונים נוסיף למחלקה listOfpoint פעולה בשם drawTrainingPoints שמציגה גרף הכולל את מערך הנקודות כאשר כל נקודה שהוגדרה במאפיין label כאחד תקבל צבע ירוק וכל נקודה שתקבל ערך מינוס אחד צבע אדום. להלן קוד התוכנית כולל מימוש הפעולה drawTrainingPoints:

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


class listOfpoint:
    def __init__(self,numberOfPoints,m=1,b=0):
        self.points=np.random.uniform(-1,1,size=(numberOfPoints,2))
        self.labels=np.sign(self.points[:,1] - (m*self.points[:,0] + b))


    def __repr__(self):
        s=''
        for point,label in zip(self.points,self.labels):
            s += "\tx="+str(point[0])+"\ty="+str(point[1])+"\tlabel="+str(label)+"\n"
        return s


    def drawTrainingPoints(self):
        '''
        Draw the training inputs and the training label
        '''
        colormap = np.array(['r', 'g'])
        plt.scatter(self.points[:,0], self.points[:,1],s=10, c=colormap[1*(self.labels==1)])
        plt.show()


#------Test The Point Class----------------
MyPoints = listOfpoint(10,2,0.5)
MyPoints.drawTrainingPoints()
```

להלן פלט התוכנית:



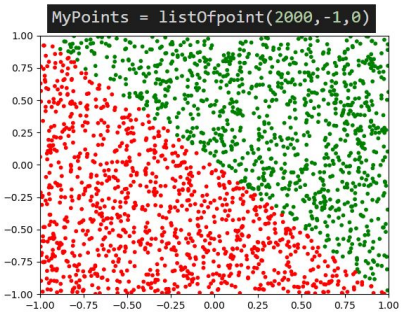
עכשיו כשיש לנו כלי גרפי להצגת נקודות נבדוק את המחלקה על 2000 נקודות ונקבל את הפלט הבא:



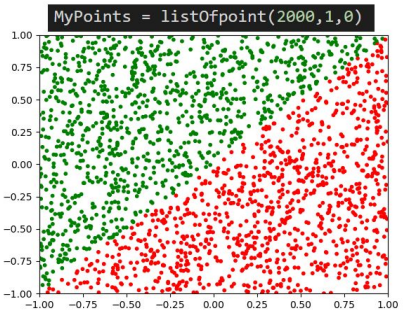
מאיור זה ניתן לראות את קו הגמה כפי שהגדרנו אותו בקוד התוכנית.

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

```
MyPoints = listOfpoint(2000,-1,0)
```



```
MyPoints = listOfpoint(2000,1,0)
```



השלב הבא במשימה יהיה לזמן את המחלקה NeuralNetwork כדי לבנות את רשת הנוירונים. 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.

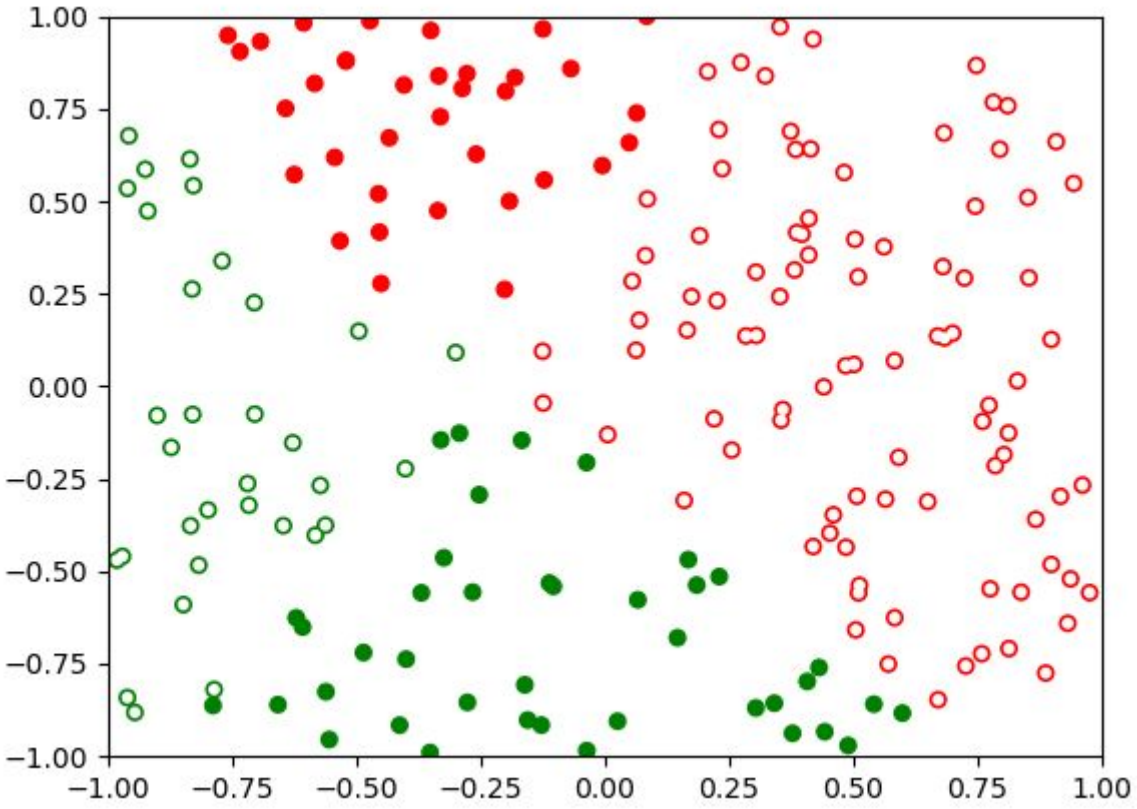
להלן הקוד:

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

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.points)
accuracy = test_points.drawMistakesPoints(predict_values)
print ('Accuracy: ',accuracy)
```

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



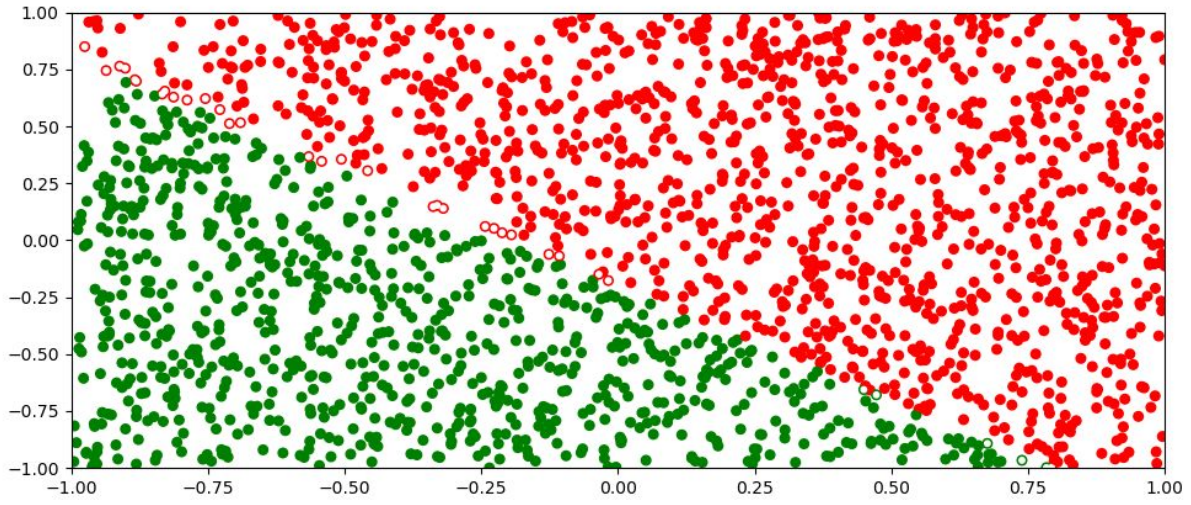
- עיגול אדום - זיהוי תקין.
- עיגול ירוק - זיהוי תקין.
- עיגול אדום עם לבן - זיהוי שגוי של נקודה.
- עיגול ירוק עם לבן - זיהוי שגוי של נקודה.

להלן מימוש הפעולה:

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

```
#Draw the correct answers
categories = []
accuracy = 0
for i in range(len(self.points)):
    if self.labels[i] > 0:
        categories.append(0)
    else:
        categories.append(1)
plt.scatter(self.points[:,0], self.points[:,1], s=30, c=colormap1[categories])
#-----------------------------------------------
categories = []
for i in range(len(predict_values)):
    if (predict_values[i] > 0.5) and (self.labels[i] > 0):
        categories.append(0)
        accuracy +=1
    elif (predict_values[i] < 0.5) and (self.labels[i] < 0):
        categories.append(1)
        accuracy +=1
    else:
          categories.append(2)

plt.scatter(self.points[:,0], self.points[:,1], s=10, c=colormap2[categories])
plt.axis([-1,1,-1,1])
plt.show()
return accuracy / len(predict_values)
```

הפעולה מחזירה מדד מספרי בשם accuracy המייצג את מספר הסיווגים הנכונים ביחס למספר הדוגמאות הכולל. לדוגמה:



נקבל:

```
Accuracy:  0.9825
```

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))
```

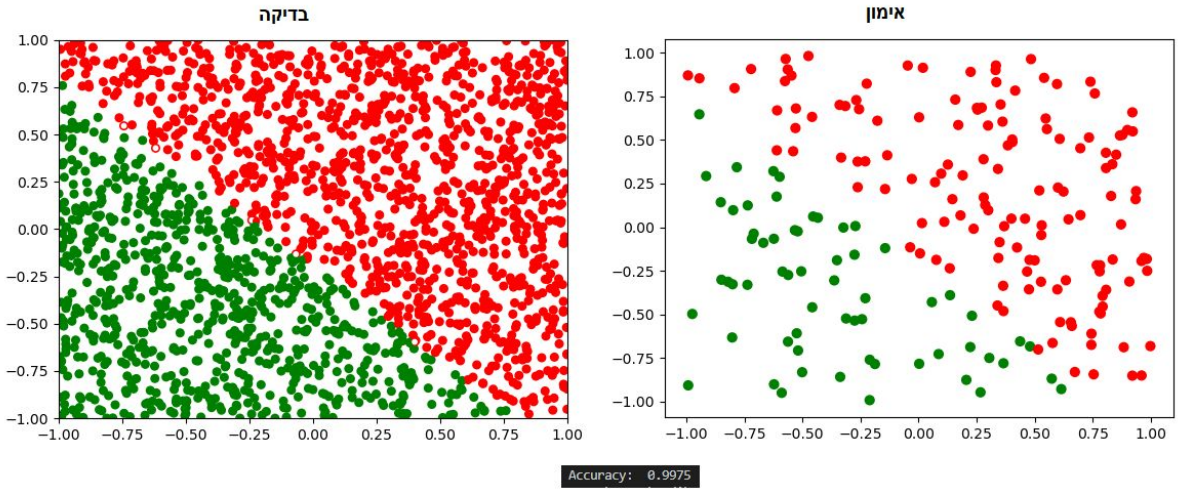
```
def sigmoid_derivative(self,x):
    return x * (1.0 - x)

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 listOfpoint:
  def __init__(self,numberOfPoints,m=1,b=0):
    '''
    y=mx+b
    '''
    self.points=np.random.uniform(-1,1,size=(numberOfPoints,2))
    self.labels=np.sign(self.points[:,1] - (m*self.points[:,0] + b))
    # x2= x.reshape(x.shape[0],1)
    self.labels= self.labels.reshape(self.labels.shape[0],1)

  def __repr__(self):
    s=''
    for point,label in zip(self.points,self.labels):
      s += "\tx="+str(point[0])+"\ty="+str(point[1])+"\tlabel="+str(label)+"\n"
    return s

  def drawTrainingPoints(self):
    '''
    Draw the training inputs and the training label
```

```
    '''
    colormap = np.array(['r', 'g'])
    categories = []
    for i in range(len(self.points)):
        if self.labels[i] > 0:
            categories.append(0)
        else:
            categories.append(1)
    plt.scatter(self.points[:,0], self.points[:,1], s=40, c=colormap[categories])
    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(['r','g','w'])
    #Draw the correct answers
    categories = []
    accuracy = 0
    for i in range(len(self.points)):
        if self.labels[i] > 0:
            categories.append(0)
        else:
            categories.append(1)
    plt.scatter(self.points[:,0], self.points[:,1], s=30, c=colormap1[categories])
    #-----------------------------------------------
    categories = []
```

```
        for i in range(len(predict_values)):
            if (predict_values[i] > 0.5) and (self.labels[i] > 0):
                categories.append(0)
                accuracy +=1
            elif (predict_values[i] < 0.5) and (self.labels[i] < 0):
                categories.append(1)
                accuracy +=1
            else:
                    categories.append(2)


        plt.scatter(self.points[:,0], self.points[:,1], s=10, c=colormap2[categories])
        plt.axis([-1,1,-1,1])
        plt.show()
        return accuracy / len(predict_values)


#------Train the Neural Network--------------------
train_points = listOfpoint(200,-1,-0.2)
train_points.drawTrainingPoints()
nn = NeuralNetwork(2,2,1)
nn.train(train_points.points, train_points.labels)
#--------TEST the Neural Network-----------------------
test_points = listOfpoint(2000,-1,-0.2)
predict_values = nn.predict(test_points.points)
accuracy = test_points.drawMistakesPoints(predict_values)
print ('Accuracy: ',accuracy)
```

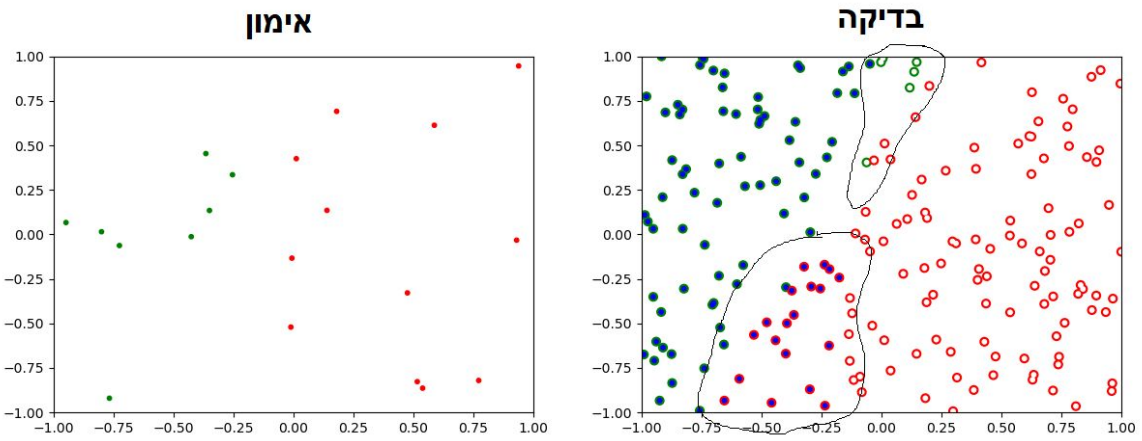
להלן פלט התוכנית:



ניתן לראות שהמערכת למדה טוב. היא הגיע ל- Accuracy של 0.9975

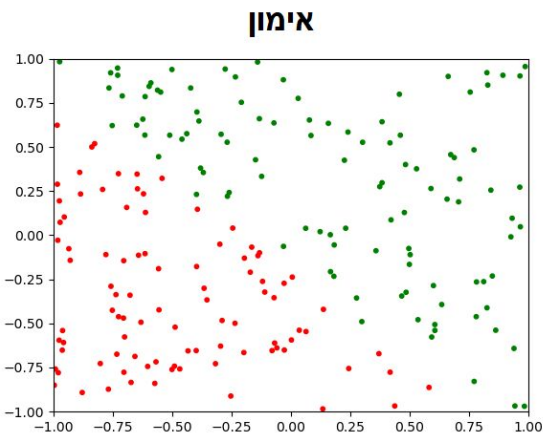
זאת כמובן בעיית הפרדה לינארית פשוטה ולכן אפשר לצפות מרשת להגיע להפרדה כזו.

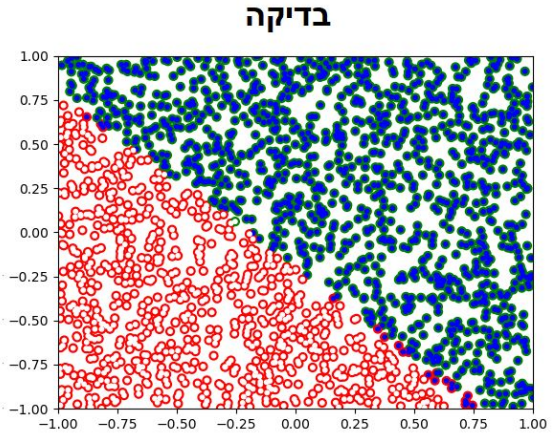
נאתגר את המערכת ביותר נקודות בדיקה ופחות אימון:



כאשר סיפקנו לערכת 20 נקודות לאימון ו- 200 נקודות בדיקה ניתן לראות שהמערכת לא הצליחה ללמוד מספיק טוב כך שקיבלנו נקודות שגויות. שאלה למחשבה: גם במקרה זה הנקודות ניתנות להפרדה לינארית. מדוע הרשת לא הצליחה ללמוד קו הפרדה נכון?

ההסבר נובע מכך שכאשר יש מעט נקודות אימון האלגוריתם בוחר קו אשר ביצועיו טובים על נתוני האימון, אולם לא בהכרח על נתוני המבחן. תופעה זו מכונה overfitting. הגדלת מספר נקודות האימון מקטינה את האפשרות ל overfitting.





תרגיל

שנו את האלגוריתם של המחלקות 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
```

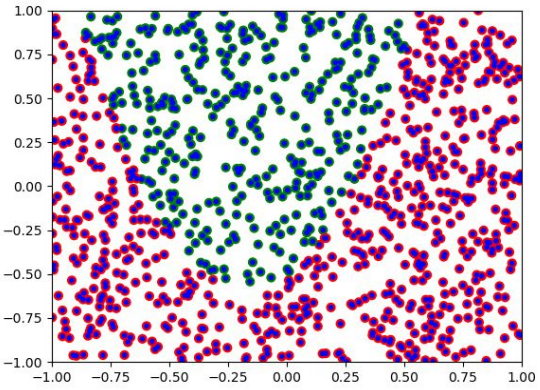
```
    '''
    self.points = []
    for _ in range(numberOfPoints):
        self.points.append(parabolaPoint(a,b,c))


    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)
```

