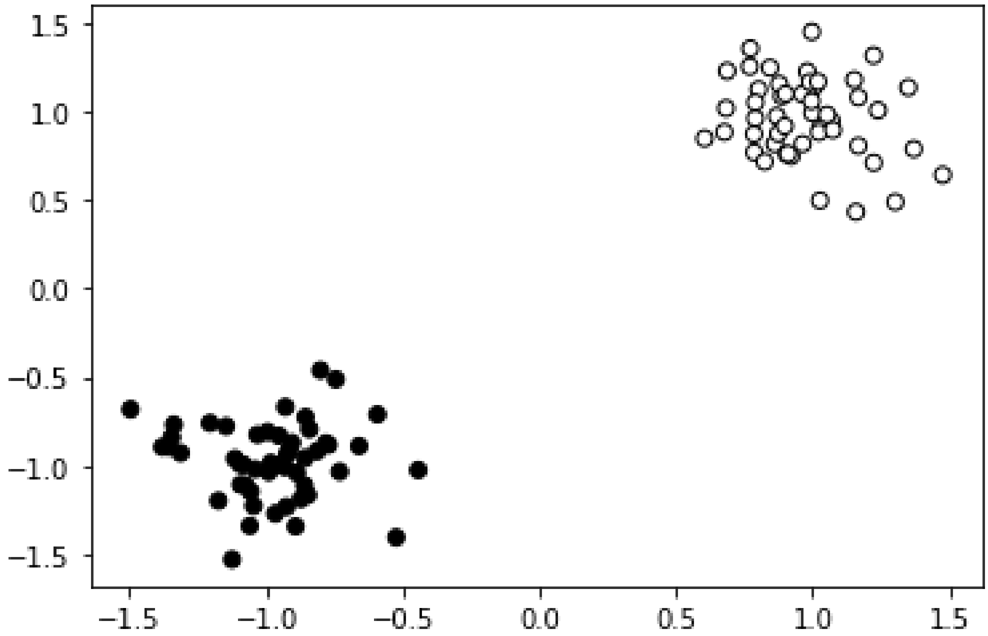
שכבות נורמליזציה

בסוף הפרק הקודם יצרנו באופן מלאכותי בעיה בסט הנתונים על ידי שינוי הסקאלה של אחד ממשתני הקלט לרשת. לבעיה זו יש פתרון הרבה יותר פשוט מהחלפת אלגוריתם האופטימיזציה, והוא תקנון אוסף הנתונים לפני הזנתו לרשת: לאחר ייצור הנקודות נחסיר מכל מימד את הממוצע שלו ונחלק בסטיית התקן, כלהלן.

```
X=(X-X.mean(dim=0))/X.std(dim=0)
print(X.mean(dim=0),X.std(dim=0),sep='\n')
plt.scatter(X[:, 0], X[:, 1],
            c=Y, cmap="Greys", edgecolor="black");
```

