אימון הנוירון - מימוש מהיסוד

כעת, כאשר ברשותנו כל הדרוש לאימון הנוירון בעזרת אלגוריתם מורד הגרדיאנט, ניגש למשימה. ראשית נגדיר פונקציה המחשבת את ההסתברות החזויה על פי המודל לכך שנקודה נתונה היא לבנה.

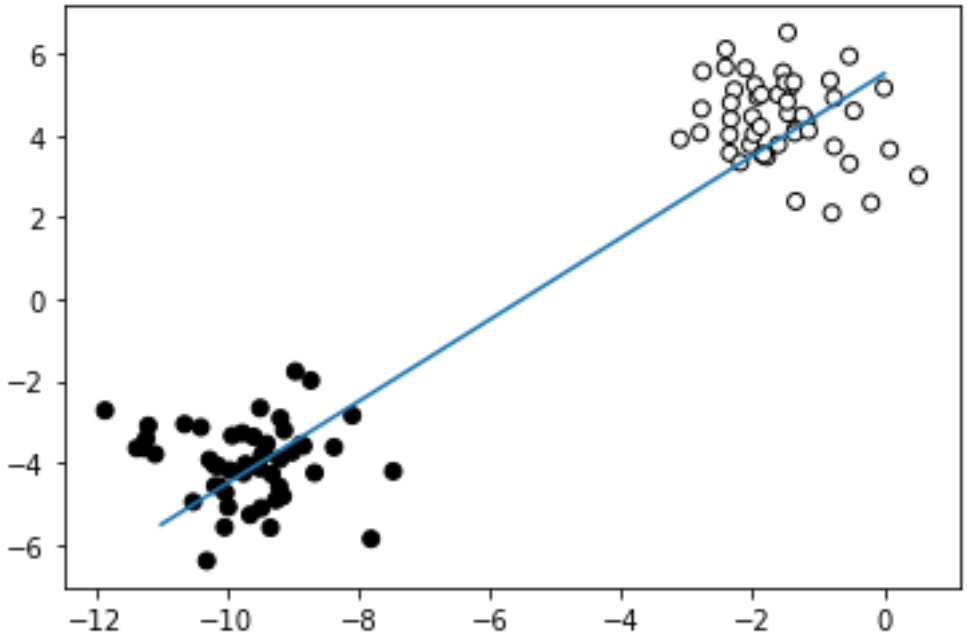
```
z =     lambda w0, w1, b, x0, x1: w0*x0+w1*x1+b
y =     lambda z: 1/(1+torch.exp(-z))
model = lambda w0, w1, b, x0, x1: y(z(w0, w1, b, x0, x1))
```

אחרי כן, נאתחל את האלגוריתם בבחירה גרועה במיוחד של פרמטרים, בכדי להמחיש את פעולת הלמידה.

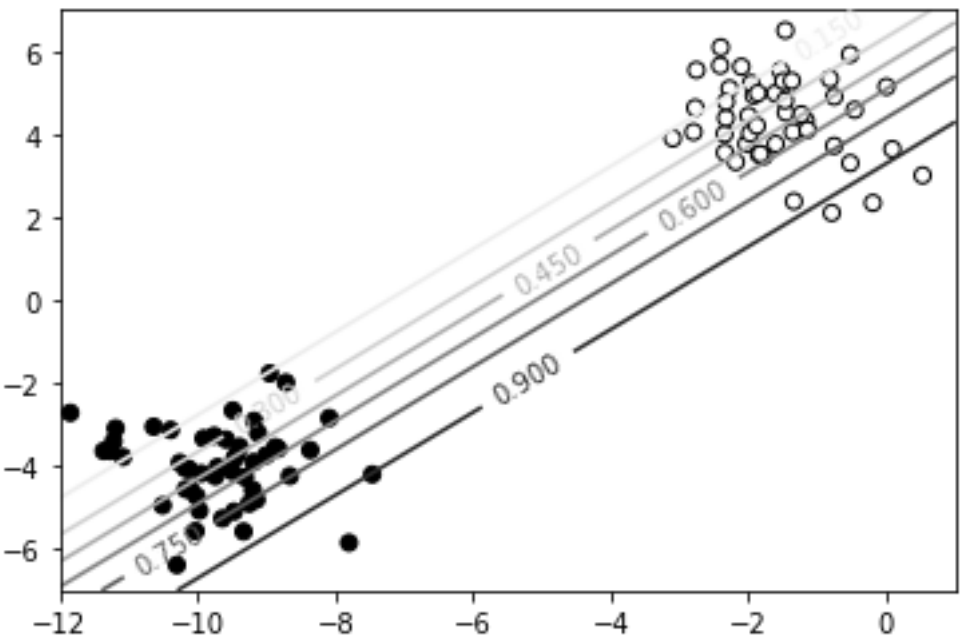
```
def draw_05_line():
  line=lambda x:-w0/w1*x-b/w1
  x0=torch.tensor([-11,0])
  x1=line(x0)

  plt.plot(x0,x1);
  plt.scatter(X[:, 0], X[:, 1],
              c=Y, cmap="Greys", edgecolor="black");
w0, w1, b = 1, -1, 5.5
draw_05_line()
```

