פעילות 8 - יישום רגרסיה לינארית על ידי Gradient Descent

מקורות:

https://www.youtube.com/watch?v=L-Lsfu4ab74

https://www.youtube.com/watch?v=jc2IthslyzM

בפעילות זו נתבסס על העקרונות המתמטיים של Gradient descent כדי לממש רגרסיה לינארית. פעילות זו כמו גם הפעילות הקודמת בנושא רגרסיה ליניארית 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 גם בפעילות הקודמת בנושא רגרסיה לינארית בסיסית נחפש את ערכי m ו- b של משוואת קו ישר:

$$y = mx + b$$

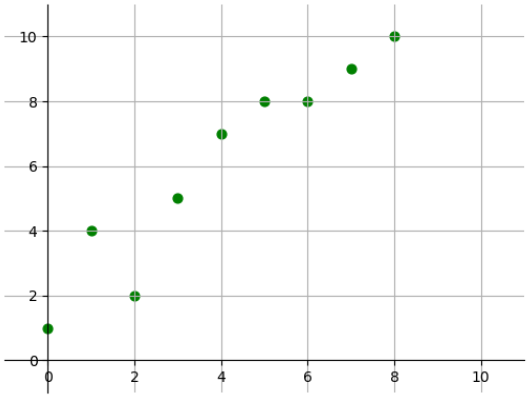
כלומר ההנחה הבסיסית כאן היא שאנו מבקשים למצוא פונקציה לינארית שמנבאת את הערך של y בצורה מדויקת ככל האפשר כפונקציה של המשתנה הבלתי תלוי x. לשם כן נגדיר 2 רשימות נתונים האחת מייצגת את הערכים של x והשנייה את הערכים של y. בפעילות זו ננתח את 2 הרשימות כדי ליצור מהם את משוואת הקו הישר המייצגת באופן אופטימלי 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

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])
```

```
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:

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

נממש פעולה בשם GradientDescent 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 מספר שיקבע מה יהיה קצב השינוי תוך כדי ביצוע חישובי הקירוב לערכים שאנו מחפשים. בפעולה זו נגדיר כברירת מחדל את הערך 0.01
- משתנה בשם epochs המכיל מספר שיקבע כמה קירובים נעשה תוך כדי חיפוש הערכים שלנו. בפעולה זו נגדיר כברירת מחדל את הערך 1000

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

להלן מימוש הפעולה:

```
def GradientDescent(x,y,learning_rate=0.01, epochs=1000):
    m=0
    b=0
    for _ in range(epochs):
        for i in range(len(x)):
            xi = x[i]
```

```
            yi = y[i]
            guess = m * xi + b
            error = yi - guess
            m = m + (error * xi) * learning_rate
            b = b + error * 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:

```
guess = m * xi + b
error = yi - guess
m = m + (error * xi) * learning_rate
b = b + error * learning_rate
```

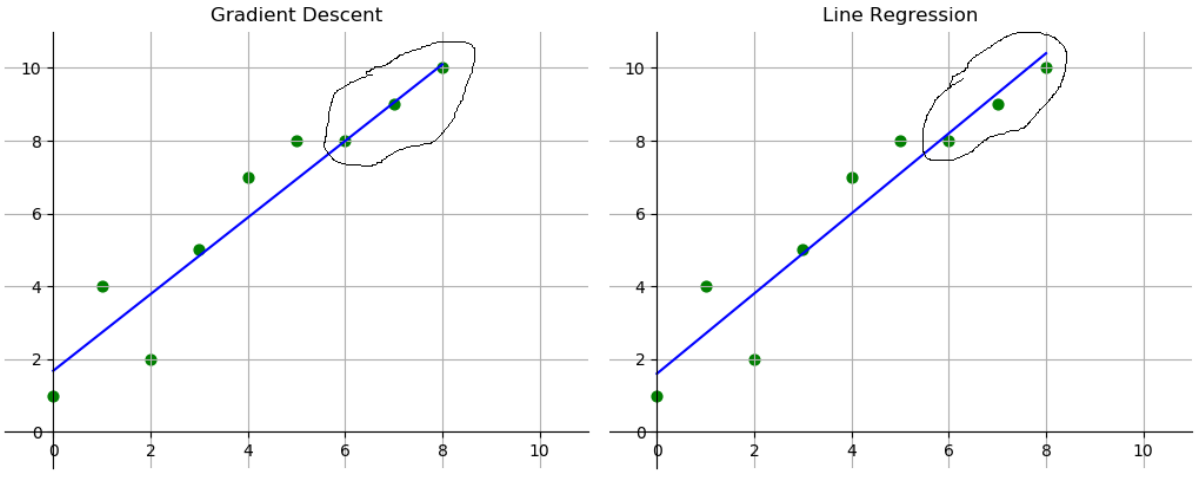
תחשב במשתנה guess את פונקציית המטרה שלנו כלומר משוואת הקו הישר שאנו מחפשים, בהמשך הפעולה חשב במשתנה error את השגיאה הנוכחית, הרגעית, ואז תכוון את ערכי m ו- b לערכים חדשים על פי קצב השינוי שנקבע במשתנה learning_rate.

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
    b=0
    for _ in range(epochs):
        for i in range(len(x)):
            xi = x[i]
            yi = y[i]
            guess = m * xi + b
            error = yi - guess
            m = m + (error * xi) * learning_rate
            b = b + error * learning_rate
    return m,b
```

```
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])

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. בפעם הראשונה שבדקנו את הפעולה Gradient Descent בצד שמאל של האיור ומנגד את קו המגמה שקיבלנו בפעילות הקודמת 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.

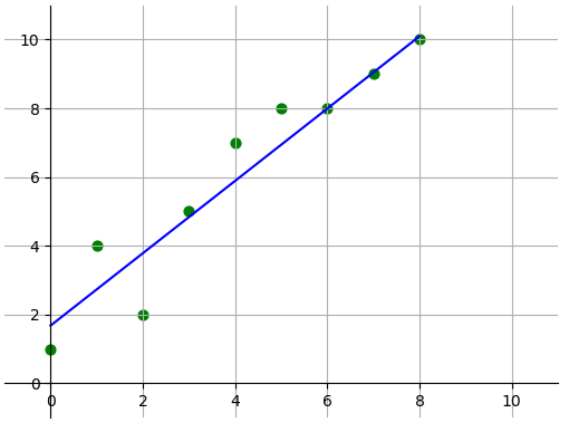
נבחן כעת את התנהגות הפעולה GradientDescent עבור ערכים שונים של learning_rate שייצגים את קצב השינוי תוך כדי ביצוע חישובי הקירוב ושל משתנה epochs הקובע את מספר הקירובים שנעשה.