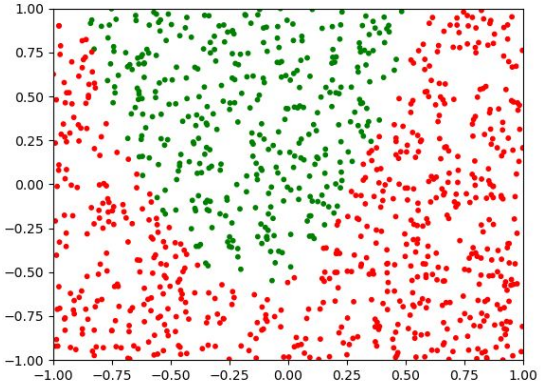
```
train_points.drawTrainingPoints()

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:

בדיקה



אימון

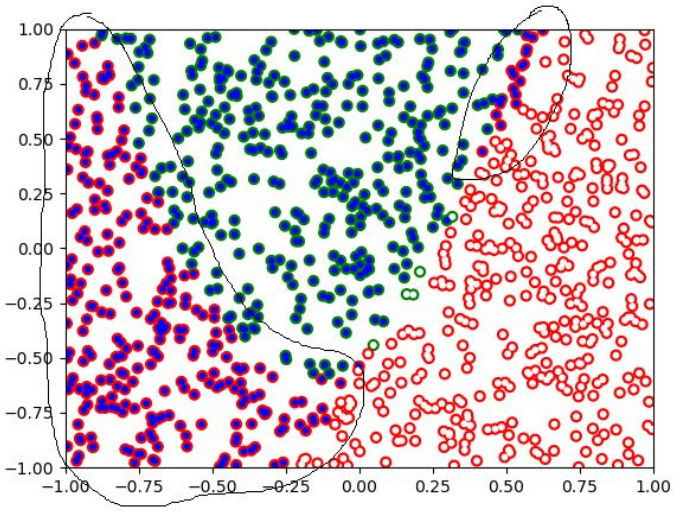


שאלה

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

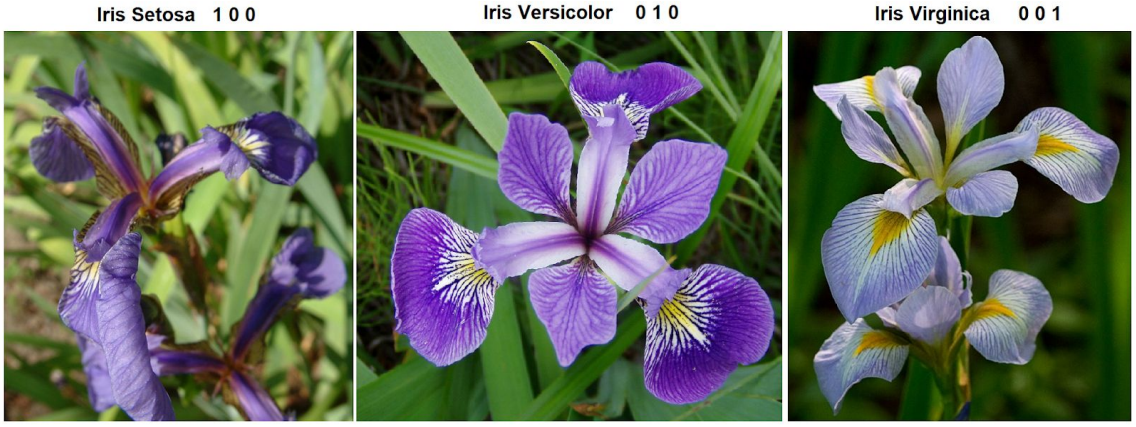
תשובה

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



משימה: מכונה לומדת לזיהוי פרחים.

במטרה לבחון את המחלקה NeuralNetwork על נתונים אמיתיים נשתמש במערך נתונים קיים הכולל מידע מתויג על 3 סוגים שונים של אירוסים. להלן דוגמא לתמונות של שלושת האירוסים:



מקור התמונה: https://commons.wikimedia.org/wiki/Iris_Iridaceae

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 מבואות. כמו כן יהיה צורך ב-3 נוירונים במוצא עבור אבחון 3 סוגי האירוסים.

מימוש רשת בעלת 250000 מבואות תדרוש מספר עצום של חישובים ולכן זמן אימון רב מאוד. ננסה לפתור את הבעיה באופן שונה וננסה לזהות מתוך התמונות מספר מאפיינים שיתנו לו את הבחנות המבדילות עבור

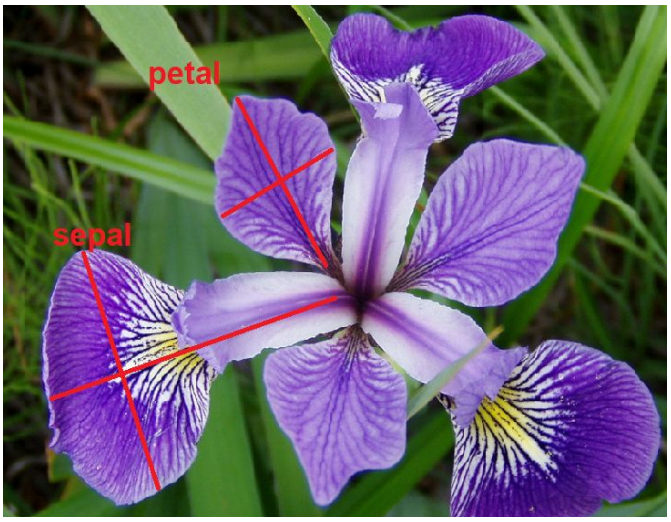
שלושת סוגי הפרחים (דוגמה לכך היא לאפיין לפי צבע, גדול.....). בדרך זו ניתן לצמצם משמעותית את מספר הנוירונים במבוא המכונה ולאפשר למחלקה שלנו לנסות לפתח אלגוריתם שיבדיל בין שלושת האירוסים.

בפרקים הבאים נראה גישות נוספות על מנת להקטין את גודל הקלט מבלי לחשב מאפיינים באופן ידני, על ידי הורדת הרזולוציה של התמונה.

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

- petel length (אורך עלי כותרת)
- petal width (רוחב עלי כותרת)
- sepal length (אורך עלי הגביע)
- sepal width (רוחב עלי הגביע)

להלן תמונה הממחישה את המאפיינים (Features) של הפרח



קישור למסמך בקישור הבא:

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

ניתן להוריד את קובץ הנתונים ישירות מהקישור הבא:

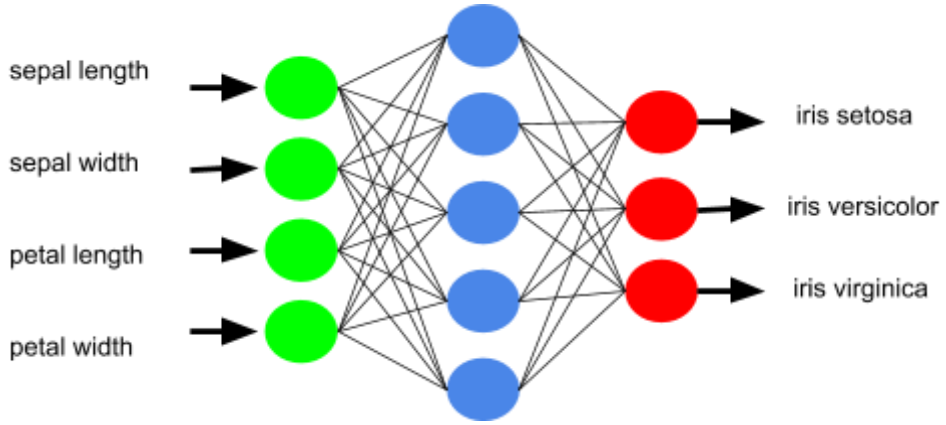
https://www.neuraldesigner.com/files/datasets/iris_flowers.csv

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

מוצג רק חלק מהקובץ.

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



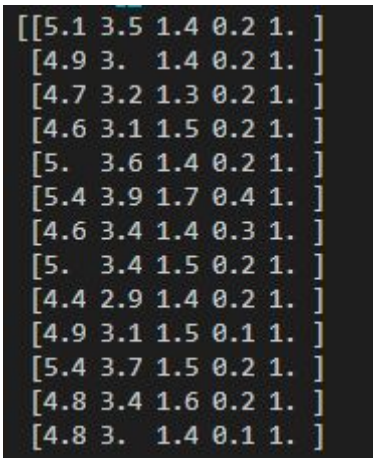
בשלב הבא נקלוט את נתוני הקובץ לתוך מערך numpy שאנו מכירים. להלן דוגמת הקוד:

```
import numpy as np
from termcolor import colored


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

הערה: יש לבצע את הפעולות הבאות על קובץ הנתונים:

- נמחק מהקובץ את שורת הכותרת.
- נשנה את שם הפרח למספרים 1 עד 3 בהתאמה לשם הפרח.

נקבל את הפלט הבא:



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

- נערבב את השורות בקובץ כדי שהפרחים לא יהיה מסודרים לפי סוג.
- נחלק את המערך ל- 4 מערכים על פי הפירוט הבא:
  a. מערך נתונים לאימון
  b. מערך תגיות לאימון
  c. מערך נתונים לבדיקה
  d. מערך תגיות לבדיקה

מערכי התגיות בנויים בשיטה המכונה one hot encoding. בגישה זו בונים עמודה עבור כל אחד מהסוגים לתיוג המכילה 1 עבור דוגמאות מסוג זה ו- 0 עבור דוגמאות מסוגים אחרים.

להלן המימוש בקוד:

```
import numpy as np
from termcolor import colored

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

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

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

test_data:
 [[5.7 3.8 1.7 0.3]
 [6.  2.9 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.9 3.  5.1 1.8]
 [6.3 2.7 4.9 1.8]
 [5.4 3.7 1.5 0.2]
 [4.6 3.2 1.4 0.2]
 [5.1 3.8 1.9 0.4]
 [4.9 3.1 1.5 0.1]
 [5.1 3.4 1.5 0.2]]

train_data:
 [[5.7 2.6 3.5 1. ]
 [5.5 2.4 3.7 1. ]
 [6.6 2.9 4.6 1.3]
 [7.1 3.  5.9 2.1]
 [5.  3.3 1.4 0.2]
 [7.7 3.  6.1 2.3]
 [4.9 2.5 4.5 1.7]
 [4.7 3.2 1.3 0.2]
 [4.9 2.4 3.3 1. ]
 [5.7 2.8 4.5 1.3]
 [7.9 3.8 6.4 2. ]
 [4.9 3.1 1.5 0.1]
 [6.1 2.8 4.7 1.2]
 [4.8 3.4 1.9 0.2]
 [6.9 3.2 5.7 2.3]

test_lbl:
 [[1.]
 [2.]
 [2.]
 [3.]
 [3.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]]

new_test_lbl:
 [[1 0 0]
 [0 1 0]
 [0 1 0]
 [0 0 1]
 [0 0 1]
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]]

train_lbl:
 [[2.]
 [2.]
 [2.]
 [3.]
 [1.]
 [3.]
 [3.]
 [1.]
 [2.]
 [2.]
 [3.]
 [1.]
 [2.]
 [1.]
 [3.]
 [1.]

new_train_lbl:
 [[0 1 0]
 [0 1 0]
 [0 1 0]
 [0 0 1]
 [1 0 0]
 [0 0 1]
 [0 0 1]
 [1 0 0]
 [0 1 0]
 [0 1 0]
 [0 0 1]
 [1 0 0]
 [0 1 0]
 [1 0 0]
 [0 0 1]
 [1 0 0]
 [1 0 0]
 [1 0 0]
```

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

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

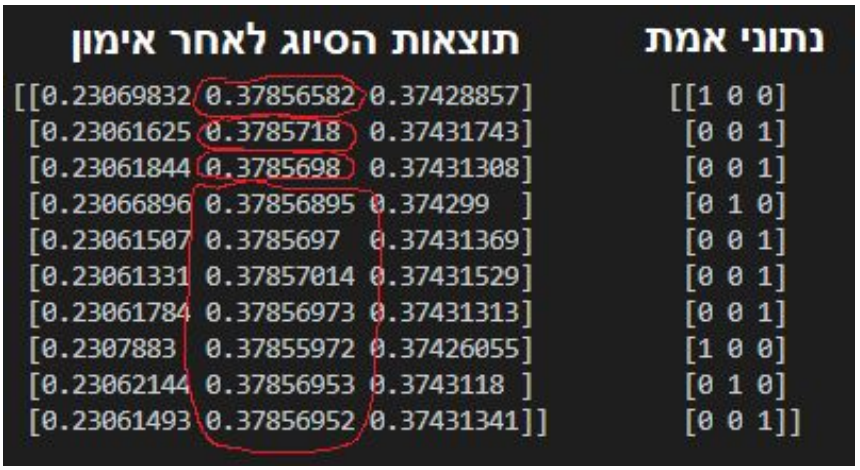
```
class NeuralNetwork:
    def __init__(self,inputLayerNeurons,hiddenLayerNeurons,outputLayerNeurons):
        self.hidden_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeurons))
        self.hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
        self.output_weights =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

#------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:



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.

נשפר את האלגוריתם שכתבנו על ידי כך הגדלת מספר הנתונים שעליהם אנו מבצעים בדיקה מ- 10 ל- 40 לצורך זה עלינו לחשב את מדד ה- accuracy כי בכמות כזו לא נצליח לזהות הבדלים. להלן הקוד המעודכן:

```
#------Train the Neural Network--------------------

nn = NeuralNetwork(4,5,3)

nn.train(train_data, new_train_lbl,0.01,1000)

#--------TEST the Neural Network--------------------

predict_values = nn.predict(test_data)

print('predict_values:\n',predict_values)

print('true_values:\n',new_test_lbl)

predict_indices = np.argmax(predict_values, axis=1)

print('predict_indices:\n',predict_indices)
```

```
true_indices = np.argmax(new_test_lbl, axis=1)

print('true_indices:\n',true_indices)

accuracy = np.mean(predict_indices==true_indices)

print('accuracy:\n',accuracy)
```

נקבל את הפלט הבא:

```
predict_indices:
 [0 2 2 0 1 2 1 0 2 1 1 2 0 0 0 2 2 0 1 1 2 1 2 2 2 1 1 2 1 2 0 0 0 2 2 0 1
 2 1 1]
true_indices:
 [0 2 2 0 1 2 1 0 2 1 1 2 0 0 0 1 1 0 1 1 2 1 2 2 2 1 1 2 1 2 0 0 0 2 2 0 1
 2 1 1]
accuracy:
 0.95
```

ראינו שכאשר הורדנו את ה-learningRate והגדלנו את ה-epochs במקביל להגדלת מספר הבדיקות הגענו לתוצאות לא רעות בכלל. כמובן אנו לא במערכת למידת מכונה אופטימאלית כי אז היינו צריכים לקבל תוצאות מובהקות יותר מאלו שקיבלנו בפעילות זו.

כמו כן בפרק הבא נפתח מכונות לומדות המבוססות על אלגוריתמים של רשתות נוירונים מלאכותיים ANN העושות שימוש בכלי תוכנה מקצועיים שם נוכל לממש אלגוריתמים מורכבים מאוד בנוחות רבה ביחס למימוש עצמי של הקוד.

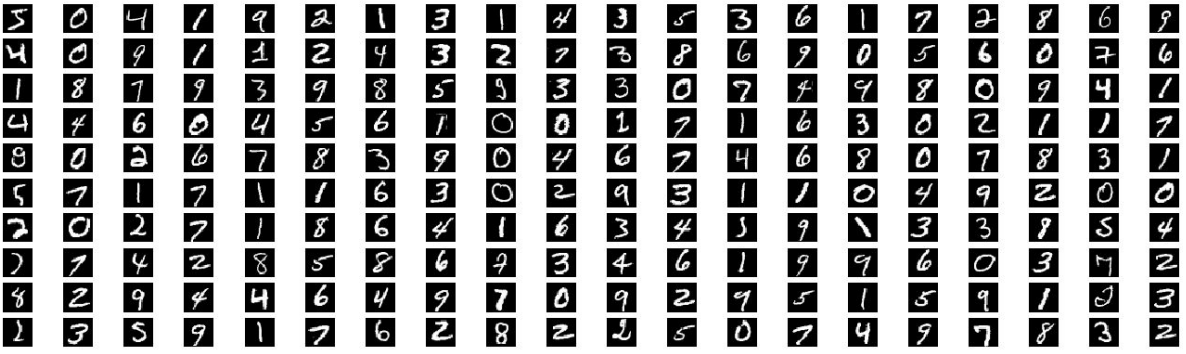
מור-טק מרכז המורים הארצי למקצועות הטכנולוגיים
הפקולטה לחינוך למדע וטכנולוגיה, קריית הטכניון, חיפה 32000 טל: 04-8293146
E-mail: Moretech@ed.technion.ac.il
http://moretech.technion.ac.il

יישומי מכונה לומדת בשפת Python

מור-טק מרכז המורים הארצי למקצועות הטכנולוגיים
הפקולטה לחינוך למדע וטכנולוגיה, קריית הטכניון, חיפה 32000 טל: 04-8293146
E-mail: Moretech@ed.technion.ac.il
http://moretech.technion.ac.il

פעילות 13 - סיווג עם ספריית scikit-learn

בפעילות זה נתרגל כתיבת קוד בשפת python המאמן מכונה לומדת לזיהוי מספרים מתוך מאגר תמונות.
בפעילות זה נשתמש ב- 70000 תמונות מתויגות הנראות כך:



כפי שסיכמנו את פרק 2 שבו למדנו את עקרונות התכנות של מכונה לומדת תוך כדי כתיבת קוד מהיסוד, או במילים אחרות כתיבת מכונה לומדת ללא שימוש בספריות python ייעודיות לך. החל מפרק 3 אנו עוברים לממש מכונות לומדות תוך שימוש בספריות מוכנות שנותנות לנו כלים נוחים, יעילים ומורכבים כאחד כדי לפתח מכונות למידה.

הספריה הראשונה שנשתמש בה כדי ללמוד לפתח מכונה לומדת היא Scikit-learn שהיא אחת מהספריות ללמידת מכונה של Python. הספרייה נכתבה כך שתתמוך בספריות משנה שכבר הכרנו כמו Numpy. הספרייה ברובה נכתבה בשפת Python כאשר חלקים ממנה נכתבו בשפת C כדי לספק ביצועים גבוהים.

הספריה נכתבה במקור על ידי David Cournapeau כחלק מפרויקט של גוגל. נכון להיום ניתן לראות שימוש של הספריה כחלק ממערך המלצות השירים באפליקציה של Spotify כמו גם נעשה שימוש בספריה Scikit-learn באתר Booking.com כדי לספק לגולשים המלצות על מלונות ויעדים לטיול כמו גם איתור הונאות במערך ההזמנות של החברה.

היכרות ראשונה על הספריה Scikit-learn

נתחיל בכתיבת קוד למכונה לומדת המבוססת על הספריה Scikit-learn כדי לממש שער XOR. כזכור בפעילות 10 כתבנו כבר קוד הממש שער XOR תוך שימוש ברשת של 5 פרספטרונים. באותה הפעילות התבססנו 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 הרשת. ננצל את הידע מפעילות 10 כדי לממש שער XOR תוך שימוש בספרייה ייעודית ללמידת מכונה.

להלן קוד התוכנית:

```
import numpy as np
import sklearn.neural_network

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

model = sklearn.neural_network.MLPClassifier(
            activation='logistic',
            max_iter=100,
            hidden_layer_sizes=(2,),
            solver='lbfgs')
model.fit(inputs, expected_output)
print('predictions:', model.predict(inputs))
```

נקבל את הפלט הבא:

```
predictions: [0 1 1 0]
```

נבחן את השורה המגדירה את המבנה של המכונה:

```
model = sklearn.neural_network.MLPClassifier(
            activation='logistic',
            max_iter=100,
            hidden_layer_sizes=(2,),
            solver='lbfgs')
```

ההוראה מזמנת את הפעולה MLPClassifier של המחלקה neural_network המהווה חלק מהספריה sklearn. הפעולה מספקת לנו כלי הגדרת רשת נוירונים מלאכותית Multi-layer Perceptron. הפעולה מקבלת סדרה ארוכה של פרמטרים שאת חלקם שילבנו בדוגמה ואלה הם:

- פרמטר activation מגדיר את פונקציית התמסורת של פרספטרון. במקרה שלנו הגדרנו את הפונקציה כ- logistic כלומר הפונקציה sigmoid כמו שפגשנו בעבר
- פרמטר max_iter מגדיר מספר הפעמים שבה רשת הנוירונים תעבור על מערך נתוני האימון.
- פרמטר hidden_layer_sizes מגדיר את מערך השכבות הפנימיות של הרשת. במקרה שלנו הגדרנו שכבה בודדת אחת שבה 2 נוירונים.
- הפרמטר solver מגדיר האלגוריתם שבו רשת הנוירונים מבצעת חישובי משקלים. על פי המלצת המפתחים יש להשתמש במאפיין lbfgs כאשר עובדים עם מערך קטן של נתוני למידה.

לקבלת מפרט מלא של הפרמטרים:

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

לקבלת קוד המחלקה:

https://github.com/scikit-learn/scikit-learn/blob/b194674c4/sklearn/neural_network/_multilayer_perceptron.py#L691

נבחן כעת את יעילת המחלקה עבור יכולת הסיווג של מערך הפרחים. נכתוב את הקוד הבא:

```
import numpy as np
import sklearn.neural_network



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 = np.ravel(random_iris_data[0:10,4:])
train_lbl = np.ravel(random_iris_data[10:,4:])
```

```
mlp = sklearn.neural_network.MLPClassifier(
            hidden_layer_sizes=(5),
            solver='sgd',
            learning_rate_init=0.01,
            max_iter=1000)

mlp.fit(train_data, train_lbl)

for i in range(len(test_data)):
   tt=test_data[i].reshape( 1,(test_data[i].shape[0]))
   p = mlp.predict(tt)
   print("test_lbl:" , test_lbl[i]  ,"prediction: " , p )
```

נקבל את הפלט הבא:

```
test_lbl: [3.] prediction:  [3.]
test_lbl: [2.] prediction:  [2.]
test_lbl: [3.] prediction:  [3.]
test_lbl: [2.] prediction:  [2.]
test_lbl: [2.] prediction:  [2.]
test_lbl: [2.] prediction:  [2.]
test_lbl: [3.] prediction:  [3.]
test_lbl: [1.] prediction:  [1.]
test_lbl: [3.] prediction:  [3.]
test_lbl: [1.] prediction:  [1.]
```

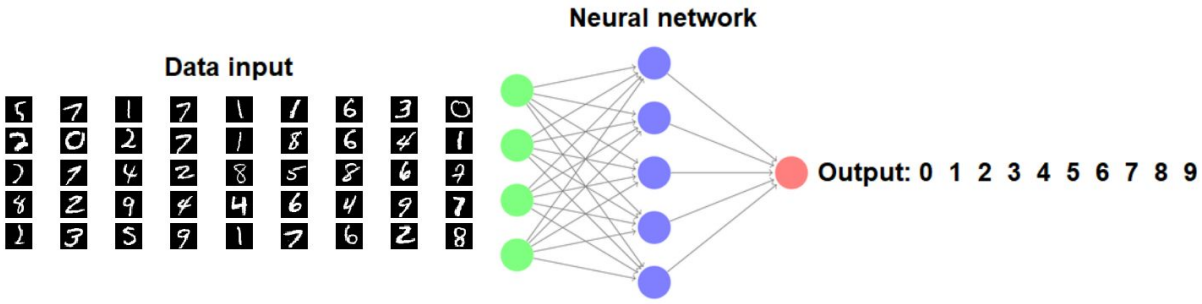
בתרגיל זה בנינו מכונה לומדת מבוססת על רשת נוירונים במבנה הבא: 4 מבואות, 5 נוירונים בשכבה האמצעית ו-3 נוירונים במוצא. כיול משקלים ברשת יתבצע על ידי אלגוריתם stochastic gradient descent with no batch-size או בקיצור sgd. כמו כן learning rate של 0.01 ו-1000 סבבי למידה.

הערה: לספריית sklearn פעולות המחשבות מגוון של מדדי ביצוע של המסווגים כולל accuracy בשיטה של cross validation ו- confusion matrix.

ראו https://scikit-learn.org/stable/modules/cross_validation.html

מכונה לומדת לזיהוי ספרות בכתב יד

נתרגל כתיבת קוד בשפת python המאמן מכונה לומדת לזיהוי מספרים מתוך מאגר תמונות. בפעילות זה נעשה שימוש ב- 70000 תמונות מתויגות.



ארגון נתוני האמון

כפי שכבר למדנו ארגון מערך נתוני האימון הוא חלק מרכזי בכתיבת מכונה לומדת. בתרגיל זה נעשה שימוש במערך תמונות מוכן המתואר בקישור הבא:

https://www.openml.org/d/554

ניתן ללמוד מהאתר שמערך הנתונים כולל 70000 תמונות, כל תמונה כוללת 784 פיקסלים (כלומר כל תמונה היא ברזולוציה של 28*28) התמונות מתויגות ל- 10 קטגוריות שונות (כלומר כל תמונה מתויגת לאחת הספרות 0 עד 9).

ניתן לעבוד עם מערך הנתונים ב-2 דרכים:

1. להוריד את קובץ התמונות ישירות למחשב האישי שלכם (קובץ בגודל 52.8M)

   להלן קישור לקובץ מסד הנתונים:

https://github.com/amplab/datascience-sp14/raw/master/lab7/mldata/mnist-original.mat

2. להשתמש בספרייה מוכנה של sklearn כדי להוריד את הנתונים ישירות מהאינטרנט בכל פעם שמפעילים את התוכנה. בדרך זו לא צריך לוודא היכן קובץ הנתונים שמור במחשב והפעולה fetch_openml שבספריה sklearn.datasets תדאג לכך. החיסרון בשיטה זו הוא שימש צורך להמתין כחצי דקה בכל פעם שמפעילים את היישום.

לשם כך יש לכתוב את הקוד הבא:

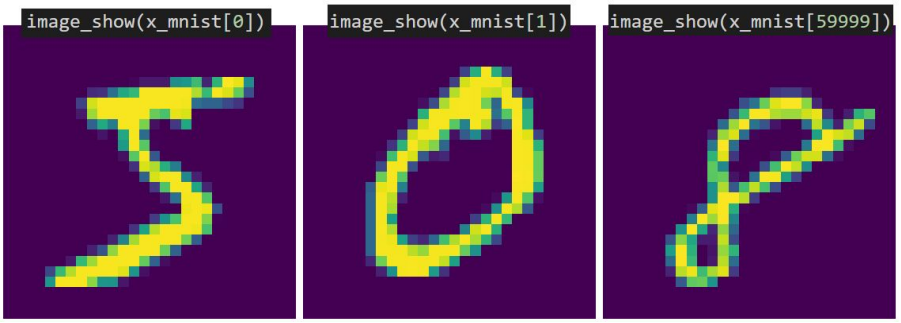
```
import sklearn.datasets

x, y = sklearn.datasets.fetch_openml('mnist_784', version=1, return_X_y=True)
```

הפעולה fetch_openml תוריד את הנתונים ישירות מאתר https://www.openml.org/d/554 לתוך 2 מערכים x ו- y כאשר x מכיל את התמונות וy מכיל את התגיות. בפעילות זו נעזר בפעולה fetch_openml כדי להוריד למחשב את קובץ הנתונים ואז נשמור אותו במחשב כדי לחסוך את הזמן שייקח כאשר נפעיל מספר פעמים את התוכנית.

נדגים תוכנית שמורידה מהאינטרנט את מערך התמונות ומציגה ממנו 3 תמונות:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml


def image_show(arr):

        p = (np.reshape(arr, (28, 28))).astype(np.uint8)

        plt.axis('off')

        plt.imshow(p)

        plt.show()


print("downloading file...")

x_mnist, y_mnist = fetch_openml("mnist_784", version=1, return_X_y=True, data_home=".")

image_show(x_mnist[0])

image_show(x_mnist[1])

image_show(x_mnist[59999])
```

נקבל את הפלט הבא:



כדי לשמור את מערך הנתונים במחשב לאחר ההורדה מהאינטרנט נכתוב את הקוד הבא:

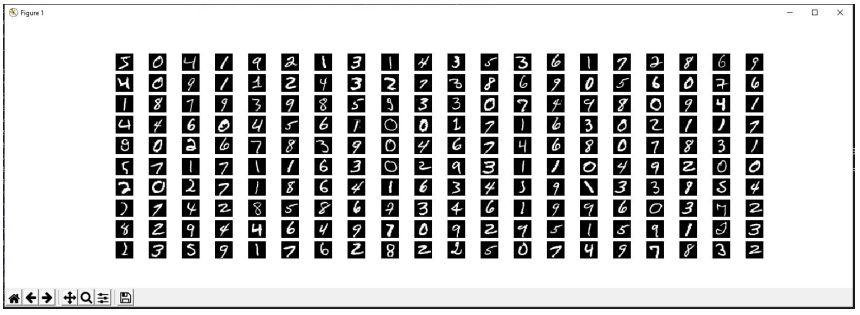
```
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from sklearn.datasets import fetch_openml


datapath = Path("mnist_784.npz")
if not(datapath.exists()):
        print("downloading file...")
        x_mnist, y_mnist = fetch_openml("mnist_784", version=1,
                                        return_X_y=True, data_home=".")
        x=np.array(x_mnist,dtype="u8")
        y=np.array(y_mnist,dtype="u8")
        np.savez(datapath,x=x,y=y)
        del x_mnist, y_mnist
print("Loading file...")
data = np.load(datapath)
print("\ntype: ", type(data))
print("\ntype: ", data.files)
```

```
x_mnist = data["x"]

y_mnist = data["y"]


plt.figure()

for i in range(200):

        plt.subplot(10,20,i+1)

        plt.axis("off")

        plt.imshow(x_mnist[i].reshape((28,28)),cmap="gray", vmin=0, vmax=255)

plt.show()
```

נקבל את הפלט הבא:

```
type:
 <class 'numpy.lib.npyio.NpzFile'>
files:
 ['x', 'y']
```

ובהמשך הפלט נקבל דוגמה של 200 התמונות הראשונות מתוך מערך הנתונים x_mnist



בקוד זה השתמשנו במחלקה Path מתוך הספרייה pathlib כדי לבדוק האם קיים במחשב המקומי קובץ מערך הנתונים.

```
if not(datapath.exists()):
```

במידה והקובץ לא קיים נעזר במחלקה fetch_openml כדי להוריד את הקובץ המכיל את התמונות מהאינטרנט

```
x_mnist, y_mnist = fetch_openml("mnist_784", version=1, return_X_y=True, data_home=".")
```

ואז לשמור את 2 המערכים x_mnist הכולל את התמונות ו- y_mnist הכולל את תיוג התמונות לקובץ בשם mnist_784.npz

```
np.savez(datapath,x=x,y=y)
```

הפעולה savez שומרת מערכי נתונים מסוג numpy.ndarray לקובץ בפורמט ייחודי בשם npz

למידע בנושא:

https://kite.com/python/docs/numpy.lib.npyio.NpzFile

אחזור מערכי הנתונים מהקובץ מתבצע על ידי ההוראות:

```
data = np.load(datapath)
x_mnist = data["x"]
y_mnist = data["y"]
```

השלב האחרון בארגון נתוני האימון יהיה לחלק את התמונות באופן הבא:

train_data - מערך הכולל 60000 תמונות לצורך שלב האימון

train_lbl - מערך הכולל 60000 מספרים בין 0 ל-9 המייצגים את המספרים שתמונות, לצורך שלב האימון

test_data - מערך הכולל 10000 תמונות לצורך שלב בדיקת המכונה

test_lbl - מערך הכולל 10000 מספרים בין 0 ל-9 המייצגים את המספרים שתמונות, לצורך בדיקת המכונה

כמו כן ראינו שכל תמונה בנוייה ממערך של 28*28 פיקסלים, כאשר כל פיקסל שמור כמספר בין 0 ל- 255.
על מנת להאיץ את תהליך הלמידה של הרשת ננרמל את הקלט כך שיתפלג בתחום 0-1

להלן קטע הקוד המסיים את שלב הכנת הנתונים:

```
x_mnist = x_mnist / 255
train_data, train_lbl = x_mnist[:60000], y_mnist[:60000]
test_data, test_lbl = x_mnist[60000:], y_mnist[60000:]
```

השלב האחרון בתרגיל שלנו יהיה ליצור את רשת הנוירונים כרשת הכוללת 784 מבואות, 50 נוירונים בשכבה פנימית ו- 10 נוירונים בשכבת המוצא.

```
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=20, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)
```

נאמן את המכונה על ידי הפעולה fit ונספק לה את מערך נתוני האימון:

```
mlp.fit(train_data, train_lbl)
```

לבסוף נבדוק את איכות המערכת ב- 2 דרכים:

הדרך הראשונה על ידי הפעולה score שבודקת את כמות ההצלחות ביחס למספר הדוגמאות ומחזירה לנו מספר בין 0 ל- 1

```
print("Training set accuracy: %f" % mlp.score(train_data, train_lbl))
print("Test set accuracy: %f" % mlp.score(test_data, test_lbl))
```

נקבל את הפלט הבא:

```
Training set score: 0.996883
Test set score: 0.972300
```

ניתן לראות שהמכונה הצליחה לסווג בהצלחה 99.67% ממערך נתוני האימון. ו- 97.23% מכלל נתוני הבדיקה. במילים אחרות המערכת טובה יותר בזיהוי תמונות שכבר ראתה מאשר בזיהוי תמונות שלא פגשה בהם בעבר. אם שגיאת הסיווג על סט האימון נמוכה משמעותי משגיאת האימון על סט המבחן הדבר עלול להעיד על התאמת יתר (over fitting)כמובן ניקח את הנתונים בפרופורציה כי המכונה הצליחה לסווג 9,723 תמונות שמעולם לא פגשה ונכשלה בסיווג 277 תמונות.

כמובן שאימון נוסף של המכונה יגרור שיפור בתוצאות. מספיק שנגדיל את מספר מחזורי האימון על ידי המאפיין max_iter מ- max_iter=20 ל- max_iter=50 נקבל את התוצאות הבאות:

```
Training set score: 1.000000
Test set score: 0.973100
```

כלומר 100% הצלחה בזיהוי מערך תמונות האימון ו- 97.31% בזיהוי תמונות ממערך הבדיקה (כלומר שיפור קטנה ביכולת הזיהוי של התמונות ממערך הבדיקה).

ניתן להמחיש יכולת המכונה על ידי הקוד הבא:

```
print("\nTest data: ",test_lbl[:30])
newp = mlp.predict(test_data[:30])
print ("\nPredict data: ",newp)
```

נקבל את הפלט הבא:

```
Test data:    [7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1]
Predict data: [7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1]
```

גם כאן נקבל התאמה בין נתוני הבדיקה לנתוני הסיווג של המוכנה.

להלן הקוד המלא של התרגיל לאימון מכונה לומדת לזיהוי ספרות הכתובות בכתב יד:

```
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from sklearn.datasets import fetch_openml
from sklearn.neural_network import MLPClassifier

datapath = Path("mnist_784.npz")
if not(datapath.exists()):
        print("downloading file...")
        x_mnist, y_mnist = fetch_openml("mnist_784", version=1,
                                        return_X_y=True, data_home=".")
        x=np.array(x_mnist,dtype="u8")
        y=np.array(y_mnist,dtype="u8")
        np.savez(datapath,x=x,y=y)
```

```
            del x_mnist, y_mnist
print("Loading file...")
data = np.load(datapath)
print("\ntype: ", type(data))
print("\ntype: ", data.files)
x_mnist = data["x"]
x_mnist = x_mnist / 255
y_mnist = data["y"]

train_data, train_lbl = x_mnist[:60000], y_mnist[:60000]
test_data, test_lbl = x_mnist[60000:70000], y_mnist[60000:70000]

plt.figure()
for i in range(200):
    plt.subplot(10,20,i+1)
    plt.axis("off")
    plt.imshow(train_data[i].reshape((28,28))*255,cmap="gray", vmin=0, vmax=255)
plt.show()

mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=20, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)

mlp.fit(train_data, train_lbl)
```

```
print("Training set score: %f" % mlp.score(train_data, train_lbl))

print("Test set score: %f" % mlp.score(test_data, test_lbl))


print("\nTest data: ",test_lbl[:30])

newp = mlp.predict(test_data[:30])

print ("\nPredict data: ",newp)
```

תרגיל

הוספת רעש לתמונות

https://debuggercafe.com/adding-noise-to-image-data-for-deep-learning-data-augmentation/

תרגיל:

נחזור לתרגיל ים/ישבה שבו סיווגנו תמונת נוף ים ותמונות נוף יבשה ונממש את המכונה תוך שימוש בספרייה sklearn.neural_network.

בתרגיל ניקח 20 תמונות מתוייגות חלקם ים וחלקם יבשה ונלמד את המחשב לבצע את הסיווג. בהמשך נציג למחשב 6 תמונות חדשות ונבדוק את יכולת המכונה.

פתרון:

```
from PIL import Image

import numpy as np

from sklearn.neural_network import MLPClassifier


#--------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)
```

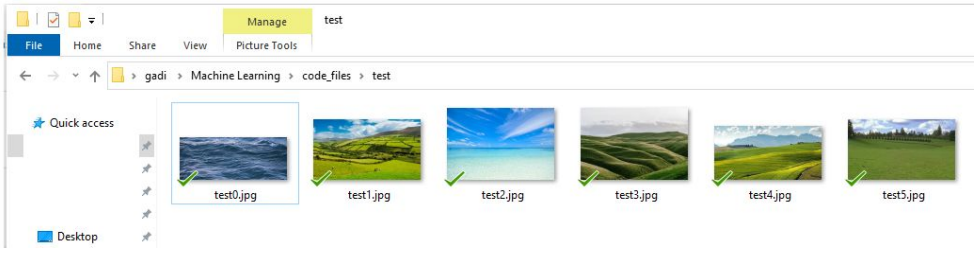
```
x1 = sea_array[:,1]

y1 = sea_array[:,2]

x2 = land_array[:,1]

y2 = land_array[:,2]

xtest = test_array[:,1]

ytest = test_array[:,2]


x = 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]

#--------End Preparing the data for machine learned----------------------


mlp = MLPClassifier(hidden_layer_sizes=(5,),max_iter=1000,

                    solver='sgd', verbose=10, random_state=1)


mlp.fit(data, lbl)

print("Training set score: %f" % mlp.score(data, lbl))

print("Test set score: %f" % mlp.score(data, lbl))


newp = mlp.predict(test_data)
```

```
print (newp)
```

נקבל את הפלט הבא (כאשר 0 זה ים ו-1 זה יבשה):

```
Training set score: 1.000000
Test set score: 1.000000
[0 1 0 1 1 1]
```

נבדוק את התוצאות מול מערך תמנות הבדיקה:



קיבלנו התאמה מלאה בין פלט המכונה לבין סדר התמונות המסופק למכונה. מכיוון שראינו בפעילות קודמת כיצד פרספטרון בודד מסוגל לבצע הפרדה זו, בהחלט אפשר לצפות ביצועים אלו מרשת של מספר פרספטרונים.