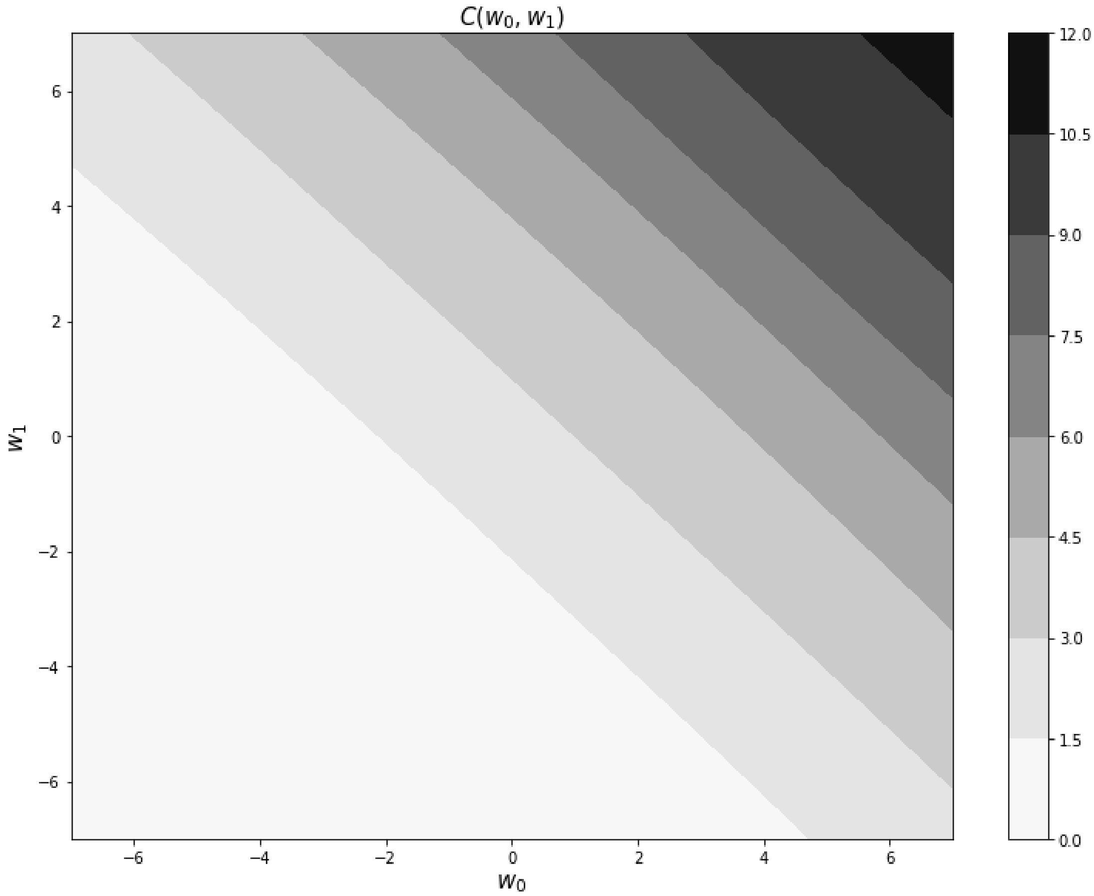
פלט:

```
tensor([-3.5527e-17,  6.6613e-18], dtype=torch.float64)
tensor([1.0000, 1.0000], dtype=torch.float64)
```



זכרו שהממוצע וסטיית התקן הן מתודות רדוקציה, והפרמטר `dim` מנחה אותן לאורך איזה מימד יש לבצע את הפעולה.

כעת, אם נסתכל על משטח המחיר של נוירון הסיווג כפונקציה של הפרמטרים w_0, w_1 (שוב נקבע $b=5$), נראה את התוצאה הבאה,



ומאיור זה ניכר שכל אלגוריתם אופטימיזציה יצליח להתמודד בהצלחה עם פונקציית מחיר זו, שכן הגרדיאנט שלה מצביע לאותו כיוון בכל נקודה. כאן מתחילה ומסתיימת העבודה כאשר מדובר בנוירון בודד, אך עבור רשת עמוקה הקלט לרשת הוא הקלט לשכבה הראשונה בלבד. לאחר מעבר הקלט בשכבה זו סביר שהפלט שלה, הוא הקלט לשכבה הבאה, כבר לא יהיה מתקונן. על אחת כמה וכמה לאחר מספר שכבות. שיקול זה הוביל לתוספת של שכבות נורמליזציה (Normalization Layers) לרשתות נוירונים מודרניות, דבר אשר ייעל בצורה משמעותית את תהליך אימון הרשתות, בין השאר משום שהוא מאפשר שימוש בקצב למידה גבוה יותר.

הדוגמה הראשונה והפופולרית ביותר לשכבת נורמליזציה היא Batch Normalization (BN), אשר מקבלת כקלט מהשכבה הקודמת ברשת טנזור $X=\left(x_1,...,x_d\right)$ ועבור קואורדינטה כלשהי, x_i, בטנזור זה היא מבצעת את החישוב בשני שלבים:

1. ראשית, מחושבים הממוצע והשונות של x_i **על פני ה-minibatch הנוכחי** (מכאן מגיע שם השכבה), ובעזרתם מתקננים את הערך של x_i:

$$\mu=\frac{1}{N}\sum_{k=1}^{N}x_i^{[k]}$$

$$\sigma^2=\frac{1}{N}\sum_{k=1}^{N}\left(x_i^{[k]}-\mu\right)^2$$

$$\hat{x}_i^{[k]}=\frac{x_i^{[k]}-\mu}{\sqrt{\sigma^2+\epsilon}}$$

שימו לב שעל מנת להדגיש את תלות החישוב ב-minibatch מסומנים העותקים השונים של x_i, אשר חושבו במקביל על סמך הקלטים השונים ב-minibatch, עם סוגריים מרובעים. את גודל ה-minibatch אנו מסמנים ב- N וכרגיל, למכנה אנו מוסיפים ערך קטן, כדי להימנע מחלוקה באפס. לרוב נהוג לבחור $\epsilon=10^{-5}$.

2. שנית, ייתכן שאקטיבציות מתוקננות אינן דווקא האופטימליות עבור המשך הרשת, ולכן הפלט הסופי של השכבה הוא

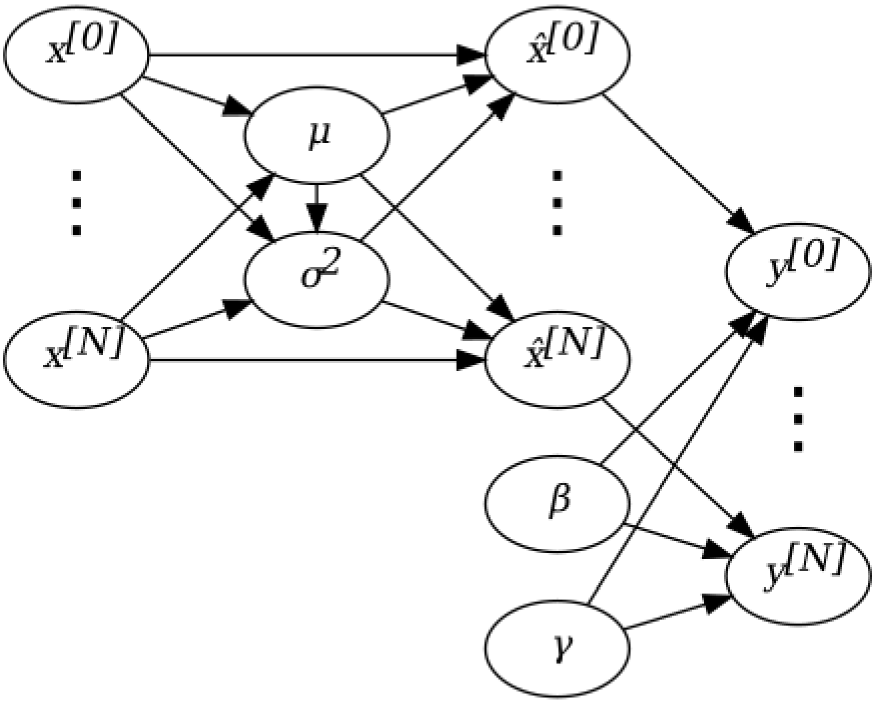
$$y_i^{[k]} = \gamma_i \hat{x}_i^{[k]} + \beta_i$$

כאשר β_i, γ_i הם פרמטרים אשר יילמדו בתהליך האימון, אך מאותחלים כך שראשית מתקיים $y_i = \hat{x}_i$.

יש לשים לב בנוסף לפרטים הבאים:

1. החישוב בשכבת BN מתבצע במקביל ובאופן בלתי תלוי עבור כל קואורדינטה של X.
2. לכל קואורדינטה של X יש בשכבה פרמטרים שני שונים, אשר נלמדים ללא תלות באחרים.
3. השכבה יוצרת תלות בין הדגימות השונות אשר נמצאת ב-batch, ובהתאם הפלט שלה עבור קלט כלשהו יהיה תלוי בשאר הקלטים הנמצאים ב-batch שלו.

