

פעילות 7 - יסודות מתמטיים ל- Gradient Descent

מקורות:

<https://www.youtube.com/watch?v=jc2lthslyzM>

<https://www.youtube.com/watch?v=sDv4f4s2SB8>

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

<https://gist.github.com/sagarmainkar/41d135a04d7d3bc4098f0664fe20cf3c>

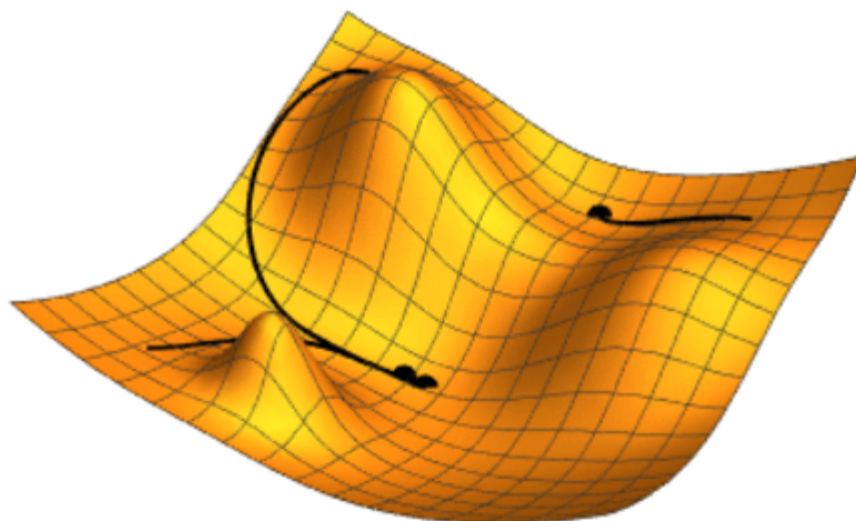
<https://towardsdatascience.com/gradient-descent-in-python-a0d07285742f>

<https://machinelearningmastery.com/gradient-descent-for-machine-learning/>

בפעילות זו נלמד את היסודות המתמטיים של Gradient descent ויישומיה בשפת python. נושא זה יחד עם גרסיה 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.



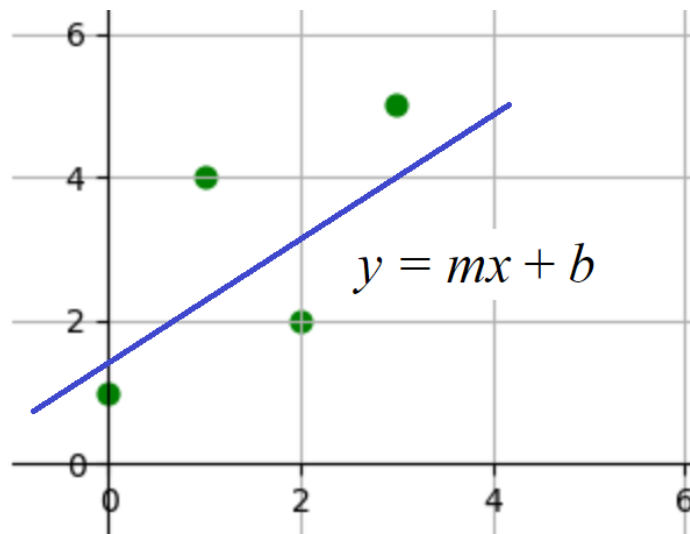
מקור התמונה: https://upload.wikimedia.org/wikipedia/commons/a/a3/Gradient_descent.gif (לשימוש חוזר עם אפשרות לשינויים)

נתרגם את אנלוגיות הירידה מההר למושגים שכבר הכרנו בתחום למידת המכונה:

- הר הכולל שיפועים - פונקציה הכוללת שני מבוואות (x,y) במקרה שלנו כאשר $z = f(x,y)$.
- מציאת הכיוון עליה או ירידה - שיפוע הפונקציה.
- הצעד שעושה האיש בכל פעם - משתנה learningRate
- חזרה במספר רב של פעמים - משתנה epochs

נתחיל לפתח את הרעיון המתמטי שבבסיס Gradient descent על ידי מבט חוזר על עקרונות שלמדנו ברגרסיה לינארית הפעילות הקודמת.