פעילות 14 - מימוש שער XOR על ידי מכונה לומדת עם ספריית Keras

Keras היא ספריית נוספת לפיתוח רשתות נוירונים מלאכותיות. הספרייה נכתבה בשפת Python כחלק מקוד פתוח. בשנת 2017 החליטו צוות מפתחי TensorFlow של גוגל להסתמך על הקוד הפתוח של Keras. משמעות הדבר היא שהספרייה Keras יושבת מעל הספריה של TensorFlow ומספקת ממשק גישה נוח לספרייה TensorFlow. לפי הערכתי TensorFlow מהווה את חוד החנית של הטכנולוגיה לפיתוח רשתות נוירונים.

בפעילות זו נתרגל כתיבת קוד תוך שימוש בכלי Keras לפיתוח ANN – Artificial Neural Network.

בדיקה ראשונית של הספריות tensorflow ו- keras

כדי לבדוק שהספריות tensorflow ו- keras מותקנות וזמינות לעבודה ניתן להריץ את הקוד הבא:

```
import keras

import tensorflow as tf

print(tf.__version__)

print(keras.__version__)
```

בפועל קוד זה מחזיר את גרסת הספריה הזמינה במחשב אך יותר מכך זה מדד טוב לכך שהספריות מוכנות וזמינות לעבודה.

נקבל את הפלט הבא:

```
2.3.0
2.4.3
```

תוכנית ראשונה ב- Keras למימוש שער XOR

נדגים את העקרונות בתכנות רשתות נוירונים תוך שימוש בדוגמא שכבר הכרנו לפני ולפנים.

המרכיב הבסיסי של רשת נוירונים מבוססת Keras היא model

```
from keras.models import Model
```

קיימים שני סוגים של מודלים:

- Sequential model - מודל סדרתי שבו קיים רצף לינארי של שכבות (כלומר כל מוצא של שכבה מחוברת למבוא של השכבה הבאה).
- Functional Model - מודל המבוסס על יכולת לבנות רשתות נוירונים שאין להם רצף לינארי, לדוגמה רשת הכוללת מספר מקורות קלט שונים, מספר מקורות פלט שונים במאפיינים שלהם או רשתות הכוללת קטעים חבויים החוזרים על עצמם ספר פעמים.

לצורך מימוש רשת נוירונים לסיווג XOR הגדרנו רשת סדרתית לינארית הכוללת 2 מבואות, 3 נוירונים בשכבה החבויה כך שבמוצא של כל אחד מהם פונקציית המעבר היא מסוג tanh כמו כן הגדרנו שכבת מוצא אחת הכוללת פונקציית מעבר מסוג sigmoid. לצורך כך נשתמש באובייקט danse המגדיר רשת fully connected.

```
model = Sequential()
model.add(Dense(3, input_dim=2))
model.add(Activation('tanh'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

כתבו את הקוד הבא והריצו אותו:

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation

model = Sequential()
model.add(Dense(3, input_dim=2))
model.add(Activation("tanh"))
model.add(Dense(1))
model.add(Activation("sigmoid"))

print(model.summary())
```

נקבל את הפלט הבא:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 3)                 9
_________________________________________________________________
activation (Activation)      (None, 3)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 4
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 13
Trainable params: 13
Non-trainable params: 0
```

מפלט התוכנית אנו למדים על מבנה הרשת הכוללת סה"כ 13 פרמטרים (מקדמי w ו b של פרספטרונים).

אופן חישוב הקשרים מתבצע כך:

מספר המוצאים * (מספר המבואות + 1)

שכבה חבויה: 3 * (2 +1) = 9

שכבת מוצא: 1 * (3+1) = 4

סה"כ 13

השלב הבא יהיה לזמן את הפעולה compile השייכת למחלקה model. פעולה זו מגדירה את מודל האימון של המכונה. נדגים זאת:

```
sgd = SGD(lr=0.1)          # type of optimization algorithm
model.compile(loss='binary_crossentropy', optimizer=sgd)
```

השלב האחרון יהיה הפעלת הפעולה שמבצעת את האימן בפועל.

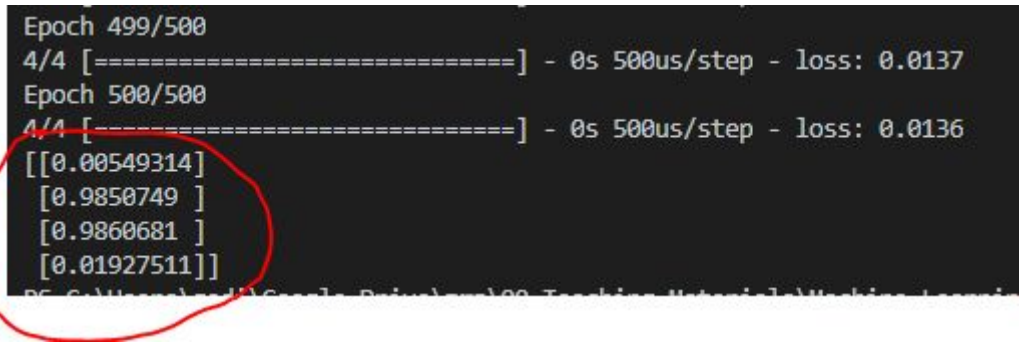
```
model.fit(X, y, batch_size=1, nb_epoch=500)
```

להלן קוד התוכנית המממש שער XOR על ידי מכונה לומדת תוך שימוש בספריה של keras לבניית רשת הנוירונים

```
from keras.models import Sequential
```

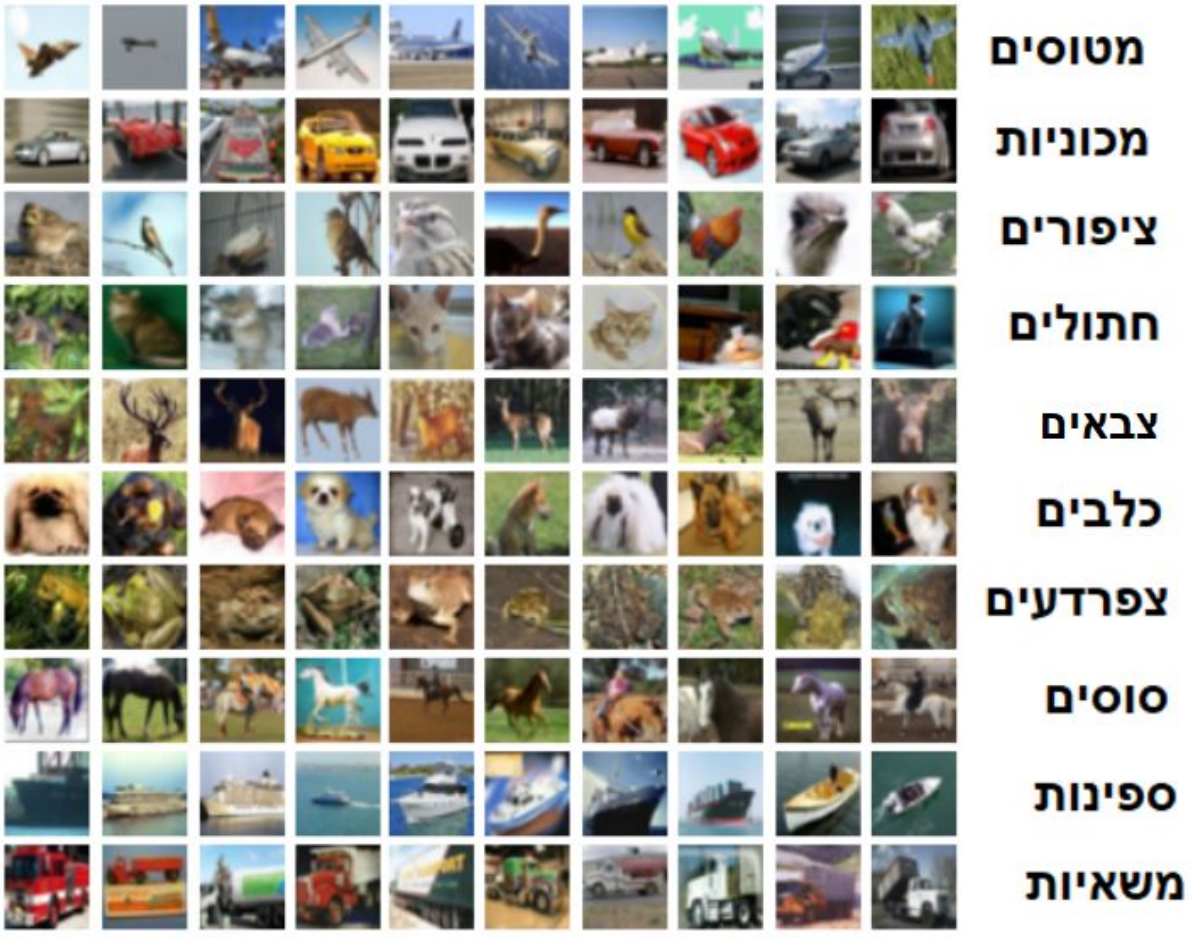
```
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import SGD
import numpy as np

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

model = Sequential()
model.add(Dense(3, input_dim=2))
model.add(Activation('tanh'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

sgd = SGD(lr=0.1)
model.compile(loss='binary_crossentropy', optimizer=sgd)

model.fit(X, y, batch_size=1, epochs=500)
print(model.predict(X))
```

פלט התוכנית יהיה:



פעילות 15 - רשת נוירונים מבוססת Keras לסיווג תוכן בתמונות

בפעילות זה נתרגל את היסודות בכתיבת מכונה לומדת מבוססת על רשת נוירונים כדי לבצע סיווג תמונות לפי תוכן מצולם. בפעילות זו נעשה שימוש במאגר התמונות CIFAR-10 הכולל 60000 תמונות ברזולוציה של 32x32 פיקסלים. כל תמונה מסווגת לאחת מ- 10 קטגוריות. באיור הבא ניתן לראות דוגמה של 10 תמונות בכל קטגוריה.



מערך התמונות שב- CIFAR-10 מחולק ל- 50000 תמונות לצורך אימון ועוד 10000 תמונות לצורך בדיקה.

הספרייה Keras כוללת פעולה להורדה ישירה של מאגר התמונות ישירות למחשב. נבחן קוד העושה שימוש בפעולה cifar10.load_data כדי להוריד את המאגר למחשב האישי שלכם:

```
from keras.datasets import cifar10
```

```
print('Loading data...')
(train_data, train_lbl), (test_data, test_lbl) = cifar10.load_data()
print(len(train_data), 'train data')
print(len(test_data), 'test data')
```

נקבל את הפלט הבא:

```
Loading data...
50000 train data
10000 test data
```

נבחן את המבנה של מערך התמונות על ידי שימוש בפעולה shape כפי שהכרנו אותה מוקדם יותר כאשר למדנו על NumPy Arrays

```
print("train_data shape:", train_data.shape)
print("train_lbl shape:", train_lbl.shape)
print("test_data shape:", test_data.shape)
print("test_lbl shape:", test_lbl.shape)
```

נקבל את הפלט הבא:

```
train_data shape: (50000, 32, 32, 3)
train_lbl shape: (50000, 1)
test_data shape: (10000, 32, 32, 3)
test_lbl shape: (10000, 1)
```

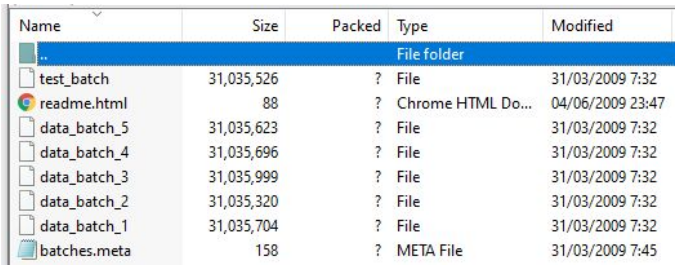
ניתן לראות שאכן כל תמונה מורכבת מ- 32x32 פיקסלים, כאשר כל פיקסל מורכב מ-3 מספרים red, green, blue. מערך הסיווגים לתמונות (lbl) מכיל מספר בודד בין 0 ל-9.

דרך נוספת לעבוד על מערך התמונות CIFAR10 הוא להוריד את הקבצים מהאינטרנט ולשמור אותם בתיקייה בשם CIFAR10dataset במחשב שלכם.

את הקבצים יש להוריד דרך הקישור הבא:

https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

נקבל קובץ אחד דחוס הכולל את הקבצים הבאים:



Name	Size	Packed	Type	Modified
..			File folder	
test_batch	31,035,526	?	File	31/03/2009 7:32
readme.html	88	?	Chrome HTML Do...	04/06/2009 23:47
data_batch_5	31,035,623	?	File	31/03/2009 7:32
data_batch_4	31,035,696	?	File	31/03/2009 7:32
data_batch_3	31,035,999	?	File	31/03/2009 7:32
data_batch_2	31,035,320	?	File	31/03/2009 7:32
data_batch_1	31,035,704	?	File	31/03/2009 7:32
batches.meta	158	?	META File	31/03/2009 7:45

יש להעתיק את הקבצים data_batch ו- text_batch לתוך התיקייה CIFAR10dataset . כל קובץ מכיל 10000 תמונות. כמו כן הקובץ batches.meta מכיל את שמות הקטגוריות שעל פיהם מתויגות התמונות. נבחן את תוכן הקובץ batches.meta על ידי הדוגמה באה:

```
import _pickle as pickle


f = open("CIFAR10dataset/batches.meta", 'rb')

meta = pickle.load(f, encoding='bytes')

print("category:", meta[b'label_names'])
```

נקבל את הפלט הבא:

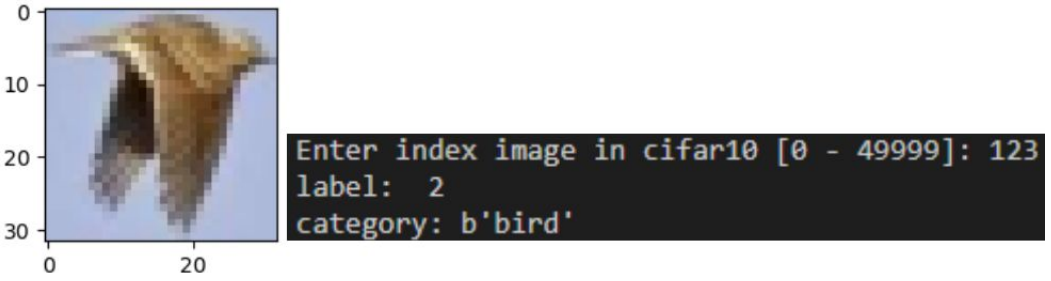
```
category: [b'airplane', b'automobile', b'bird', b'cat', b'deer', b'dog', b'frog', b'horse', b'ship', b'truck']
```

נעבור לחקור את חמשת הקבצים המכילים את תמונות האימון. ניתן לבחון כל אחת מ- 50000 התמונות על ידי הקוד הבא:

```
import _pickle as pickle

import numpy as np

import matplotlib.pyplot as plt


imageid = int(input("Enter index image in cifar10 [0 - 49999]: "))

batch = (imageid // 10000) + 1

idx = imageid - (batch-1)*10000
```

```
f1 = open("CIFAR10dataset/data_batch_" + str(batch), 'rb')
data = pickle.load(f1, encoding='bytes')
f2 = open("CIFAR10dataset/batches.meta", 'rb')
meta = pickle.load(f2, encoding='bytes')

im = data[b'data'][idx, :]
im_r = im[0:1024].reshape(32, 32)
im_g = im[1024:2048].reshape(32, 32)
im_b = im[2048:].reshape(32, 32)
img = np.dstack((im_r, im_g, im_b))

print("label: ", data[b'labels'][idx])
print("category:", meta[b'label_names'][data[b'labels'][idx]])
plt.imshow(img)
plt.show()
```

נקבל את הפלט הבא:



כאשר ייבאנו את הקבצים עם cifar10.load_data() התמונות התקבלו בגודל 32X32X3. כאשר ייבאנו את הקבצים מהדיסק המקומי הם נפתחים כאשר התמונות מיוצגות כוקטור באורך 3072 ולכן נדרש להמיר אותו לפורמט של תמונה בגודל 32X32X3.

תרגיל

כתבו קוד המציג תמונה אחת לבחירה מתוך קובץ הבדיקה test_batch. קובץ זה מכיל 10000 תמונות.

פתרון

```
mport _pickle as pickle

import numpy as np

import matplotlib.pyplot as plt


imageid = int(input("Enter index image in cifar10 test data [0 - 9999]: "))


f1 = open("CIFAR10dataset/test_batch", 'rb')

data = pickle.load(f1, encoding='bytes')

f2 = open("CIFAR10dataset/batches.meta", 'rb')

meta = pickle.load(f2, encoding='bytes')


im = data[b'data'][imageid, :]


im_r = im[0:1024].reshape(32, 32)

im_g = im[1024:2048].reshape(32, 32)

im_b = im[2048:].reshape(32, 32)

img = np.dstack((im_r, im_g, im_b))


print("label: ", data[b'labels'][imageid])

print("category:", meta[b'label_names'][data[b'labels'][imageid]])
```

```
plt.imshow(img)

plt.show()
```

נקבל את הפלט הבא:

להלן קוד תוכנה הכולל את השלבים הבאים:

1. יבוא הספריות שבהם נשתמש לאמן את המכונה.
2. הורדת מערך התמונות למחשב והתאמתם למכונה.
3. בניית המודל שעליו תעבוד המכונה.
4. הידר המודל.
5. אימון ובדיקת המכונה.
6. פלט גרפים המכילים מידע על האימון.

בדוגמה זו נעשה שימוש בשכבות נוירונים מסוג קונבולוציה. שכבות אלו שונות מהשכבות fully connected אותן ראינו עד כה והן משמשות כמסננים אדפטיביים עבור התמונות. שימוש בשכבות אלו נפוץ מאוד בעיבוד תמונה על מנת לצמצם את הנפח של הקלט. בנוסף, שכבות אלו מאפשרות לזהות אובייקטים בלי קשר למיקום בהן הם נמצאים בתמונה. שכבת pooling מצמצמת עוד יותר את נפח התמונה על ידי מסננים נוספים.