פלט:



עבור ערכי הפרמטרים הנבחרים, הקו המפריד שמתחתיו המודל יסווג את הנקודות השחורות בהסתברות גבוהה מ0.5, מצוייר בכחול, וניכר שעבור בחירה זו כחצי מהנקודות יסווגו כנכון. למעשה המודל לא יביע בטחון רב בסיווג אף נקודה, כפי שניתן לראות באיור הבא.



כעת, נחשב את הגרדיאנט של H עבור כל אחת מנקודות הדגימה, ולבסוף נמצא את הערכים המתקבלים בכדי לקבל את $\nabla C(w_0, w_1, b)$.

```
dHdy =  lambda y,yt: -1(yt-y)/(y-y**2)
dydz =  lambda y: y*(1-y)
dzdw0 = lambda x0: x0
dzdw1 = lambda x1: x1
dzdb =  1
def calc_dC():
  dH=torch.zeros(len(Y),3)
  for idx in range(len(Y)):
    data=(X[idx,0],X[idx,1],Y[idx])
    y_model=y(z(w0,w1,b,data[0],data[1]))

    A=dHdy(y_model,data[2])
    B=dydz(y_model)
    dH[idx,0]=A*B*dzdw0(data[0])
    dH[idx,1]=A*B*dzdw1(data[1])
    dH[idx,2]=A*B*dzdb
  return dH.mean(0)
calc_dC()
```

פלט:

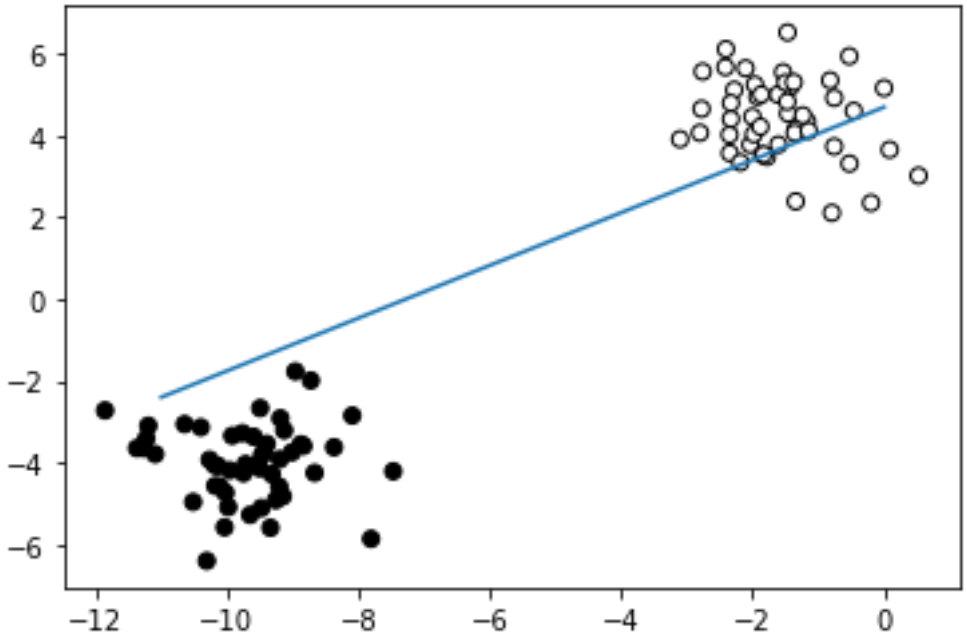
```
tensor([ 2.4333,  1.7574, -0.0710])
```

בשלב הבא עלינו לעדכן את ערכי הפרמטר לכיוון השלילי של הגרדיאנט. נבחר גודל צעד קטן, נעדכן את ערכי הפרמטרים ונצייר שוב את הקו המפריד בין הסיווג של הנקודות.

```
alpha=0.1
dC=calc_dC()
 (w0, w1, b) = torch.tensor((w0, w1, b))- alpha*dC
print(w0, w1, b)
draw_05_line()
```