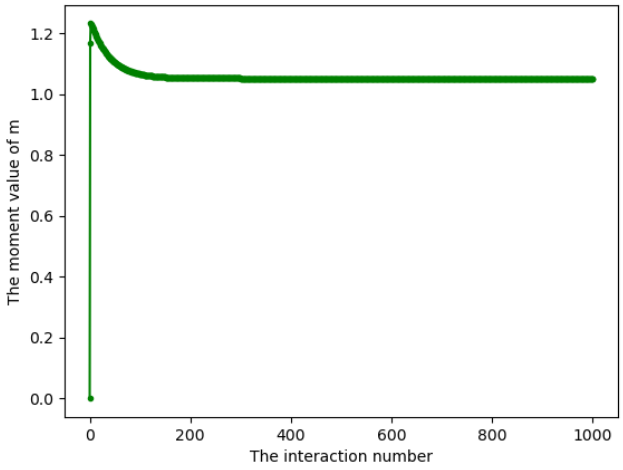
להלן קוד תוכנית שיעזור לנו בביצוע המשימה:

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


def GradientDescent(x,y,learning_rate=0.001, epochs=200):
    m=0
    b=0
    all_m = np.array([0])
    for _ in range(epochs):
        for i in range(len(x)):
            xi = x[i]
            yi = y[i]
            guess = m * xi + b
            error = yi - guess
            m = m + (error * xi) * learning_rate
            b = b + error * 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['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])


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:

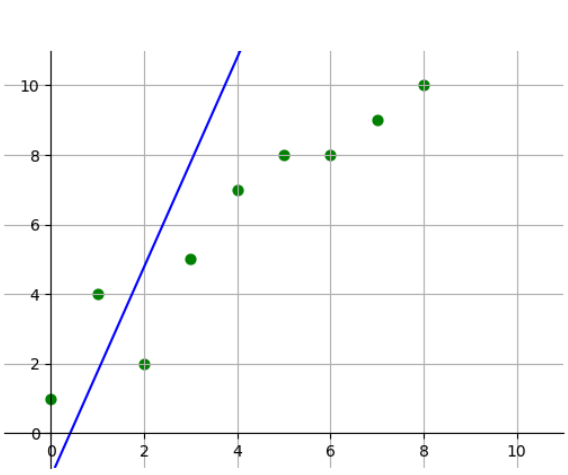


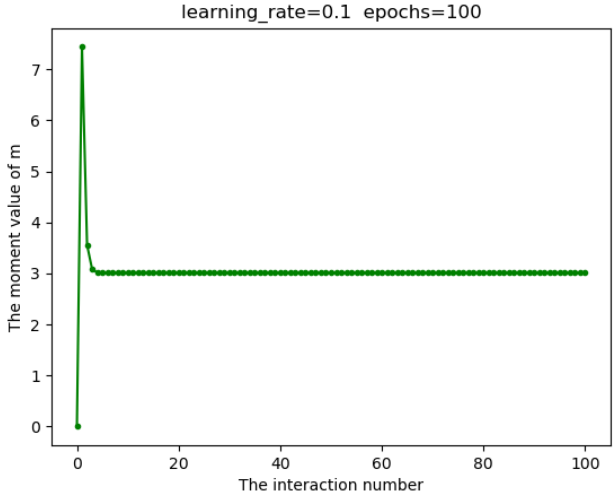


כבר כעת ניתן ללמוד שמספר האינטראקציות שקבענו ל- 1000 גדול מדי כי לאחר כ- 200 פעולות ערכו של m נשאר יציב וקובע.

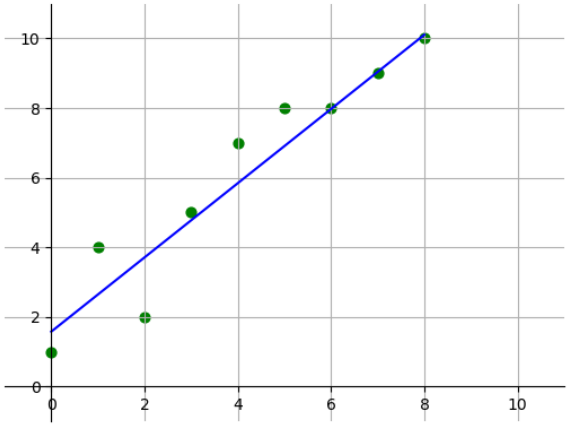
נבחן את השפעתו של המשנה learning_rate על משתנה m תוך כדי שינוי ערכו לערכים הבאים: 0.1 0.01 ו- 0.001

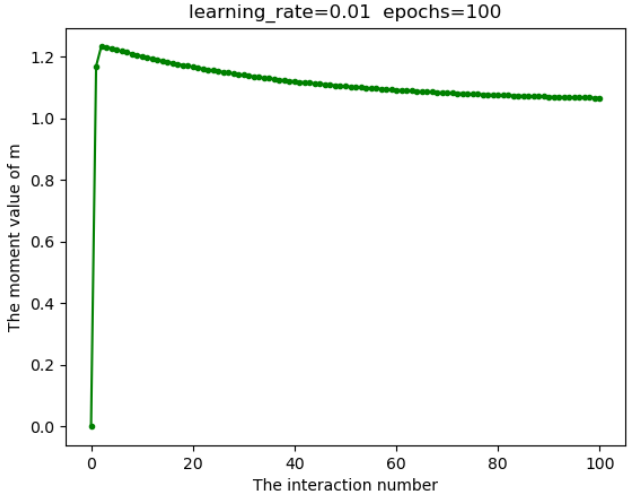
הרצה ראשונה learning_rate שווה ל-0.1:



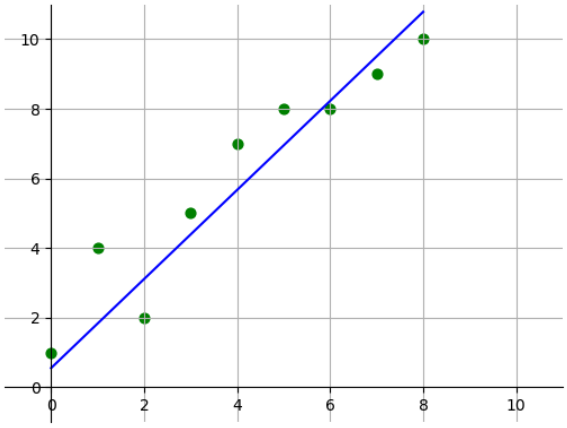


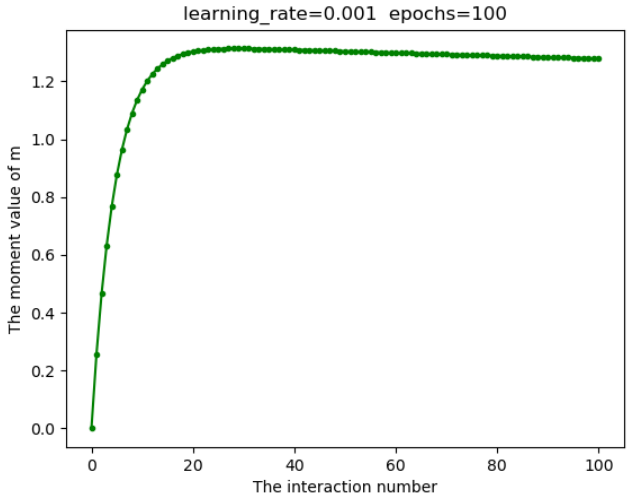