שלבי החישוב בשכבת BN מאוירים להלן,



נממש שכבה זו ב-PyTorch, ודרך תהליך זה נלמד כיצד לכתוב שכבות חדשות. המחלקה הבסיסית של שכבה היא `nn.Module`, והשכבה שניצור תירש ממנה, ובשורת קוד:

```
class BN(nn.Module):
```

כעת, צריך להגדיר את הבנאי של השכבה שלנו. הוא יקרא לבנאי של `nn.Module`, ואחרי כן יאתחל את הפרמטרים γ, β, אשר מהם יש לנו עותק לכל קואורדינטה של טנזור הקלט. את גודל הקלט לשכבה הבנאי מקבל ב-`in_features`, בדומה לשכבות הליניאריות שבהן השתמשנו בעבר.

```
class BN(nn.Module):
  def __init__(self,in_features):
      super().__init__()
      self.gamma=nn.Parameter(torch.ones(in_features))
      self.beta=nn.Parameter(torch.zeros(in_features))
```

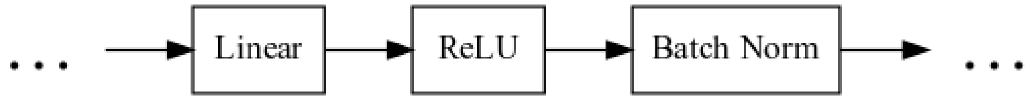
אנו משתמשים בפונקציה `nn.Parameter` בכדי להצהיר על משתנים אלו כפרמטרים של הרשת, כך שאובייקט האופטימיזציה יידע לעדכן אותם בתהליך אימון הרשת.

אחרי כן, כל שנשאר הוא להגדיר את פלט השכבה – אותו מחשבים במתודה `forward()`. ראו להלן שהממוצע והשונות מחושבים לאורך המימד 0 של טנזור הקלט, זהו מימד ה-batch באופן מקובל, ובכל הרשתות שמימשנו עד כה. שאר החישוב מתבצע בעזרת שידור לאורך כל הדגימות ב-minibatch הנתון במשתנה `input`.

```
def forward(self,input):
  mu=input.mean(dim=0)
  sigma2=input.var(dim=0)
  xhat=(input-mu)/torch.sqrt((sigma2)+10**-5)
  y=self.gamma*xhat+self.beta
  return y
```

לאחר ששכבה זו הוגדרה, ניתן לשלבה ברשת נוירונים ככל שכבה מובנית של `torch.nn`. נהוג לשלב שכבה זו בין שכבה ליניארית לאקטיבציה כמאויר להלן,

או מיד לאחר האקטיבציה,



אין אנו נדרשים לממש את המעבר אחורה דרך שכבה זו, שכן מערכת ה-Autograd עוקבת אחרי כל החישובים המבוצעים במעבר קדימה ועל כן המעבר לאחור יתבצע באופן אוטומטי. יחד עם זאת, יש ערך בחישוב האנליטי של הפעפוע לאחור: דרכו ניווכח שחישוב היוצר תלויות בין דגימות שונות מאוסף הנתונים אינו בהכרח בעייתי מבחינת אלגוריתם האופטימיזציה. ניגש אם כן למשימה. פלט השכבה הוא Y ומימדיו כמימד טנזור הקלט X. נניח שבידינו גרדיאנט פונקציית המחיר ביחס לפלט $\frac{\partial C}{\partial Y}$, וכעת עלינו לחשב את $\frac{\partial Y}{\partial \gamma},\frac{\partial Y}{\partial \beta}$ בשביל עדכון ערכי הפרמטרים של שכבה זו, וכן את $\frac{\partial Y}{\partial X}$

בכדי לפעפע את הנגזרת הלאה. ראשית נשים לב שכל הנגזרות מהצורה $\frac{\partial y_i^{[k]}}{\partial x_m^{[n]}},\frac{\partial y_i^{[k]}}{\partial \gamma_m},\frac{\partial y_i^{[k]}}{\partial \beta_m}$ כאשר $i \neq m$ מתאפסות, שכן עבור כל קואורדינטה של הקלט X מתבצע חישוב בלתי תלוי בקואורדינטות האחרות. שנית, באופן מיידי לפי הנוסחה $y_i^{[k]} = \gamma_i \hat{x}_i^{[k]} + \beta_i$ מתקיים