נתונים לנו מספר נקודות על גבי מערכת צירים וקו לינארי שמשוואתו היא $y = mx + b$



באופן כללי במערכות של מכונה לומדת נרצה לנבא את ערך של y לפי ערך ידוע של x . (אך קודם לכך נרצה לכוון (לאמן) את המכונה להגיע לרמת דיוק גבוהה). לשם כך נגדיר נקודה x כלשהי 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 .

$$error = guess_i - y_i$$

$$guess = mx_i + b$$

נגדיר פונקציית מחיר באופן הבא:

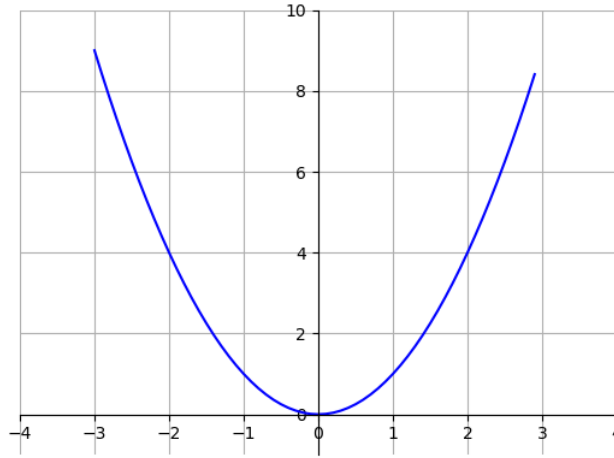
$$COST = \sum_{i=1}^n (guess_i - y_i)^2$$

בהתבסס על האנלוגיה שלנו לגבי הירידה מההר הערך של COST הוא הגובה הנוכחי של האדם על ההר מכאן שנסכים 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.

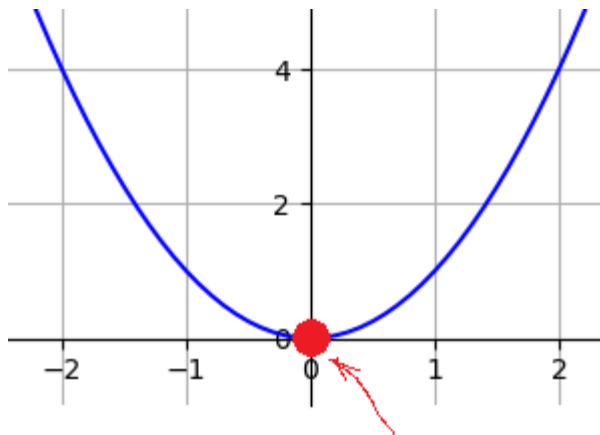
נבחן את המשוואה שכתבנו ונקבל שהיא לא רחוקה מהפונקציה הפולינומית הבא:

$$y = x^2$$

גרף פונקציה כזו נראה באופן הבא:



מכאן שעלינו לחפש את הערך של x המספק לנו את הערך הנמוך ביותר של y . כלומר לחפש את נקודת המינימום בגרף.



נגזרת של פונקציה פולינומית

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

נתונה לנו 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['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
plt.xlim([-4, 4])
plt.ylim([-1, 10])
plt.grid()