פלט:

```
tensor(0.7567) tensor(-1.1757) tensor(5.5071)
```



ניכר מהציור שהמודל מסווג נכונה את רוב הנקודות השחורות. עכשיו עלינו לחזור על פעולה זו באופן איטרטיבי:

```
w0, w1, b = 1, -1, 5.5
alpha=0.1

H = lambda y,yt: -(yt*torch.log(y)+(1-yt)*torch.log(1-y))
cost_per_point = H(model(w0, w1, b,X[:, 0], X[:, 1]),Y)

cost=torch.zeros(1000)
for iter_num in range(len(cost)):
  dC=calc_dC()
  params=torch.tensor((w0, w1, b))
  (w0, w1, b) =params - alpha*dC
  cost_per_point = H(model(w0, w1, b,X[:, 0], X[:, 1]),Y)
  cost[iter_num]=cost_per_point.mean()
```

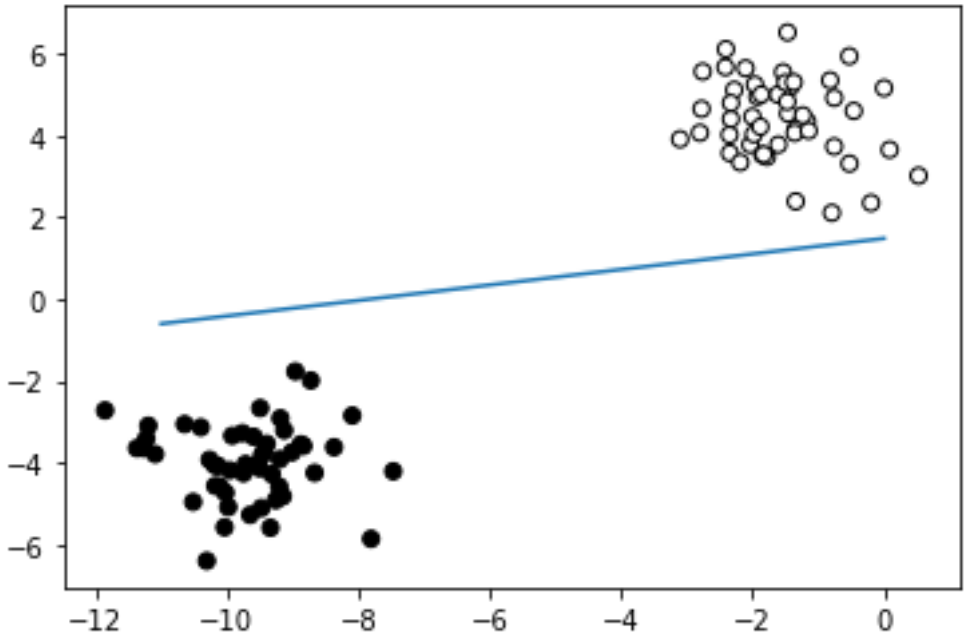
יש לשים לב בכל איטרציה של עדכון הפרמטרים אנו מחשבים את ∇H עבור **כל אחת** מנקודות אוסף הנתונים. כבר כעת כאשר מדובר ב-100 נקודות, זמן הריצה של חישוב זה אינו זניח ובעתיד, כאשר גודל אוסף הנתונים יהיה מאות אלפי ואף מיליוני דגימות מחיר חישוב הגרדיאנט על בסיס כל הדגימות יהיה גבוה מדי. גם בפתרון בעיה זו נדון בהמשך הקורס.

אחרי ריצת הקוד ננתח את התוצאות. ראשית נצייר את הקו המפריד שהתקבל לאחר 1000 איטרציות:

```
draw_05_line()
print(w0, w1, b)
```