```
from keras.datasets import cifar10

from keras.utils import np_utils

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Flatten

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.optimizers import Adam

import matplotlib.pyplot as plt
```

```
# Download the photos and Tags to my computer and customize them
(train_data, train_lbl), (test_data, test_lbl) = cifar10.load_data()
print("train_data shape:", train_data.shape)
print(train_data.shape[0], "train samples")
print(test_data.shape[0], "test samples")
Y_train = np_utils.to_categorical(train_lbl, 10)
Y_test = np_utils.to_categorical(test_lbl, 10)
train_data = train_data.astype("float32")
test_data = test_data.astype("float32")
train_data /= 255
test_data /= 255

# Building the neuron network model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation="relu", padding="same", input_shape=(32,32,3)))
model.add(Conv2D(32, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation="relu", padding="same"))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
model.summary()
```

```
# Compile the neuron network model
model.compile(      loss="categorical_crossentropy",
                    optimizer=Adam(lr=1.0e-4),
                    metrics=["accuracy"])

# Neuron network training
results = model.fit(train_data, Y_train,
                    batch_size=128,
                    epochs=1,
                    validation_split=0.2,
                    validation_data= (test_data, Y_test),
                    verbose=1)

# Saving the Neuron network training into 2 files
model_json = model.to_json()
open("C:/saved_models/CIFAR10model.json", "w").write(model_json)
model.save_weights("C:/saved_models/CIFAR10weights.h5", overwrite=True)

# Creating the system learning graph
plt.plot(results.history["val_accuracy"])
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.legend(["train", "test"], loc="upper right")
plt.show()
```

נקבל את הפלט הבא:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
conv2d_2 (Conv2D)	(None, 30, 30, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 15, 15, 32)	0
dropout_1 (Dropout)	(None, 15, 15, 32)	0
conv2d_3 (Conv2D)	(None, 15, 15, 64)	18496
conv2d_4 (Conv2D)	(None, 13, 13, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 64)	0
dropout_2 (Dropout)	(None, 6, 6, 64)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 512)	1180160
dropout_3 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130

Total params: 1,250,858

מבנה הרשת הכולל מבוא הקולת תמונות ברזולציה של 28x28 פיקבלים. בהמשך מספר המרות מטריציוניות ובהמשך רשת הכוללת מעל מליון פרמטרים. סה"כ הרשת כולל 1250858 פרמטרים.

הערה:

שלב הלמידה של מכונה מסוג זה לוקח בין דקה למספר דקות כל אימון. כדי להגיע לתוצאות מספקות קבענו לבצע 100 אימונים. משמעות הדבר שזמן האימון הכללי של הרשת נמשך בין שעה למספר שעות בהתאם למאפייני המחשב ובמיוחד האם ואיזה סוג של כרטיס גרפי GPU קיים במחשב.

המחשב שלי לא כולל מאיץ גרפי GPU על כן זמן הלמידה לקח שלוש וחצי שעות!!!

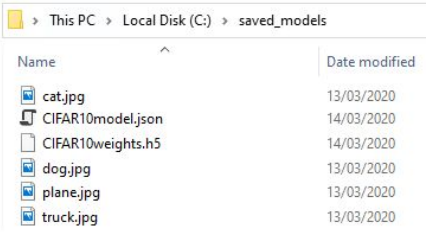
כדי לא לגרור את התוכנה שלנו לאימון של מעל שעה בכל פעם שנבצע בה שינויים. הוספתי בסוף התוכנית קוד השומר את מודל רשת הנוירונים בקובץ חיצוני בשם CIFAR10model.json ואת כלל מערך המשקלים בקובץ נוסף בשם CIFAR10weights.h5. את שני הקבצים נשמור בספרייה בשם saved_models בכונן C.

להלן הקוד:

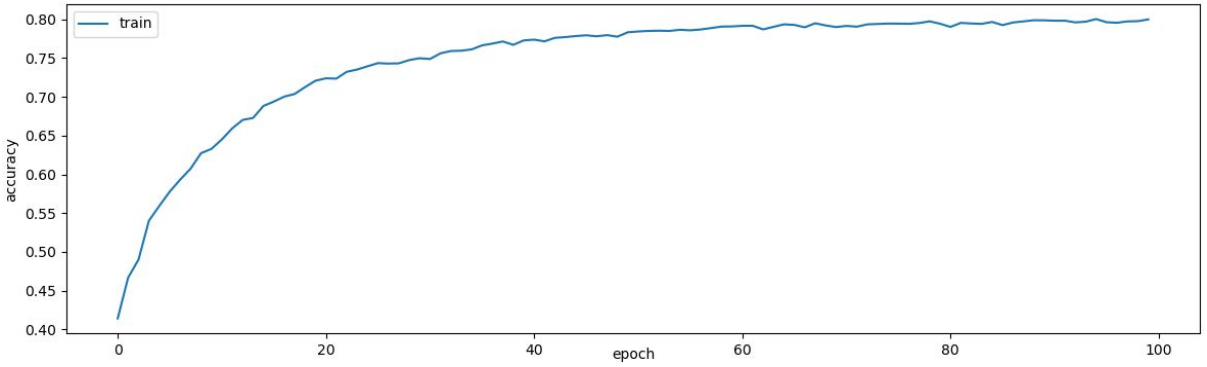
```
model_json = model.to_json()

open("C:/saved_models/CIFAR10model.json", "w").write(model_json)

model.save_weights("C:/saved_models/CIFAR10weights.h5", overwrite=True)
```

בשלב הבא נכתוב קוד הקורא את שני הקבצים בונה את רשת הנוירונים לפי תוכן קובץ CIFAR10model.json בהמשך מכוון את משקלי הרשת על פי תוכן קובץ CIFAR10weights.h5 ובכך אנו מדלגים על הצורך באימונים חוזרים ונשנים בכל פעם שמריצים את התוכנית. מידע מעמיק יותר בנושא חשוב

זה נלמד בהמשך המדריך בפעילות 18 בנושא גיבוי ושחזור מערך המשקלים של רשת נוירונים. להלן דוגמה לתוכן הספרייה saved_models:



פלט תוצאות האימון בגרף:



ניתן לראות שלאחר 100 אימונים כאשר כל אימון מבצע עיבוד של 50000 תמונות אימון ועוד 10000 תמונות בדיקה הגענו במערכת המסוגלת לסווג תוכן בתמונות ברמה של מעל 80 אחוז הצלחה. כמו כן ניתן לראות תוצאות דומות התקבלו כבר לאחר 60 אימונים.

בדיקת המכונה על ידי תמונות חדשות שלא מתוך מאגר התמונות

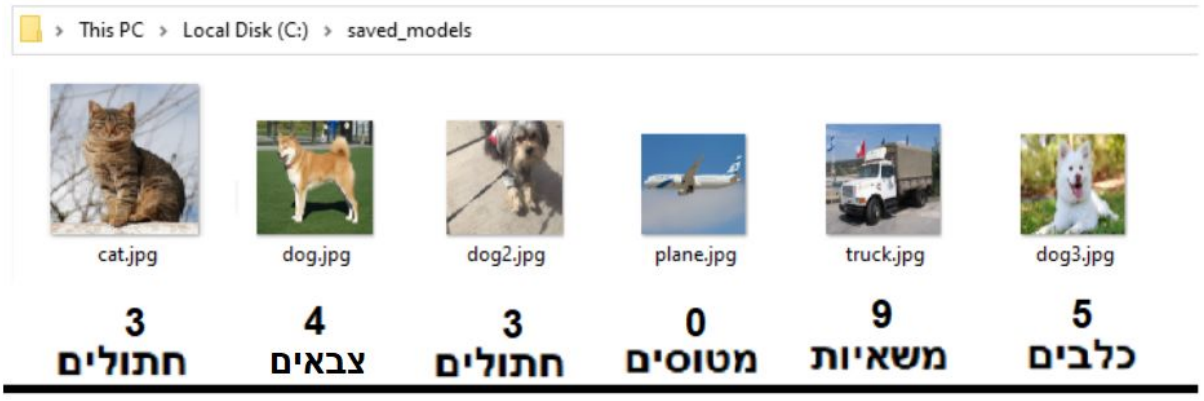
נכתוב קוד הקורא את 2 הקבצים CIFAR10model.json ו- CIFAR10weights.h5 כדי לבנות מחדש את מודל המכונה. אז נבדוק את יכולות המכונה על תמונות חדשות מהאינטרנט. להלן קוד התוכנית:

```
from keras.models import model_from_json

from keras.optimizers import Adam

from keras.preprocessing import image

# Load the Neuron network training from 2 files

model_architecture = "C:/saved_models/CIFAR10model.json"

model_weights = "C:/saved_models/CIFAR10weights.h5"
```

```
model = model_from_json(open(model_architecture).read())
model.load_weights(model_weights)
# Lode the image from my computer and do customize
MyPIC = image.load_img('C:/saved_models/dog.jpg', target_size=(32,32))
MyPIC = image.img_to_array(MyPIC)
MyPIC = MyPIC.reshape((1,) + MyPIC.shape)
MyPIC = MyPIC/255.
# Compile the neuron network model
model.compile(     loss="categorical_crossentropy",
                   optimizer=Adam(lr=1.0e-4),
                   metrics=["accuracy"])
# Predicting the image
predictions = model.predict_classes(MyPIC)
print(predictions)
```

נדגום מהאינטרנט מספר תמונות לבדיקה ונבחן את התוצאות.

להלן תוצאות הבדיקה:



ניתן ללמוד שהמכונה מזהה את תמונת המטוס החתול והמשאית אך מתקשה לזהות את הכלב. עושה הרושם שכלב קטן במימדים של חתול מזוהה כחתול וכלב עם רגליים ארוכות מזוהה כצבי. רק כלב שהראש שלו מצולם במרכז התמונה מזוהה נכון.

פעילות 16 - רשת נוירונים מבוססת Keras לסיווג הודעות טקסט

בפעילות זה נתרגל את היסודות בסיווג טקסט. נבין כיצד מבצעים עיבוד להודעות טקסט, כיצד מייצגים מילים במכונה לומדת, נלמד כיצד להכניס את רצף המילים לרשת נוירונים וכיצד מערכת לומדת מבצעת סיווג תוכן למשפטים.

כפי שכבר למדנו בפעיליות הקודמות חלק מרכזי בפיתוח מכונה לומדת בכלל ורשת נוירונים מלאכותית בפרט הוא ארגון מערך הנתונים. בפעילות זו נשתמש בבסיס נתונים בשם IMDB הכולל 50000 ביקורות על סרטים באנגלית מתוייגות באופן בינארי לביקורות חיוביות (1 לוגי) וביקורות שליליות (0 לוגי).

כל הביקורות במערך הנתונים IMDB מקודדות כרצף של אינדקסים לפי מילים. כאשר כל מילה מקבלת מספר שלם המייצג את אינדקס המילה במילון. לצורך פשטות, המילון מסודר לפי שכיחות כך שהמילה הראשונה במילון מופיעה בשכיחות גבוהה יותר בנתונים מאשר השניה וכן הלאה. כלומר המילה שמופיעה הכי הרבה פעמים במערך הנתונים תקודד למספר 1, המילה שמקודדת כמספר 2 היא במקום השני במספר ההופעות שלה במערך הנתונים וכך הלאה.

מערך הנתונים IMDB מחולק ל-2 קבוצות זהות, 25000 ביקורות מתויגות לצורך שלב האימון של המכונה ו-25000 ביקורות לשלב הבדיקת של המכונה.

הסבר על המאגר ניתן למצוא פה:

https://keras.io/api/datasets/imdb/

מערך הנתונים נוצר בשנת 2011 על ידי צוות חוקרים מאוניברסיטת סטנפורד.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

כדי לעבוד עם מערך הנתונים IMDB נשתמש בפעולה imdb.load_data כדי להוריד את קובץ הנתונים ישירות מהאינטרנט. נדגים זאת על ידי הקוד הבא:

```
from keras.datasets import imdb

(train_data, train_lbl), (test_data, test_lbl) = imdb.load_data(num_words=10000)

print(len(train_data), 'train data')

print(len(test_data), 'test data')
```

נקבל את הפלט הבא:

```
25000 train data
25000 test data
```

מפלט התוכנית ניתן ללמוד שהפעולה imdb.load_data הורידה 25000 הודעות אימון ו- 25000 הודעות בדיקה.

נבחן כיצד נראית הודעה:

```
from keras.datasets import imdb

(train_data, train_lbl), (test_data, test_lbl) = imdb.load_data(num_words=10000)

print(train_data[123])
```

נקבל את הפלט הבא:

```
 [1, 307, 5, 1301, 20, 1026, 2511, 87, 2775, 52, 116, 5, 31, 7, 4, 91, 1220, 102, 13, 28, 110, 11, 6, 137, 1
3, 115, 219, 141, 35, 221, 956, 54, 13, 16, 11, 2714, 61, 322, 423, 12, 38, 76, 59, 1803, 72, 8, 2, 23, 5, 9
67, 12, 38, 85, 62, 358, 99]
```

נו טוב, מה כבר אפשר להבין מאוסף המספרים הזה. למעשה ניתן להבין שכל מספר הוא אינדקס של המילה במילון של המאגר.

נבחן את 100 המילים השכיחות במערך:

```
from keras.datasets import imdb

from heapq import nsmallest


index = imdb.get_word_index()

index = {k:(v+3) for k,v in index.items()}

index["<PAD>"] = 0

index["<START>"] = 1

index["<UNK>"] = 2


MostPopular = nsmallest(100, index, key = index.get)
```

```
for val in MostPopular:
    print(val, ":", index.get(val))
```

נקבל את הפלט הבא:

```
<PAD> : 0       you : 25      there : 50    we : 75
<START> : 1     are : 26      what : 51     much : 76
<UNK> : 2       his : 27      good : 52     been : 77
the : 4         have : 28     more : 53     bad : 78
and : 5         he : 29       when : 54     get : 79
a : 6           be : 30       very : 55     will : 80
of : 7          one : 31      up : 56       do : 81
to : 8          all : 32      no : 57       also : 82
is : 9          at : 33       time : 58     into : 83
br : 10         by : 34       she : 59      people : 84
in : 11         an : 35       even : 60     other : 85
it : 12         they : 36     my : 61       first : 86
i : 13          who : 37      would : 62    great : 87
this : 14       so : 38       which : 63    because : 88
that : 15       from : 39     only : 64     how : 89
was : 16        like : 40     story : 65    him : 90
as : 17         her : 41      really : 66   most : 91
for : 18        or : 42       see : 67      don't : 92
with : 19       just : 43     their : 68    made : 93
movie : 20      about : 44    had : 69      its : 94
but : 21        it's : 45     can : 70      then : 95
film : 22       out : 46      were : 71     way : 96
on : 23         has : 47      me : 72       make : 97
not : 24        if : 48       well : 73     them : 98
you : 25        some : 49     than : 74     too : 99
```

נבחן את 10 המילים הנדירות במערך:

```
from keras.datasets import imdb
from heapq import nlargest

index = imdb.get_word_index()
index = {k:(v+3) for k,v in index.items()}
index["<PAD>"] = 0
index["<START>"] = 1
index["<UNK>"] = 2

RareWords = nlargest(10, index, key = index.get)
for val in RareWords:
```

```
print(val, ":", index.get(val))
```

נקבל את הפלט הבא:

```
'l' : 88587
voorhees' : 88586
artbox : 88585
copywrite : 88584
pipe's : 88583
wheelers : 88582
sics : 88581
transacting : 88580
chicatillo : 88579
ev : 88578
```

להלן קוד המתרגם הודעת טקסט כך שניתן יהיה לקרוא אותה:

```
from keras.datasets import imdb


(train_data, train_lbl), (test_data, test_lbl) = imdb.load_data(num_words=10000)


index = imdb.get_word_index()

index = {k:(v+3) for k,v in index.items()}

index["<PAD>"] = 0

index["<START>"] = 1

index["<UNK>"] = 2


id_to_word = {value:key for key,value in index.items()}

print(" ".join(id_to_word[id] for id in train_data[123] ))
```

נקבל את הפלט הבא:

```
<START> beautiful and touching movie rich colors great settings good acting and one of the most charming
movies i have seen in a while i never saw such an interesting setting when i was in china my wife liked i
t so much she asked me to <UNK> on and rate it so other would enjoy too
```

התג <UNK> מציין שהמילה הנ"ל לא נמצאת באחד מ-10000 המילים הפופולריות ועל פי קוד התוכנית שכתבנו (imdb.load_data(num_words=10000)) היא לא חלק ממאגר המילים שירד.

נבחן את פלט התוכנית על ידי מבט על המילה השלישית and. האינדקס של המילה במילון הוא 5 ואכן במקום השלישי קיימת המילה and.

לצורך הכנת הקלט למכונה לומדת נקודד את הטקסט בקידוד הידוע בשם bag of words. זהו מערך בגודל קבוע שבה בכל מיקום יש 1 אם המילה קיימת במשפט ו-0 אם המילה לא קיימת במשפט.

להלן קטע קוד הממיר מערך מספרים המייצגים מילים במשפט בבסיס הנתונים למערך אינדקסים בגודל קבוע בתצורת bag of words.

לצורך הדגמה פלט התוכנית נגדיר מערך אינדקסים בגודל של 40 מקומות. כלומר רק 40 המילים הפופולריות ביותר יקבלו ערך

```
import numpy as np
from keras.datasets import imdb
from nltk import word_tokenize
import string


def Preparing_string(text_string, dimension = 40):
    text_string = text_string.lower()
    table = str.maketrans(dict.fromkeys(string.punctuation))
    text_string = text_string.translate(table)
    word2index = imdb.get_word_index()
    test=[]
    for word in word_tokenize(text_string):
        test.append(word2index[word])
    print(text_string)
    print(test)
    out = np.zeros(dimension)
```

```
for _ , sequence in enumerate(test):
    if sequence < dimension:
        out[sequence] = 1
print ("\nOutput:",out)
```

Preparing_string("First off, this is NOT a war film. It is a movie about the bond of men in war. It is by far the best movie I've seen in a very, very long time. I had high expectations and was not disappointed. At first I was eager to see the one shot idea Sam Mendes went into this with but, after awhile, I stopped paying attention to that. While everything about the movie was well done I was so caught up in the two central characters that nothing else mattered. I will watch this again and again.")

נקבל את הפלט הבא:

```
first off this is not a war film it is a movie about the bond of men in war it i
ted at first i was eager to see the one shot idea sam mendes went into this with
as so caught up in the two central characters that nothing else mattered i will


[83, 122, 11, 6, 21, 3, 322, 19, 9, 6, 3, 17, 41, 1, 1645, 4, 346, 8, 322, 9, 6,
 13, 4508, 5, 64, 1, 28, 321, 323, 1281, 11748, 432, 80, 11, 16, 18, 100, 5233,
2, 12, 161, 331, 12943, 10, 77, 103, 11, 171, 2, 171]


Output: [0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 0.
 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0.]
```

נבנה את המכונה שלנו כך שתקבל מערך אינדקסים (אפסים ואחדים) בגודל של 10000 ולא 40 כמו בדוגמה. משמעות הדבר היא שמצד אחד המערכת מסוגלת לקבל מגוון של עד 10000 מילים במשפט. מצד שני נקבל מכונה לומדת הכוללת מעל חצי מיליון פרמטרים.

להלן קוד המכונה הלומדת:

```
import numpy as np
from keras import models
from keras import layers
from keras.datasets import imdb
TOP_WORDS = 10000
```

```
def Convert_to_vectors(data_list, dimension = TOP_WORDS):
    out = np.zeros((len(data_list), dimension))
    for i, sequence in enumerate(data_list):
        out[i, sequence] = 1
    return out


(train_data, train_lbl), (test_data, test_lbl) = imdb.load_data(num_words=TOP_WORDS)
train_data = Convert_to_vectors(train_data)
test_data = Convert_to_vectors(test_data)

model = models.Sequential()
model.add(layers.Dense(50, activation = "relu", input_shape=(TOP_WORDS, )))
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dense(1, activation = "sigmoid"))
model.summary()

model.compile(
            optimizer = "adam",
            loss = "binary_crossentropy",
            metrics = ["accuracy"]
```

```
        )


results = model.fit(
                train_data, train_lbl,
                epochs= 5,
                batch_size = 10,
                validation_data = (test_data, test_lbl)
                )


print("Test-loss:", results.history["loss"])
print("Test-Accuracy:", results.history["accuracy"])
```

נקבל את הפלט הבא:

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 50)                500050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                2550
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
=================================================================
Total params: 505,201
```

ניתן ללמוד שהמכונה כוללת 10000 מבואות, מבוא אחד לכל מילה, 2 שכבות פנימיות הכוללות 50 נוירונים כל אחת ומוצא אחד. סה"כ מעל חמישים מיליון פרמטרים.