הרצה שניה learning_rate שווה ל-0.01:





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.

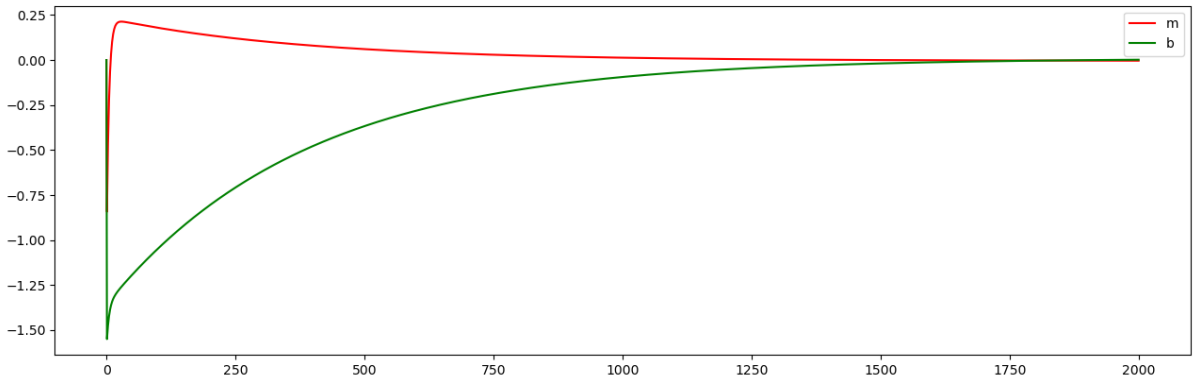
לסיכום נממש זה לצד זה קוד למציאת קו מגמה תוך שימוש בשיטת Line Regression ובשיטת Gradient Descent כך שניתן יהיה להשוות בין 2 השיטות. התוכנה שלהלן מפיקה גרף המתאר את השגיאה בין הערכים שמחשבת כל שיטה.

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):
        for i in range(len(x)):
            xi = x[i]
            yi = y[i]
            guess = m * xi + b
            error = yi - guess
            m = m + (error * xi) * learning_rate
            b = b + error * learning_rate
        err_m = np.append(err_m,m-lin_m)
        err_b = np.append(err_b,b-lin_b)
```

```
    return m,b,err_m,err_b

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])

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:



תנאי השימוש

תנאי השימוש במסמך זה הם לפי הסטנדרט הבא:

You are free:

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

Under the following conditions:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
NonCommercial — You may not use the material for commercial purposes.
ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.