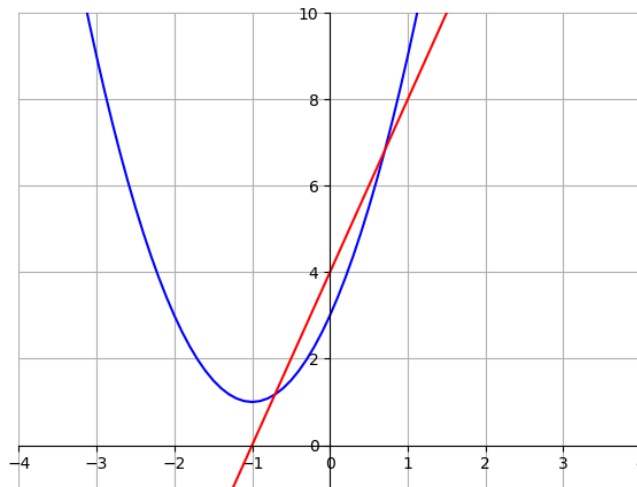
x = np.arange(-4.0, 3.0, 0.1)
y = 2*x**2 + 4*x + 3
y2 = 4*x+4
plt.plot(x, y, color = "b")
plt.plot(x, y2, color = "r")
plt.show()
```

שימו לב לאופן כתיבת משוואת פונקציה בשפת python:

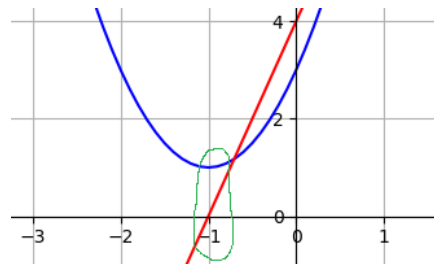
$$y = 2*x**2 + 4*x + 3$$

$$y2 = 4*x+4$$

נקבל את הפלט הבא:



מהגרף ניתן לראות את הפונקציה y בצבע כחול והנגזרת שלה y' 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.



<https://www.derivative-calculator.net>

לתרגול הנושא מומלץ לבקר באתר הבא:

קירת Cost Function תוך שימוש בנגזרות

עד כה פגשנו כבר את 2 המשוואות הבאות:

$$error = guess - y$$

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

נסמן,

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

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

$$J_{(m,b)} = err^2$$

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

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

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

$$m = m + \Delta m$$

$$b = b + \Delta b$$

מכאן שאנו צריכים למצוא את השיפוע בכל אחד מ-2 הצירים (m ו- b). 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 קובע ובפעם השנייה נחשב את הנגזרת של J לפי b כך m קובע. אך תחילה יש להבין כיצד גוזרים את הפונקציה הבאה:

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

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 מורכבות.

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 בנגזרת 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 $J_{(m,b)} = (mx_i + b - y_i)^2$ ונממש 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 J לפי m

ראינו כבר ש:

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

מכאן ניתן לתאר את הנגזרת של J לפי m כך:

$$J_{(m)} = (err)^2$$

$$J_{(m)}' = 2(err)^1 \cdot err'$$

נגזרת של err לפי m תראה כך:

$$err = mx + b - y$$

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

מכאן ש:

$$J_{(m)}' = 2 \cdot err \cdot x$$

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

$$J_{(m)}^1 = (2 \cdot err \cdot x) \cdot Learning Rate$$

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:

$$m = m + \Delta m$$

$$J_{(m)}^1 = err \cdot x \cdot Learning Rate$$

$$m = m + err \cdot x \cdot Learning Rate$$

נגזרת של J לפי b

ניתן לתאר את הנגזרת של J לפי b כך:

$$J_{(b)} = (err)^2$$

$$J_{(b)}^1 = 2(err)^1 \cdot err^1$$

נגזרת של err לפי b תראה כך:

$$err = mx + b - y$$

$$err_{(b)}^1 = 0 + 1 \cdot b^0 - 0 = 1$$

מכאן ש:

$$J_{(b)}^1 = 2 \cdot err \cdot 1$$

$$J_{(b)}^1 = (2 \cdot err \cdot 1) \cdot Learning Rate$$

$$b = b + \Delta b$$

$$J_{(b)}^1 = err \cdot x \cdot Learning Rate$$

$$b = b + (2 \cdot err \cdot 1) \cdot Learning Rate$$

$$b = b + err \cdot Learning Rate$$

בפעילות הבא נשתמש ב-2 הנוסחאות שנלמדו כדי לממש פעולה בשפת python למציאת הערכים של m ו-b תוך שימוש בעקרונות Gradient Descent. להלן הפעולה שנלמד בפעילות הבא

```

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

```

$$m = m + err \cdot x \cdot Learning\ Rate$$

$$b = b + err \cdot Learning\ Rate$$

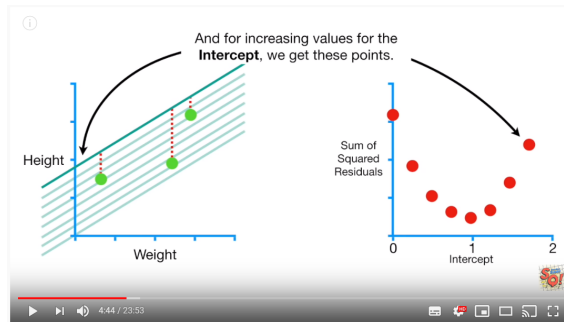
לסיכום

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.

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.