הפעולה model.fit מאמנת את המכונה על ידי 25000 נתונים ב-5 סבבי אימון.

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 47s 2ms/step - loss: 0.3377 - accuracy: 0.8561 - val_loss: 0.2822 - val_accuracy: 0.8828
Epoch 2/5
25000/25000 [==============================] - 48s 2ms/step - loss: 0.2002 - accuracy: 0.9206 - val_loss: 0.2998 - val_accuracy: 0.8761
Epoch 3/5
25000/25000 [==============================] - 50s 2ms/step - loss: 0.1337 - accuracy: 0.9485 - val_loss: 0.3519 - val_accuracy: 0.8706
Epoch 4/5
25000/25000 [==============================] - 46s 2ms/step - loss: 0.0929 - accuracy: 0.9666 - val_loss: 0.4625 - val_accuracy: 0.8728
Epoch 5/5
25000/25000 [==============================] - 45s 2ms/step - loss: 0.0680 - accuracy: 0.9763 - val_loss: 0.5331 - val_accuracy: 0.8687
Test-loss: [0.3376910109914839, 0.20015975978411735, 0.1337492174546933, 0.0929183293148737, 0.06798984598161041]
Test-Accuracy: [0.85608, 0.92064, 0.94852, 0.9666, 0.97632]
```

לאחר שלב האימון נקבל מערכת הפועלת בדיוק של כ- 97.6 אחוז.

כדי לבדוק את המערכת עם נתונים עתידיים, כלומר על ידי האפשרות לכתוב ביקורת חדשה ולבדוק אותה על המכונה, נכתוב את הפעולה הבא:

```
def Preparing_string(text_string, dimension = 1000):
    text_string = text_string.lower()
    table = str.maketrans(dict.fromkeys(string.punctuation))
    text_string = text_string.translate(table)
    word2index = imdb.get_word_index()
    test=[]
    for word in word_tokenize(text_string):
        test.append(word2index[word])
    out = np.zeros(dimension)
    for _ , sequence in enumerate(test):
        if sequence < dimension:
            out[sequence] = 1
    return out
```

הפעולה Preparing_string מקבלת כקלט מחרוזת טקסט. הפעולה מבצעת את הפעולות הבאות:

1. פעולה שעוברת על המילים של מחרוזת וממירה את כל האותיות לקטנות.
2. הסרת כל סימני הפיסוק.

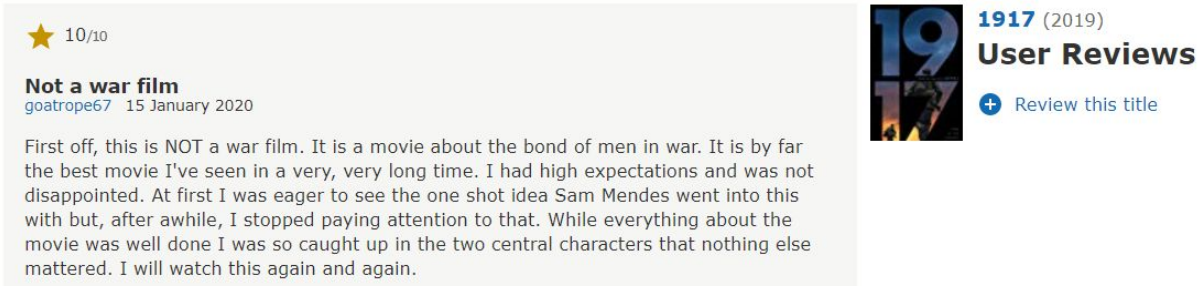
3. המרת המילים למספר האינדקס של כל מילה (במידה והאינדקס של המילה קטן-dimension ערך המילה יהיה 1)

4. המרת המספרים לאינדקס המיקום של כל מילה. (במדומה למערך אינדקסים)

נדגים את השלבים תוך שימוש בנתונים הבאים:

נבחן ביקורת מאתר:

https://www.imdb.com/title/tt8579674/reviews?ref_=tt_urv


10/10

Not a war film
goatrope67 15 January 2020

First off, this is NOT a war film. It is a movie about the bond of men in war. It is by far the best movie I've seen in a very, very long time. I had high expectations and was not disappointed. At first I was eager to see the one shot idea Sam Mendes went into this with but, after awhile, I stopped paying attention to that. While everything about the movie was well done I was so caught up in the two central characters that nothing else mattered. I will watch this again and again.

1917 (2019)
User Reviews
Review this title

המחרוזת המקורית:

```
First off, this is NOT a war film. It is a movie about the bond of men in war. It is by far the best movie I've seen in a very, very long time. I had high expectations and was not disappointed. At first I was eager to see the one shot idea Sam Mendes went into this with but, after awhile, I stopped paying attention to that. While everything about the movie was well done I was so caught up in the two central characters that nothing else mattered. I will watch this again and again.
```

המחרוזת לאחר המרה לאותיות קטנות והסרת סימני הפיסוק:

```
first off this is not a war film it is a movie about the bond of men in war it is by far the best movie ive seen in a very very long time i had high expectations and was not disappointed at first i was eager to see the one shot idea sam mendes went into this with but after awhile i stopped paying attention to that while everything about the movie was well done i was so caught up in the two central characters that nothing else mattered i will watch this again and again
```

המרת מילים למספרים:

```
[83, 122, 11, 6, 21, 3, 322, 19, 9, 6, 3, 17, 41, 1, 1645, 4, 346, 8, 322, 9, 6, 31, 227, 1, 115, 17, 18778, 107, 8, 3, 52, 52, 193, 55, 10, 66, 309, 1395, 2, 13, 21, 682, 30, 83, 10, 13, 4508,
```

```
5, 64, 1, 28, 321, 323, 1281, 11748, 432, 80, 11, 16, 18, 100, 5233, 10, 2227, 2645, 689, 5,
12, 134, 282, 41, 1, 17, 13, 70, 221, 10, 13, 35, 1056, 53, 8, 1, 104, 1372, 102, 12, 161, 331,
12943, 10, 77, 103, 11, 171, 2, 171]
```

יצירת מערך אינדקסים:

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

להלן קוד התוכנית הסופית:

```
import numpy as np
from keras import models
from keras import layers
from keras.datasets import imdb
from nltk import word_tokenize
import string
TOP_WORDS = 10000


def Preparing_string(text_string, dimension = TOP_WORDS):
    text_string = text_string.lower()
    table = str.maketrans(dict.fromkeys(string.punctuation))
    text_string = text_string.translate(table)


    word2index = imdb.get_word_index()
    test=[]
    for word in word_tokenize(text_string):
        test.append(word2index[word])
```

```
    results = np.zeros(dimension)
    for _ , sequence in enumerate(test):
        if sequence < dimension:
            results[sequence] = 1

    print("\nOriginal string:", text_string,"\n")
    print("\nIndex conversion:", test,"\n")
    results = np.reshape(results,(1, TOP_WORDS))
    print("\nConvert to vectors:", results,"\n")
    return results

def vectorize(sequences, dimension = TOP_WORDS):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1
    return results

(training_data, training_targets), (testing_data, testing_targets) =
imdb.load_data(num_words=TOP_WORDS)
data = np.concatenate((training_data, testing_data), axis=0)
targets = np.concatenate((training_targets, testing_targets), axis=0)

data = vectorize(data)
targets = np.array(targets).astype("float32")
```

```
test_x = data[:10000]

test_y = targets[:10000]

train_x = data[10000:]

train_y = targets[10000:]

model = models.Sequential()

model.add(layers.Dense(50, activation = "relu", input_shape=(TOP_WORDS, )))

model.add(layers.Dropout(0.3, noise_shape=None, seed=None))

model.add(layers.Dense(50, activation = "relu"))

model.add(layers.Dropout(0.2, noise_shape=None, seed=None))

model.add(layers.Dense(50, activation = "relu"))

model.add(layers.Dense(1, activation = "sigmoid"))

model.summary()


model.compile(

        optimizer = "adam",

        loss = "binary_crossentropy",

        metrics = ["accuracy"]

        )


results = model.fit(

            train_x, train_y,

            epochs= 10,

            batch_size = 500,

            validation_data = (test_x, test_y)
```

```
)


data_string = Preparing_string("First off, this is NOT a war film. It is a movie about the bond of men in war. It is by far the best movie I've seen in a very, very long time. I had high expectations and was not disappointed. At first I was eager to see the one shot idea Sam Mendes went into this with but, after awhile, I stopped paying attention to that. While everything about the movie was well done I was so caught up in the two central characters that nothing else mattered. I will watch this again and again.")

print("predict:",model.predict(data_string))

print("predict_classes:",model.predict_classes(data_string))

print("------1 is Good-------\n")

data_string = Preparing_string("I don't feel like I know the characters at all. I have no idea why the two soldiers were friends or what they had been through together. The cinematography tried so hard to make this an emotional shocking movie that it had the opposite effect. War scenes with gratuitous up close views of corpses and body parts that don't add anything to the story got old quick.")

print("predict:",model.predict(data_string))

print("predict_classes:",model.predict_classes(data_string))

print("------0 is Bad-------\n")
```

להלן דוגמה לפלט תוכנית המדגים סיווג של 5 ביקורות: 3 טובות ו-2 גרועות מתוך לסרט "1917":

הערה: הפלט המוצג להלן הוא עבור דוגמאות טקסט אחרות מאלו המופיעות בקוד מעלה.

```
Original string: first off this is not a war film it is a…….

predict: [[0.5612222]]

predict_classes: [[1]]

------1 is Good-------


Original string: one of the best films …...
```

predict: [[0.99998033]]

predict_classes: [[1]]

------1 is Good-------

Original string: i felt dirty i felt tired i felt hungry ……..

predict: [[0.5503042]]

predict_classes: [[1]]

------1 is Good-------

Original string: i dont feel like i know the characters at …….

predict: [[0.06769704]]

predict_classes: [[0]]

------0 is Bad-------

Original string: predictable and horrendous the acting was terrible …..

predict: [[0.21110523]]

predict_classes: [[0]]

------0 is Bad-------

קיבלנו התאמה מלאה בין תוכן הביקורת לבין יכולת המכונה לזהות אם מדובר בביקורת חיובית או שלילית.

פעילות 17 - סיווג פריטי לבוש בתמונות תוך שימוש keras

בדומה לפרק 15, נבנה בפרק זה מכונה לומדת לסיווג תמונות. רמת המורכבות של הפתרון בפרק זה גבוהה יותר מרמת המורכבות של הפתרון בפרק 15. בפרק זה נשתמש ברשתות מסוג קונבולוציה ובנוסף נפעיל שכבת softmax שכבה אחרונה ברשת. בפעילות זו נעשה שימוש במסד נתונים בשם Fashion MNIST dataset הכולל אלפי תמונות של פריטי לבוש מתוייגים. להלן דוגמה לחלק מהתמונות במסד הנתונים:

נעזר בספריה keras היושבת כמעטפת מעל TensorFlow כך שתעזור לנו לבנות מכונה לומדת המבוססת על רשת נוירונים מלאכותית ANN המסוגלת לזהות פריטי לבוש.

מקורות:

https://github.com/tensorflow/docs/blob/master/site/en/tutorials/keras/classification.ipynb

https://www.tensorflow.org/tutorials/keras/classification

https://www.tensorflow.org/install/pip?lang=python3

https://www.tensorflow.org/install/pip?lang=python3

בדיקת ההתקנה

נכתוב את הקוד הבא:

```
import tensorflow as tf

import keras

print("tensorflow version: " , tf.version.GIT_VERSION, tf.version.VERSION)

print("keras version:" ,keras.__version__)
```

נקבל כפלט את המסך הבא:

```
Using TensorFlow backend.
tensorflow version:  v2.1.0-rc2-17-ge5bf8de410 2.1.0
keras version: 2.3.1
```

יבוא מסד הנתונים Fashion MNIST dataset

נעזר בפעולה load_data כדי להוריד למחשב את מסד הנתונים Fashion MNIST dataset. קובץ המסד מכיל 70000 תמונות בגודל של 28 פיקסלים על 28 פיקסלים לכל תמונה. כמו כן קובץ תמונות מתויגות ל- 10 קטגוריות שונות והם:

```
lbls = ['T-shirt/top',
        'Trouser',
        'Pullover',
        'Dress',
        'Coat',
        'Sandal',
        'Shirt',
        'Sneaker',
        'Bag',
        'Ankle boot']
```

ניתן להוריד את קבצי התמונות באופן ידני דרך הקישור הבא:

https://github.com/zalandoresearch/fashion-mnist

Name	Content	Examples	Size	Link	MD5 Checksum
`train-images-idx3-ubyte.gz`	training set images	60,000	26 MBytes	Download	`8d4fb7e6c68d591d4c3dfef9ec88bf0d`
`train-labels-idx1-ubyte.gz`	training set labels	60,000	29 KBytes	Download	`25c81989df183df01b3e8a0aad5dffbe`
`t10k-images-idx3-ubyte.gz`	test set images	10,000	4.3 MBytes	Download	`bef4ecab320f06d8554ea6380940ec79`
`t10k-labels-idx1-ubyte.gz`	test set labels	10,000	5.1 KBytes	Download	`bb300cfdad3c16e7a12a480ee83cd310`

קבצי הנתונים מחולקים ל- 60000 תמונות מתויגות שישמשו לאימון הרשת ועוד 10000 תמונות מתויגות לבדיקה לאחר האימון.

הקוד הבא מוריד את קבצי הנתונים ישירות מהאינטרנט למחשב ומציג את התמונה הראשונה במערך תמונות:

```
from tensorflow import keras

import numpy as np


fashion_mnist = keras.datasets.fashion_mnist


(train_data, train_lbl), (test_data, test_lbl) = fashion_mnist.load_data()

lbls = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']


print("\n",np.where(train_data[0] > 128, 8, 1))

print(lbls[train_lbl[0]])
```

פלט התוכנית יראה כך:

השתמשנו בהוראה הבא כדי לשרטט את התמונה ישירות על גבי מסך הטרמינל:

```
print("\n",np.where(train_data[0] > 128, 8, 1))
```

כפי שלמדנו בפעילות 2 בנושא מערכים תחת NumPy Arrays שניתן לבצע פעולות לוגיות מתמטיות ישירות על מבני נתונים שלמים. ההוראה המתוארת ממירה את הערכים המרכיבים את התמונה הראשונה במערך כאשר כל ערך הגדול מ-128 יקבל את הערך 8 וכל ערך קטן או שווה ל-128 יקבל את הערך 0. זאת במטרה ליצור דמות יחסית ברורה על מסך הטרמינל.

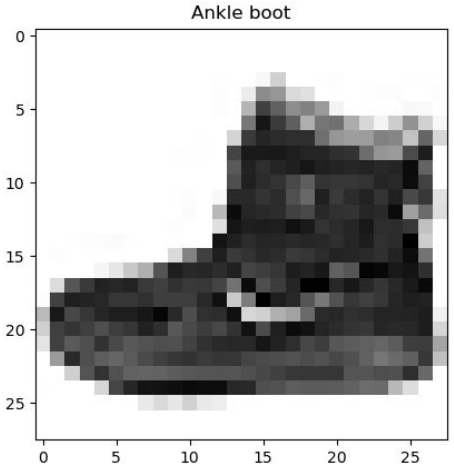
כמובן שניתן להיעזר בספריה matplotlib כדי להציג את התמונה בכלי גרפי. להלן הקוד:

```
from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt


fashion_mnist = keras.datasets.fashion_mnist

lbls = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
print("\ntrain images shape: ",train_data.shape)
print("\ntrain labels shape: ",train_lbl.shape)
print("\ntest images shape: ",test_data.shape)
print("\ntest labels shape: ",test_lbl.shape)
plt.figure()
plt.imshow(train_data[0], cmap='Greys')
plt.title(lbls[train_lbl[0]])
plt.show()
```

פלט התוכנית יראה כך:

```
train images shape:  (60000, 28, 28)
train labels shape:  (60000,)
test images shape:  (10000, 28, 28)
test labels shape:  (10000,)
```



מפלט התוכנית ניתן ללמוד את הדברים הבאים:

1. גודל כל תמונה הוא 28X28 פיקסלים.
2. קובץ הנתונים לאימון מכיל 60000 תמונות.
3. קובץ הבדיקה מכיל 10000 תמונות.
4. קובץ התגים מכיל מספרים בין 0 ל-9 בהתאם למתואר בתמונה על פי המפתח הבא:
5. אכן התמונה הראשונה המציגה מגף מתויגת כ-9 כלומר Ankle boot.

הצגת מערך התמונות כולל התיוג שלהם

קובץ המסד מכיל 70000 תמונות בגודל של 28 פיקסלים על 28 פיקסלים לכל תמונה. כמובן שלא ניתן להציג על המסך את כלל התמונות אך ניתן להציג לפחות חלק מהם. בקוד הבא נציג את 49 התמונות הראשונות במאגר כולל התיוג של כל תמונה:

```
from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt


fashion_mnist = keras.datasets.fashion_mnist

(train_data, train_lbl), (test_data, test_lbl) = fashion_mnist.load_data()

lbls = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

train_data = train_data / 255.0

plt.figure()

for i in range(70):

    plt.subplot(7,10,i+1)

    plt.axis("off")

    plt.imshow(train_data[i], cmap='Greys')

    plt.title(lbls[train_lbl[i]], fontsize=10)

plt.show()
```

כפלט נקבל את המסך הבא:

התאמת מערך התמונות למבואות המכונה הלומדת

אחת המשימות החשובות בעבודה עם מכונות לומדות הוא להתאים את נתוני האימון והבדיקה לערכים המתאימים להיקלט במבואות של המכונה.

במכונה שלנו אנו צריכים לבצע 2 מטלות:

1. להמיר את הצבעים המרכיבים כל פיקסל בתמונה מטווח בין 0 ל- 255 לערכים בין 0 ל-1 כפי שלמדנו בפעיליות הקודמות.
2. המרת מטריצה דו-מימדית של כל תמונות (28X28) למערך חד ממדי הכולל 784 ערכים.

נעשה זאת על ידי חלוקת כל ערכי הפיקסלים ב- 255 על ידי ההוראות הבאות:

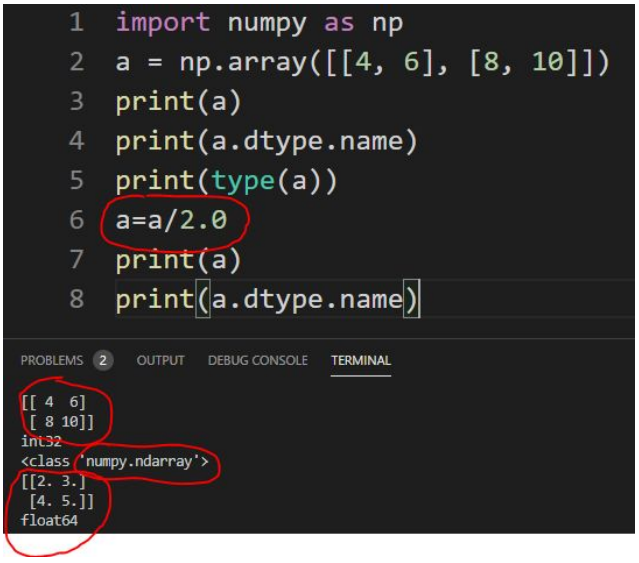
```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

שאלה:

כיצד 2 השורות הבאות מצליחות להמיר כל אחד מ- 784 הפיקסלים שבכל אחת מ- 70000 התמונות?