פלט:

```
tensor(0.6219) tensor(-3.2859) tensor(4.8640)
```



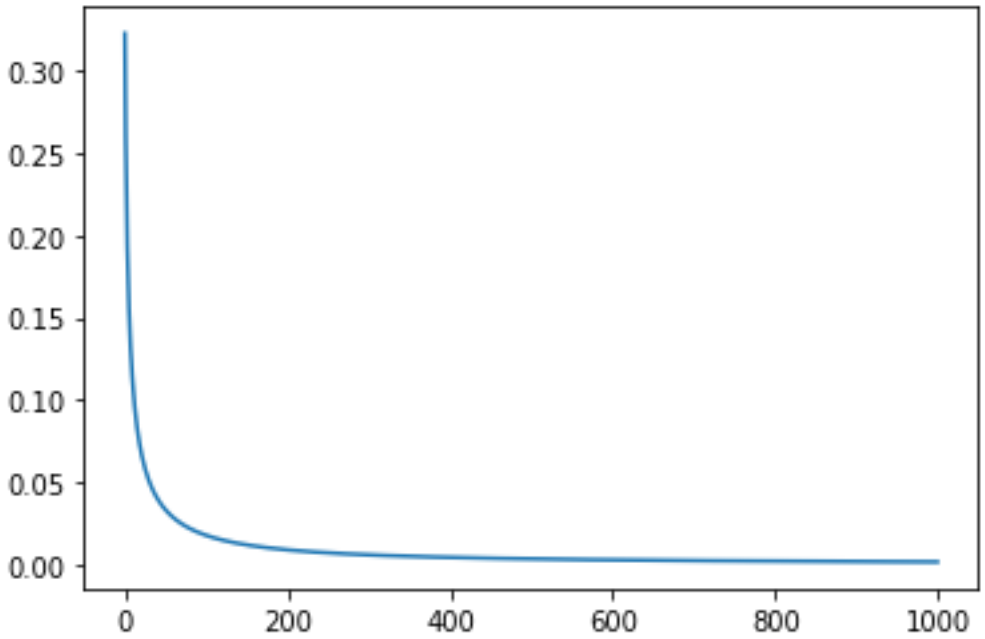
קו זה רחוק מלהיות הקו האידיאלי, שכן הוא עובר קרוב מאוד לנקודות הלבנות ועל כן ייתכן שכאשר נדגום בעתיד נקודות לבנות חדשות הן יסווגו בשוגג כשחורות. האם איטרציות אימון נוספות יעזרו? מקטע הקוד הבא ניתן לראות שלא וזאת מפני ש:

1. האיטרציה האחרונה הורידה את מחיר המודל רק בעשירית אחוז מהערך קודם – שיפור זניח.
2. הגרדיאנט באיטרציה האחרונה אפסי, ועל כן בתוספת איטרציות נוספות, ערכי הפרמטרים כמעט ולא ישתנו.

```
plt.plot(range(len(cost)),cost)
err = abs(cost[-1]-cost[-2])/(cost[-2])
dC=calc_dC()
print(err,dC,sep='\n')
```

פלט:

```
tensor(0.0010)
tensor([0.0002, 0.0040, 0.0015])
```



בכדי לקבל תוצאה טובה יותר, היה כדאי לאתחל את הפרמטרים בערכים קרובים יותר למטרה הרצויה, במקרה שלנו למשל, קו בשיפוע יורד. במקרים מורכבים יותר לא יהיה פשוט כל כך למצוא ערכי פרמטרים התחלתיים מוצלחים, ובכך נדון בהמשך הקורס.

שאלות לתרגול

1. שנו את קצב הלמידה `alpha` (הגדילו אותו והקטינו אותו) באלגוריתם הנ"ל ובדקו האם התוצאות המתקבלות טובות יותר.
2. אתחלו את המודל עם ערכי פרמטרים טובים יותר ובדקו את ביצועיו.
3. בהנתן ערכי הפרמטרים המתקבלים בסוף ריצת האלגוריתם, חשבו את הגרדיאנט ∇H עבור נקודה שחורה אחת ונקודה לבנה אחת, ונסו להסביר למה ערכים אלו כה קטנים, על אף שמודל הסיווג שהתקבל אינו הטוב ביותר.
4. שנו את קוד אימון הנוירון בדרך הבאה: הציבו את ערכי הנגזרות האנליטיים של ∇H במקום להשתמש בחישובי ביניים וכלל השרשרת. השוו את ביצועי שתי הגרסאות איזו לדעתכם עדיפה? האם תוכלו להסביר למה?
5.
   א. כתבו את קוד אימון הנוירון באופן וקטורי.
   הנחיות:
   - הניחו שהקלט הוא נקודות שחורות/לבנות במרחב k מימדי, והחישוב שמבצע הנוירון לחיזוי הסתברות שנקוד נתונה היא שחורה הוא:

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_{k-1} x_{k-1} + b$$

$$y = \frac{1}{1+e^{-z}}$$

- שימו לב שכעת למודל יש k+1 פרמטרים: $w_0, w_1, ..., w_{k-1}, b$. בהתאם, יש לחשב נגזרת של פונקציית המחיר עבור כל אחד מהם, ולעדכן את כולם בכל איטרציה של אלגוריתם האימון.

ב. בדקו את הקוד שכתבתם בסעיף א' על ידי דגימת נקודות שחורות ולבנות במרחב k מימדי (בעזרת שינוי הפרמטר `n_features` של הפונקציה `make_blobs`) ואימון הנוירון לסיווגן.