$$\frac{\partial y_i^{[k]}}{\partial \beta_i} = 1$$

$$\frac{\partial y_i^{[k]}}{\partial \gamma_i} = \hat{x}_i^{[k]}.$$

כעת, נחשב את הנגזרות המיידיות של כל קודקוד לפי כל אב מיידי שלו בגרף החישוב לעיל:

$$\hat{x}_i^{[k]}:\ \frac{\partial y_i^{[k]}}{\partial \hat{x}_i^{[k]}}=\gamma_i$$

$$\sigma^2:\ \frac{\partial \hat{x}_i^{[k]}}{\partial \sigma^2}=\left(x_i^{[k]}-\mu\right)\cdot\frac{-1}{2}\left(\sigma^2+\epsilon\right)^{-3/2}$$

$$\mu\ :\ \frac{\partial \hat{x}_i^{[k]}}{\partial \mu}=\frac{-1}{\sqrt{\sigma^2+\epsilon}},\quad \frac{\partial \sigma^2}{\partial \mu}=\frac{1}{N}\sum_{k=1}^{N}-2\left(x_i^{[k]}-\mu\right)$$

$$x_i^{[k]}:\ \frac{\partial \hat{x}_i^{[k]}}{\partial x_i^{[k]}}=\frac{1}{\sqrt{\sigma^2+\epsilon}},\quad \frac{\partial \sigma^2}{\partial x_i^{[k]}}=\frac{2\left(x_i^{[k]}-\mu\right)}{N},\quad \frac{\partial \mu}{\partial x_i^{[k]}}=\frac{1}{N}$$

כל שנשאר הוא לחבר את הנגזרות המיידיות בעזרת שימוש זהיר בכלל השרשרת – יש לגזור אב מיידי לפי כל אחד מבניו, ולחבר את התוצאה. למשל, עבור $x_i^{[k]}$,

$$\frac{\partial C}{\partial x_i^{[k]}}=\frac{\partial C}{\partial \mu}\frac{\partial \mu}{\partial x_i^{[k]}}+\frac{\partial C}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial x_i^{[k]}}+\frac{\partial C}{\partial \hat{x}_i^{[k]}}\frac{\partial \hat{x}_i^{[k]}}{\partial x_i^{[k]}}.$$

התלות הנוצרת בין הדגימות השונות ב-batch מועילה למטרת האופטימיזציה, אך אינה רצויה כאשר הרשת כבר אומנה: אנו לא רוצים שהחיזוי עבור דגימה נתונה ישתנה בהתאם לשאר הדגימות המוזנות במקביל אליה ברשת. לכן, בעת החיזוי נשתמש בהערכות אלטרנטיביות עבור μ ו- σ^2.
התבוננו שוב בגרף החישוב לעיל והיווכחו שהדגימות השונות ב-batch תלויות זו בזו רק דרך ערכים אלו. מכאן שניתוק התלות שלהם ב-batch יוביל לחישוב דטרמיניסטי: החישוב עבור כל דגימה יהיה בלתי תלוי בדגימות אחרות. אחת הדרכים המקובלות לעשות זאת היא לשמור בעת האימון ממוצע רץ של ערכים אלו, בדומה לנעשה בשיטת המומנטום עבור וקטור המהירות. כאשר נרצה לעבור לחיזוי, נציב ממוצעים אלו במקום של μ,σ^2. שאר החישוב יבוצע באופן זהה.

שכבת BN היא הרכיב הראשון של רשתות נוירונים בעל התנהגות שונה בעת אימון ובעת חיזוי בו אנו נתקלים, אך נכיר עוד רכיבים כאלו בהמשך ונשתמש בהם לרוב. בכדי להקל על המעבר בין אימון לחיזוי קיימות ב-PyTorch המתודות `model.train()` ו- `model.eval()`, אשר מעבירות את הרשת בין המצבים. אם ברצוננו להתנות את החישוב המתבצע בשכבה מסויימת בהיותה במצב אימון/חיזוי, נוסיף תנאי למתודת החישוב, כדלקמן.

```
def forward(self,input):
  if self.training:
    .
    .
    .
  else:
    .
    .
    .
```