תשובה:

כפי שלמדנו על מערכים מסוג numpy.ndarray אנו יכולים לבצע פעולות ישירות על כל מבנה הנתונים.
לדוגמה, הקוד הבא מראה כיצד מחלקים ערכים בתוך מטריצה מסוג numpy.ndarray:

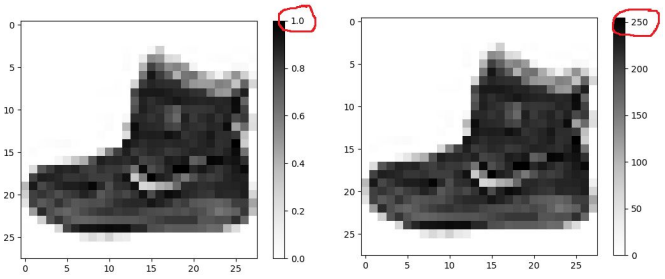


נבחן את תמונת המגף לאחר המרת הערכים מ 0 ל- 255 לערכים מ 0 ל-1 להלן קוד הבדיקה ופלט התוכנית:

```
from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt


fashion_mnist = keras.datasets.fashion_mnist

(train_data, train_lbl), (test_data, test_lbl) = fashion_mnist.load_data()


plt.figure()

plt.imshow(train_data[0], cmap='Greys')

plt.colorbar()

plt.grid(False)
```

```
plt.show()


train_data = train_data / 255.0


plt.figure()
plt.imshow(train_data[0], cmap='Greys')
plt.colorbar()
plt.grid(False)
plt.show()
```

נקבל את התמונות הבאות:



המרת מטריצה דו-מימדית של כל תמונות (28X28) למערך חד ממדי הכולל 784 ערכים נעשית תוך כדי בניית רשת נוירונים האופן הבא:

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

רשת הנוירונים לניתוח התמונות המורכבות מ- 28X28 פיקסלים ומתוייגות ל- 10 קטגוריות שונות בנוייה באופן הבא:

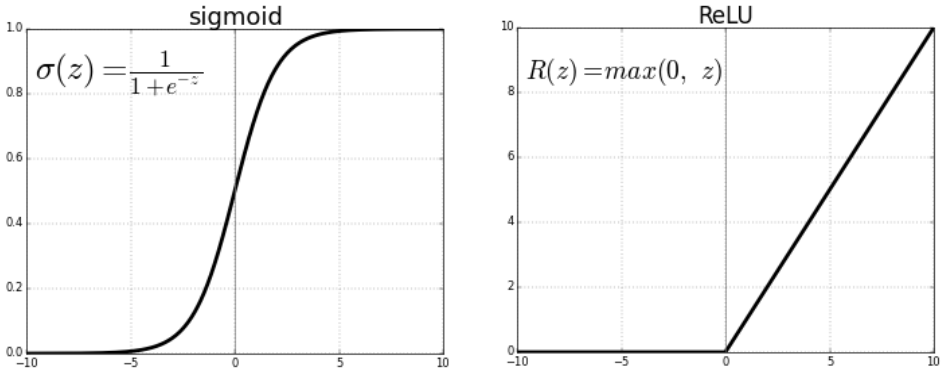
שכבת מבוא הכולל 784 כאשר השכבה הראשונה היא שכבה המשטחת את התמונה:

```
Flatten(input_shape=(28, 28))
```

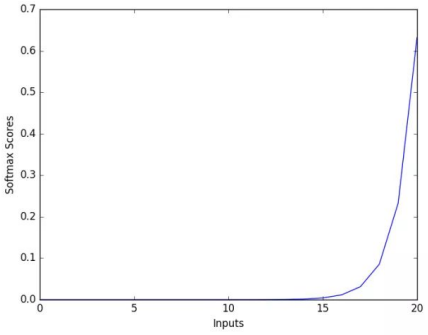
כלומר ממירה את הפיקסלים בתמונה למערך חד מימדי.

בנוסף הרשת מורכבת מ- 128 נוירונים בשכבה שנייה ועוד 10 נוירונים בשכבת המוצא.

פונקציית האקטיבציה של 128 הנוירונים בשכבה השנייה היא relu המתוארת באיור הבא תוך השוואה לפונקציה sigmoid שהכרנו כבר מהפעילויות הקודמות



פונקציית האקטיבציה של 10 הנוירונים בשכבה המוצא היא softmax המתוארת באיור הבא:



פונקציה זו שימושית בעיקר במערך נוירוני המוצא כאשר צריך להפריד בין יותר משני סיווגים. ניתן לממש את פונקציית האקטיבציה softmax על ידי הקוד הבא:

```
def softmax(x):
    np.exp(a) / np.sum(np.exp(a))
```

$$Softmax(z_j) = \frac{\sum e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

לאחר הגדרת רשת הנוירונים נעזר בפעולה compile כדי ליצור את הרשת. הפעולה מקבלת מספר פרמטרים הקובעים את האלגוריתמים ליישום הרשת:

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

לאחר בניית המודל נעזר בפעולה fit כדי לאמן את רשת הנוירונים. הפעולה fit מקבלת כקלט שלושה פרמטרים.

train_data - מערך התמונות לאימון

train_lbl- רשימת התגיות מתאימה לתמונות

epochs - מספר האיטרציות של לולאת האימון.

```
model.fit(train_data, train_lbl, epochs=10)
```

להלן ריכוז כל הקוד אותו למדנו עד עכשיו:

```
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

fashion_mnist = keras.datasets.fashion_mnist
(train_data, train_lbl), (test_data, test_lbl) = fashion_mnist.load_data()

train_data = train_data / 255.0
test_data = test_data / 255.0
```

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_data, train_lbl, epochs=10)
```

לאחר הרצת הקוד נקבל את הפלט הבא:

```
Train on 60000 samples
Epoch 1/10
60000/60000 [==============================] - 3s 48us/sample - loss: 0.5008 - accuracy: 0.8232
Epoch 2/10
60000/60000 [==============================] - 2s 40us/sample - loss: 0.3773 - accuracy: 0.8626
Epoch 3/10
60000/60000 [==============================] - 2s 40us/sample - loss: 0.3383 - accuracy: 0.8763
Epoch 4/10
60000/60000 [==============================] - 2s 40us/sample - loss: 0.3145 - accuracy: 0.8838
Epoch 5/10
60000/60000 [==============================] - 2s 40us/sample - loss: 0.2957 - accuracy: 0.8902
Epoch 6/10
60000/60000 [==============================] - 2s 41us/sample - loss: 0.2826 - accuracy: 0.8950
Epoch 7/10
60000/60000 [==============================] - 3s 53us/sample - loss: 0.2703 - accuracy: 0.8994
Epoch 8/10
60000/60000 [==============================] - 3s 51us/sample - loss: 0.2576 - accuracy: 0.9047
Epoch 9/10
60000/60000 [==============================] - 3s 49us/sample - loss: 0.2487 - accuracy: 0.9072
Epoch 10/10
60000/60000 [==============================] - 3s 51us/sample - loss: 0.2397 - accuracy: 0.9090
```

ניתן לראות שהמחשב עבר 10 פעמים על 60000 תמונות ובכל פעם ערכה של פונקציית המחיר קטן, כלומר עלתה רמת הדיוק של הרשת.

בדיקת הרשת לאחר שלב האימון

לאחר ביצוע האימן הרשת נבדוק את רמת הדיוק הרשת על ידי כך שנציג לה תמונות שלא היו חלק מהאימון, כלומר תמונות שהרשת לא הראתה לפני כן.

כדי לעשות את השלב הזה השארנו לנו 10000 תמונות מתוייגות ברשימה בשם test_images. נעזר בפעולה evaluate כדי לבדוק את יעילות הרשת. להלן ההוראה:

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=0)

print('\nTest accuracy:', test_acc)

print('\nTest loss:', test_loss)
```

נקבל סיכום של הבדיקה באופן הבא:

```
Test accuracy: 0.8909

Test loss: 0.3177053982079029
```

משמעות הדברים היא שהרשת הצליחה לזהות נכון כ-89% מהתמונות בסט המבחן. המדד לפנקצית המחיר הוא 0.317, ככל שערך זה קטן יותר משמעות דברים שהערכים במוצא הרשת קרובים יותר לערכים הנכונים בהשוואה לתוויות.

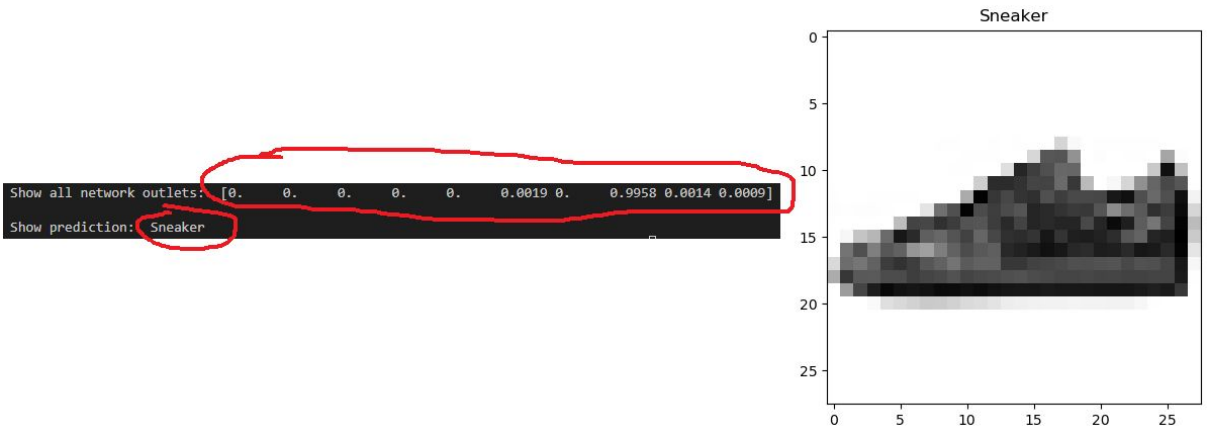
לאחר שקיבלנו את התוצאות הסטטיסטיות של ביצועי הרשת, נבחן עכשיו את המערכת על ידי כך שנבדוק את יכולות הסיווג שלה עבור תמונה בודדת ממאגר תמונות הבדיקה. נעשה זאת על ידי הקוד הבא:

```
from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt


lbls = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

fashion_mnist = keras.datasets.fashion_mnist

(train_data, train_lbl), (test_data, test_lbl) = fashion_mnist.load_data()
```

```
train_data = train_data / 255.0

test_data = test_data / 255.0


model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])


model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


model.fit(train_data, train_lbl, epochs=3)


predictions = model.predict(test_data)
np.set_printoptions(precision=4, suppress=True)
print("\n   Show all network outlets: ",predictions[22])
predic = np.argmax(predictions[22])
print("\n   Show prediction: ", lbls[predic])


plt.figure()
plt.imshow(test_data[22],cmap='Greys')
```

```
plt.title(lbls[predic])

plt.grid(False)

plt.show()
```

נקבל את הקוד הבא:



ניתן לראות תמונה בודדת מתוך התמונות. מדובר בתמונה של נעל ואכן המכונה סיווגה אותה כנעל ספורט.

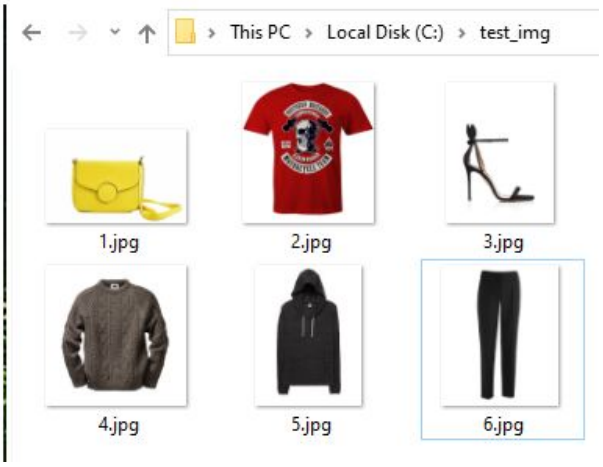
הרשת כוללת 10 מוצאים, אחד לכל קטגוריה, ניתן לראות שקיבלנו בכל אחד מהממוצאים את הפלט הבא:

0	T-shirt/top
0	Trouser
0	Pullover
0	Dress
0	Coat
0.0019	Sandal
0	Shirt
0.9958	Sneaker
0.0014	Bag
0.0009	Ankle boot

ניתן לראות שיש לנו זיהוי מושלם ברמת דיוק גבוהה מאוד ביחס לקטגוריות האחרות. יש לזכור כי זאת דוגמא בודדת והיא מוצגת לצורך ההדגמה.

בדיקת המכונה על ידי תמונות חדשות שלא מתוך מאגר התמונות

נבדוק את יכולות המכונה לסווג תמונות חדשות מהאינטרנט. נוריד תחילה מספר תמונות של פריטי לבוש לתוך תיקייה ייעודית (קחו בחשבון שאין חשיבות לגודל התמונה אך כן חשוב סוג הקובץ).



נכתוב תוכנית בדיקה העוברת בלולאה על כל התמונות שבתיקייה ומבצעת את הפעולות הבאות:

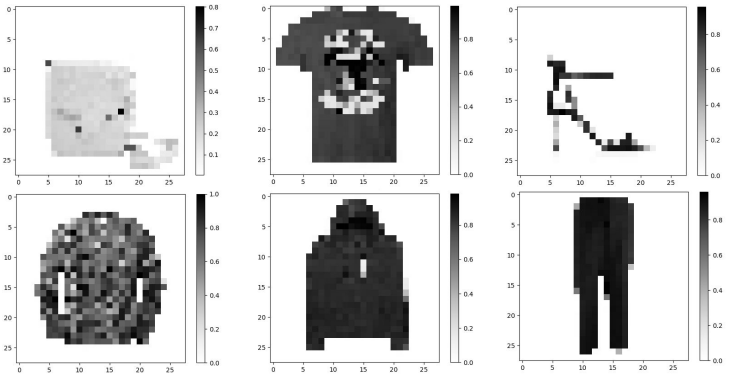
1. קוראת את התמונה המקורית וממירה אותה לגודל 28 על 28 פיקסלים.
2. ממירה את התמונה למערך מספרים ושומרת אותה בתוך מערך NumPy
3. ממירה את התמונה לגווני אפור ומנרמלת אותה לטווח שבין 0 ל- 1
4. מציגה את התמונה לבדיקה

להלן קוד התוכנית:

```
import numpy as np

import matplotlib.pyplot as plt

from keras.preprocessing import image

import glob

file_list = glob.glob("C:/test_img/*.jpg")
```

```
x=[]
for each in file_list:
    img = image.load_img(each, color_mode = "grayscale", target_size=(28, 28))
    img = image.img_to_array(img)
    img = img.reshape(28, 28)
    img = img.astype('float32')
    img = (255-img)/255.0
    x.append(img)
    plt.figure()
    plt.imshow(img , cmap='Greys')
    plt.grid(False)
    plt.colorbar()
    plt.show()
x=np.array(x)
```

פלט התוכנית יהיה כך



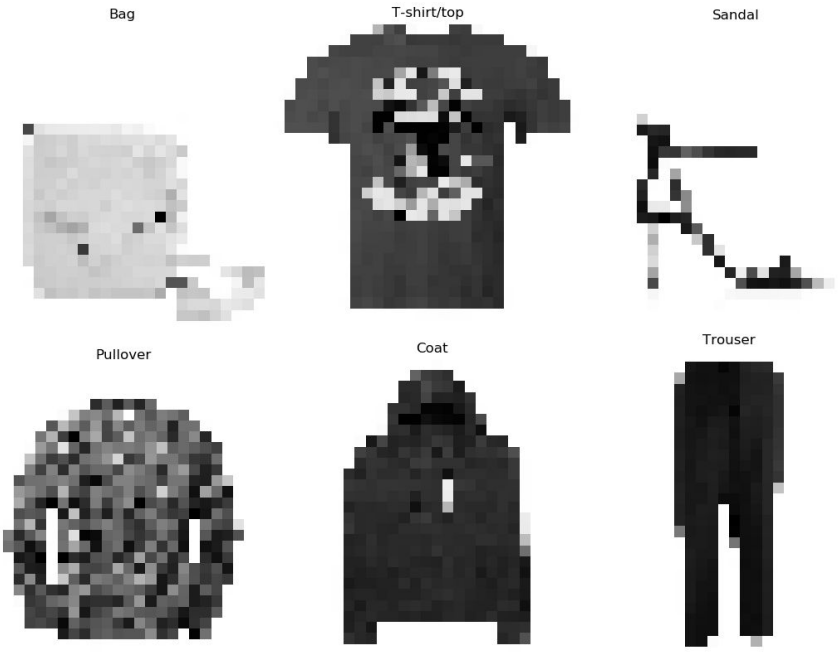
נשלב את הקוד שכתבנו זה עתה יחד עם המכונה הלומדת שכבר כתבנו בתחילת הפעילות ונקבל את הקוד הבא:

```
from tensorflow import keras
```

```
import numpy as np
import matplotlib.pyplot as plt
from keras.preprocessing import image
import glob

lbls = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

fashion_mnist = keras.datasets.fashion_mnist
(train_data, train_lbl), (test_data, test_lbl) = fashion_mnist.load_data()
train_data = train_data / 255.0
test_data = test_data / 255.0

model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(128, activation='relu'),
        keras.layers.Dense(10, activation='softmax')])
model.compile(optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])
model.fit(train_data, train_lbl, epochs=50)

file_list = glob.glob("C:/test_img/*.jpg")
x=[]
for each in file_list:
```

```
        img = image.load_img(each, color_mode = "grayscale", target_size=(28, 28))
        img = image.img_to_array(img)
        img = img.reshape(28, 28)
        img = img.astype('float32')
        img = (255-img)/255.0
        x.append(img)
x=np.array(x)
result = model.predict_classes(x)
for i in range(len(result)):
        print(lbls[result[i]], result[i])
print(result)
for i in range(len(result)):
        plt.figure()
        plt.imshow(x[i] , cmap='Greys')
        plt.grid(False)
        plt.axis('off')
        plt.colorbar()
        plt.title(lbls[result[i]])
        plt.show()
```

נקבל את הפלט הבא:



הצלחנו לייצר מכונה לומדת מבוססת על רשת נוירונים מלאכותיים ANN המסגולת לסווג תמונות של פרטי לבוש.

פעילות 18 - גיבוי ושחזור מערך המשקלים של רשת נוירונים

בפעילות זו נלמד כיצד ניתן להפריד בין שלב האימון של רשת הנוירונים, שלב שלוקח זמן ומשאבי מחשב גדולים, לבין שלב החיזוי המאפשר להפעיל את הרשת על סדרות נתונים ולקבל סיווג. יכולת זו תאפשר לנו בפעילויות הבאות לבצע אימן של רשות על מחשב PC חזק ולהפעיל את הרשת על מעבדים חלשים יותר כמו על לוח Raspberry PI.

כמו כן בפעילות זו נבחן את יכולת הניבוי של מכונה לומדת לחזות את הופעת מחלת הסוכרת בקרב מטופלים על סמך מספר מדדים פיזיולוגיים שיש לנו עליהם. כדי לאמן את המכונה נעשה שימוש בקובץ הנתונים Pima Diabetes Diabetes המכיל מידע רפואי על 768 נשים, הנתונים נאספו במקור במטרה לחזות הופעת את מחלת הסוכרת על סמך מספר מדדים שנאספו ממטופלות. מערך הנתונים נאסף על ידי המכון הלאומי לסוכרת ומחלות עיכול וכליות. המטרה היא לחזות האם חולה צפוי לסבול מסוכרת על סמך המדדים שנאספו עליו.

הקובץ כולל מדדים הבאים:

1. מספר ההריונות
2. ריכוז הגלוקוז בדם (שעתיים לאחר הבדיקה)
3. לחץ הדם הדיאסטולי
4. עובי העור
5. רמת האינסולין
6. מדד מסת גוף
7. מדד אילן היוחסין של הסוכרת
8. גיל המטופל
9. האם אובחנה כחולת סוכרת

ניתן להוריד את הקובץ דרך הקישור הבא:

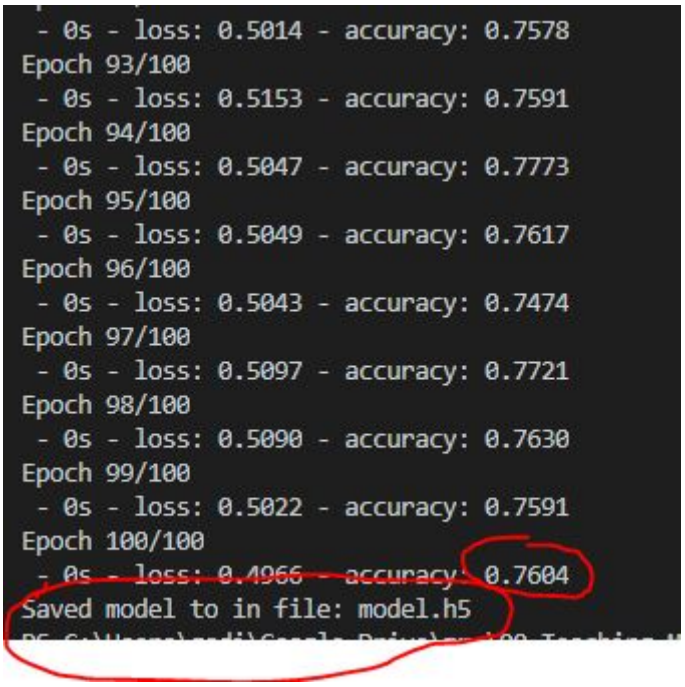
https://gist.github.com/ktisha/c21e73a1bd1700294ef790c56c8aec1f

נבנה את הרשת באה:

```
from numpy import loadtxt
```

```
from keras.models import Sequential
from keras.layers import Dense

dataset = loadtxt("pima-indians-diabetes.csv", delimiter=",")

X = dataset[:,0:8]
Y = dataset[:,8]

model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X, Y, epochs=100, batch_size=5, verbose=2)

model.save("model.h5")
print("Saved model to in file: model.h5")
```

נקבל את הפלט הבא:

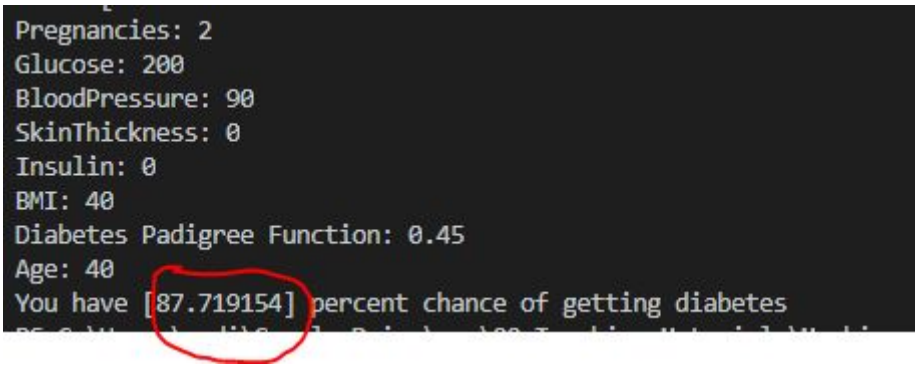


ניתן לראות שהפעולה שמרה את מבנה רשת הנוירונים כולל המשקלים שבה לאחר האימון ועכשיו היא זמינה לשימוש על ידי תוכנות אחרות לצורך חיזוי ללא צורך בביצוע של האימן מחדש. להלן הקוד:

```
from numpy import loadtxt

import numpy as np

from keras.models import load_model


model = load_model('model.h5')

dataset = loadtxt("pima-indians-diabetes.csv", delimiter=",")

X = dataset[:,0:8]

Y = dataset[:,8]

for i in range(20,23):

    print ("data=",X[i],"------",Y[i])
```

```
Pregnancies = float(input("Pregnancies: "))

Glucose = float(input("Glucose: "))

BloodPressure = float(input("BloodPressure: "))

SkinThickness = float(input("SkinThickness: "))

Insulin = float(input("Insulin: "))

BMI = float(input("BMI: "))

DiabetesPadigreeFunction = float(input("Diabetes Padigree Function: "))

Age = float(input("Age: "))

t = np.array([  Pregnancies,Glucose,

                BloodPressure,SkinThickness,

                Insulin,BMI,

                DiabetesPadigreeFunction,Age

            ],ndmin=2)

prediction = model.predict(t)

print ("You have", str(prediction[0]*100) ,"percent chance of getting diabetes")
```

נקבל את הפלט הבא:



לסיכום

ניתן לראות שאפשר להפריד בין קטע הקוד המאמן את רשת הנוירונים, שלבשלוקח זמן ומשאבי מחשב גדולים. לבין הקוד המאפשר להפעיל את הרשת על סדרות נתונים חדשות כדי לקבל חיזוי. יכולת זו תאפשר

לנו בפעילויות הבאות לבצע אימן של רשות על מחשב PC חזק ולהפעיל את הרשת על מעבדים חלשים יותר כמו על לוח Raspberry PI.

מקורות וקישורים

יסודות:

פעילות 1 - תכנות גרפים תוך שימוש ב- Matplotlib

https://matplotlib.org/tutorials/index.html

https://www.science-emergence.com/Articles/How-to-create-a-scatter-plot-with-several-colors-in-matplotlib-/

פעילות 2 - מערכים תחת NumPy Arrays

http://cs231n.github.io/python-numpy-tutorial/#python-containers

https://www.pythoninformer.com/python-libraries/numpy/index-and-slice/

https://jakevdp.github.io/PythonDataScienceHandbook/02.02-the-basics-of-numpy-arrays.html

https://www.programiz.com/python-programming/matrix

https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/

פעילות 3 - עבודה עם מטריצות תוך שימוש ב- NumPy

https://www.programiz.com/python-programming/matrix

פעילות 4 - פעולות (פונקציות) ומחלקות ב- Python

https://docs.python.org/3.5/tutorial/classes.html

https://docs.python.org/3.5/tutorial/controlflow.html#defining-functions

http://cs231n.github.io/python-numpy-tutorial/#python-functions

פעילות 5 - ייצוג מידע ויזואלי במחשב

https://matplotlib.org/

https://stackoverflow.com/questions/46615554/how-to-display-multiple-images-in-one-figure-correctly/46616645

https://matplotlib.org/users/pyplot_tutorial.html

אלגוריתמי למידת מכונה בסיסיים

פעילות 6 - אלגוריתם KNN ככלי לפיתוח מכונה לומדת

https://medium.com/datadriveninvestor/knn-algorithm-and-implementation-from-scratch-b9f9b739c28f

https://www.youtube.com/watch?v=aMtckmWAzDg&t=393s

https://github.com/muzzart/knn-cifar10/blob/master/KNN%20-%20CIFAR10.ipynb

Run file for k- Nearest neighbor algorithm

https://github.com/dhingratul/k-Nearest-Neighbors/blob/master/run_knn.py

פעילות 7 - רגרסיה לינארית בסביבת Python

https://www.geeksforgeeks.org/linear-regression-python-implementation/

https://www.youtube.com/watch?v=szXbuO3bVRk

פעילות 8 - יסודות מתמטיים ל- Gradient Descent

https://www.youtube.com/watch?v=jc2IthslyzM

https://www.youtube.com/watch?v=sDv4f4s2SB8

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

https://gist.github.com/sagarmainkar/41d135a04d7d3bc4098f0664fe20cf3c

https://towardsdatascience.com/gradient-descent-in-python-a0d07285742f

https://machinelearningmastery.com/gradient-descent-for-machine-learning/

פעילות 9 - יישום רגרסיה לינארית על ידי Gradient Descent

https://www.youtube.com/watch?v=L-Lsfu4ab74

https://www.youtube.com/watch?v=jc2IthslyzM

תכנות מכונה לומדת צעד אחר צעד מהיסוד:

פעילות 10 - יישום פרספטרון בודד

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

פעילות 11 - מימוש שער XOR על ידי מספר נוירונים

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/@thomascountz/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

פעילות 12 - רשת נוירונים לסיווג תמונות

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

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)

יישומי מכונה לומדת בשפת Python

פעילות 13 - פיתוח מכונה לומדת לזיהוי ספרות בכתב יד עם ספרייתscikit-learn

https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py

https://matplotlib.org/

https://scikit-learn.org/stable/install.html

https://playground.tensorflow.org/

https://www.openml.org/d/554

https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d

https://machinelearningmastery.com/make-predictions-scikit-learn/

https://scikit-learn.org/stable/index.html

https://gogul09.github.io/software/image-classification-python

פעילות 14 - מימוש שער XOR על ידי מכונה לומדת עם ספריית Keras

https://github.com/penseeartificielle/perceptron-xor-examples/blob/master/keras-xor.py

https://gist.github.com/stewartpark/187895beb89f0a1b3a54

https://victorzhou.com/blog/keras-neural-network-tutorial/

פעילות 15 - רשת נוירונים מבוססת Keras לסיווג תוכן בתמונות

http://www.cs.toronto.edu/~kriz/cifar.html

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

https://keras.io/datasets/

פעילות 16 - רשת נוירונים מבוססת Keras לסיווג הודעות טקסט

https://builtin.com/data-science/how-build-neural-network-keras

http://tech.couponall.in/tag/imdb/

https://stackoverflow.com/questions/51363709/how-to-predict-sentiment-analysis-using-keras-imdb-dataset

https://kite.com/python/docs/nltk.punkt

פעילות 17 - סיווג פריטי לבוש בתמונות תוך שימוש keras

https://keras.io/models/model/

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/2.1-a-first-look-at-a-neural-network.ipynb

https://www.tensorflow.org/tutorials/keras/classification

https://www.manning.com/books/deep-learning-with-python?a_aid=keras&a_bid=76564dff

https://livebook.manning.com/book/deep-learning-with-python/chapter-3/8

פעילות 18 - גיבוי ושחזור מערך המשקלים של רשת נוירונים (איך להעתיק רשת נוירונים)

https://github.com/KriAga/Pima-Indians-Diabetes-Dataset-Classification

https://machinelearningmastery.com/save-load-keras-deep-learning-models/

https://gist.github.com/vihar/682dccbe9f92b68f882b42c754df07f3

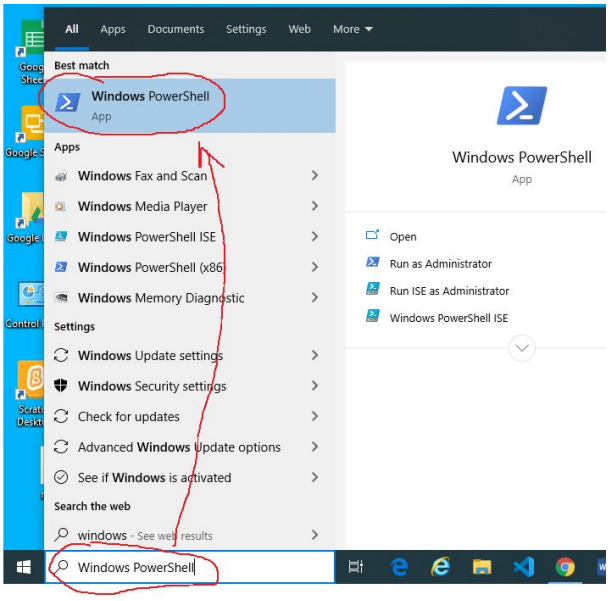
מור-טק מרכז המורים הארצי למקצועות הטכנולוגיים
הפקולטה לחינוך למדע וטכנולוגיה, קריית הטכניון, חיפה 32000 טל: 04-8293146
E-mail: Moretech@ed.technion.ac.il
http://moretech.technion.ac.il

התקנת ספריות Python

ניתן להתקין את jupyter notebook מהאתר של anaconda או להשתמש ב- google colab ללא צורך בהתקנות כלל.

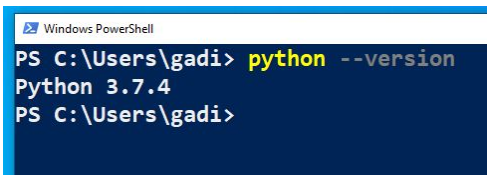
התקנת הספרייה Matplotlib

נפתח את חלון הפקודות Windows PowerShell על ידי ההוראה הבא:



נבדוק תחילה שהמפרש של Python מותקן במחשב, נעשה זאת על ידי ההוראה:

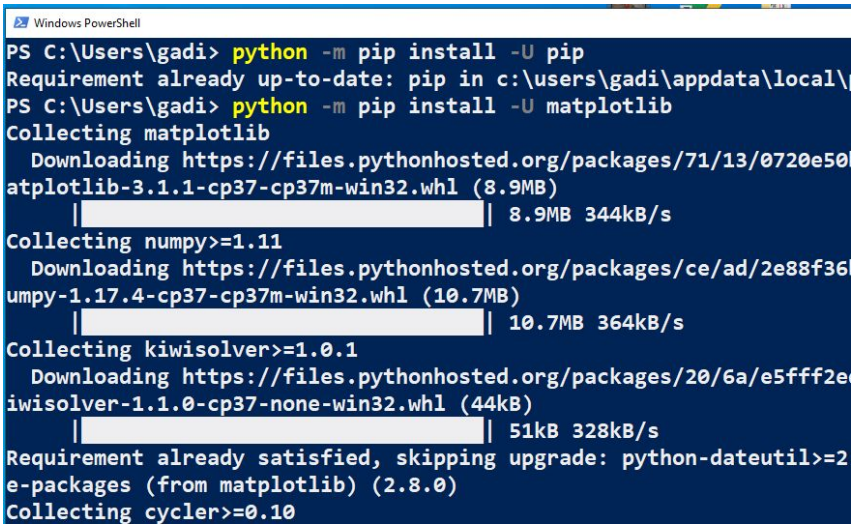
```
> python --version
```



התקנת הספרייה תתבצע על ידי ההוראה הבאה:

```
> python -m pip install -U pip

> python -m pip install -U matplotlib

> python -m pip install -U Pillow

> python -m pip install -U numpy
```

נקבל כפלט את המסך הבא:

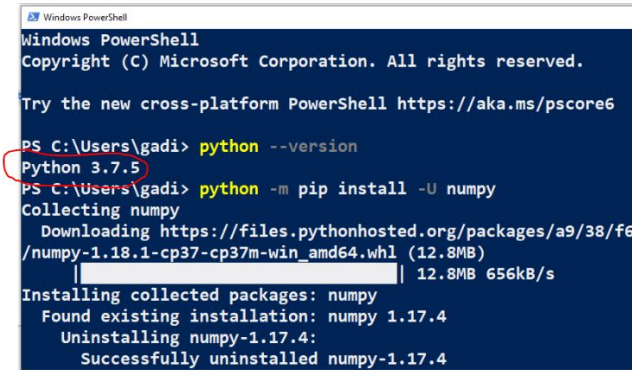


התקנת הספרייה NumPy

התקנת הספרייה NumPy תתבצע על ידי ההוראה הבאה:

```
> python -m pip install -U numpy
```

נקבל את הפלט הבא:



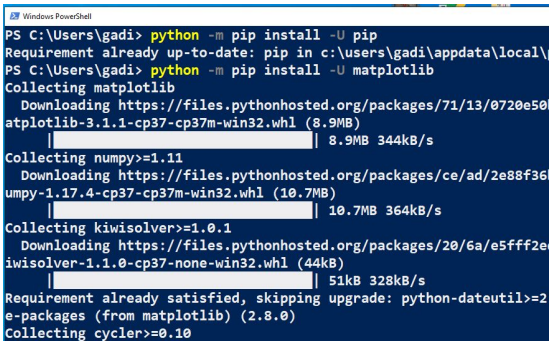
התקנת הספרייה Pillow - PIL

Pillow היא היא ספרייה ב- Python לטיפול בתמונות.

התקנת הספרייה תתבצע על ידי ההוראה הבאה:

```
> python -m pip install -U pip
> python -m pip install -U matplotlib
> python -m pip install -U Pillow
> python -m pip install -U numpy
```

נקבל כפלט את המסך הבא:



התקנת הספריות scikit-learn

התקנת הספרייה תתבצע על ידי ההוראה הבאה:

```
> python --version
> python -m pip install -U pip
> python -m pip install -U matplotlib
> python -m pip install -U pandas
> python -m pip install -U scikit-learn
```

התקנת הספריות tensorflow ו- keras

התקנת הספרייה תתבצע על ידי ההוראה הבאה:

```
> python --version
> python -m pip install -U pip
> python -m pip install -U tensorflow
> python -m pip install -U keras
```

כדי לבדוק את גרסה tensorflow ו- keras שהתקנתם ניתן לכתוב את ההוראות הבאות:

```
> python -c 'import tensorflow as tf; print(tf.__version__)'
> python -c 'import keras; print(keras.__version__)'
```

נקבל את הפלט הבא:

```
PS C:\Users\gadi> python -c 'import tensorflow as tf; print(tf.__version__)'
2020-02-29 16:39:10.718906: W tensorflow/stream_executor/platform/default/dso
ry 'cudart64_101.dll'; dlerror: cudart64_101.dll not found
2020-02-29 16:39:10.742956: I tensorflow/stream_executor/cuda/cudart_stub.cc:
ot have a GPU set up on your machine.
2.1.0
PS C:\Users\gadi> python -c 'import keras; print(keras.__version__)'
Using TensorFlow backend.
2020-02-29 16:42:05.661348: W tensorflow/stream_executor/platform/default/dso
ry 'cudart64_101.dll'; dlerror: cudart64_101.dll not found
2020-02-29 16:42:05.666217: I tensorflow/stream_executor/cuda/cudart_stub.cc:
ot have a GPU set up on your machine.
2.3.1
```