Batch Normalization אינה שכבת הנורמליזציה היחידה בשימוש נפוץ, וקיימות נוספות כגון Layer Normalization או Group Normalization בהן משתמשים בקואורדינטות השונות (בכולן, או בחלקן, כתלות בשיטה) בטנזור הקלט לשכבה בכדי לנרמל את האקטיבציות. מכיוון ששיטות אלו אינן יוצרות תלות בין דגימות שונות ב-minibatch, הן לרוב עדיפות על BN כאשר גודל ה-batch קטן, שכן אז הפרמטרים μ ו- σ^2 המחושבים בכל איטרציה אינם יציבים – ערכם משתנה בהתאם למעט הדגימות אשר נמצאות באותה העת בזכרון. השיטות הפופולריות מומשו כשכבות סטנדרטיות ב-`torch.nn`.

בעוד שאין ספק בקרב הקהילה המדעית ששכבות נורמליזציה חיוניות לאימון רשת בצורה אפקטיבית, הסיבות לכך הן נושא לדיון ער, ובהחלט אפשרי הדבר שהאינטואיציה המקורית שהנחתה את פיתוחן, נרמול הקלט בכל שכבה ברשת אינה הסיבה העיקרית לכך. ייתכן שבדרכים אחרות, שאינן ידועות לנו כעת, שכבות אלו הופכות את משטח המחיר לפשוט יותר עבור אלגוריתמי אופטימיזציה מבוססי גרדיאנט, שכן המחקר בנושא ממשיך להתפתח עוד בימינו.

שאלות לתרגול

1.
   א. צרו רשת עמוקה המשרשרת שכבות ליניאריות ושכבות ReLU בזו אחר זו, כך שהקלט שלה הוא דו מימדי, ואמנו אותה לסווג אוסף נתונים הנוצר בעזרת הפונקציה `make_moons`.
   ב. הזינו לתוכה את נתוני הנקודות השחורות והלבנות, לאחר נרמול. שכבה אחר שכבה, ושמרו את **הפלט של כל שכבה** במשתנה חדש.
   ג. חשבו את הממוצע וסטיית התקן של הפלט של כל זוג שכבות Linear -> ReLU. שימו לב שיש לבצע את הרדוקציות על מימד ה-batch.
   ד. ציירו את התוצאה בגרפים: בציר ה-X, מיקום השכבה ברשת. בציר ה-Y התוחלת/סטיית התקן של כל נוירון בשכבה זו. **הערה:** מומלץ להשתמש למטרה זו בפקודה `boxplot` של הספרייה matplotlib.
2. חזרו על שאלה 1 לאחר הוספת שכבות נורמליזציה לרשת הנוירונים. זכרו להעביר את הרשת למצב חיזוי לאחר סעיף א'.
3. השלימו את חישוב הפעפוע לאחור דרך שכבת BN: בטאו את $\frac{\partial C}{\partial x_i^{[k]}}$ בעזרת ערכים שחושבו במעבר קדימה ברשת ו- $\frac{\partial C}{\partial y_i^{[k]}}$ בלבד. פשטו את הביטוי ככל האפשר.
4. הוסיפו למחלקה BN המוגדרת לעיל מצב חיזוי בדרך הבאה:
   a. שמרו משתנים פרטיים בתוך האובייקט עבור הממוצעים הרצים של μ, σ^2.
   b. בכל מעבר קדימה בשכבה, כאשר מצב אימון מופעל, יש לעדכן את הממוצע של μ לפי הנוסחה $\mu_{avg} = 0.9\mu_{avg} + 0.1\mu$, כאשר μ חושב על בסיס ה-batch הנוכחי.
   בדומה יש לעדכן את σ^2. שימו לב שמשתנים אלו לא משתתפים בתהליך האימון ולכן אין לעקוב אחריהם בעזרת מערכת ה-Autograd.
   c. כאשר מצב חיזוי מופעל, יש להשתמש ב- $\mu_{avg}, \sigma_{avg}^2$ במקום μ, σ^2.
5. שימוש ב-BN דורש minibatch המכיל לפחות שתי דגימות. הסבירו דרישה זו.